1. Tekoälyn perusteista [RN95, osa I][Nil98, luku 1]

Tekoäly on *uusi* tieteenala: nimensä se sai 1956. Se syntyi vain noin 10 vuotta ensimmäisten yleiskäyttöisten tietokoneiden jälkeen, joten se on yksi vanhimpia alueita *tietojenkäsittelytieteen sisällä*.

Se on samalla *vanha* tieteenala: yli 2000 vuotta on pohdittu, miten havaitsemista/oppimista/muistamista/päättelyä/. . . tehdään/tulisi tehdä.

Toisin kuin vanhemmat sisarensa *filosofia* ja *psykologia*, tekoäly pyrkii *rakentamaan* älykkäitä olentoja/yksikköjä/koneita/. . . eikä (pelkästään) ymmärtämään itse älykkyyttä.

Tämän luvun eri asiat liittyvät monin eri tavoin toisiinsa, joten se pakostikin sisältää runsaasti ristiviittauksia.

1.1. Käsitteen "tekoäly" merkityksiä

Jotta tekoälystä voi "filosofoida", pitää ensin määritellä, mitä sanalla tarkoittaa.

Eri määritelmät johtavat erilaisiin tutkimustavoitteisiin ja -menetelmiin.

Eräs mahdollinen jako saadaan tekemällä yhtä aikaa kaksi erottelua: *ajatella/toimia* ja *inhimillisesti/rationaalisesti*. Näin syntyy nelikenttä:

"The exciting new effort to make computers think … *machines with minds*, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning …" (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Taulukko 1: 4 käsitystä tekoälystä [RN95, kuva 1.1].

Eri lokeroiden väliset raja-aidat eivät ole tiukkoja:

- Kummankin toiminnan painottaja edellyttää (tietenkin) myös kykyä vastaavaan ajatteluun.
- Ajattelun motivaationa on kääntäen tarve myöhemmin toimia.
- Ainakin toivoisi, että inhimillisyys ja rationaalisuus olisivat edes kaukaista sukua toisilleen. . .

Lokeroinnissa onkin kysymys siitä, mitä pitää *itse tavoitteen saavuttamisena* ja mitä sitä edeltävänä/seuraavana lisätyönä.

1.1.1. ”tekoäly” = ”toimia inhimillisesti”

Taulukon 1 vasemman yläkulman tunnetuin muotoilu on *Alan Turingin (1912-1954)* vuonna 1950 esittämä ”oikeussalidraama”, jossa konetta syytetään ”väärennetyillä henkilöllisyyspapereilla esiintymisestä”:

syytetyistä A ja B toinen on ihminen ja toinen ihmistä teeskentelevä kone M.

ihmistuomarin C tehtävänä on päättää kumpi on kumpi. Hän esittää valitsemiaan kysymyksiä syytetyille ja vastausten perusteella julistaa tuomionsa.

”Oikeus on sokea”: tuomarilla C ei ole muuta informaatiota (tuntoa, näköä, hajua,. . .) syytetyistä kuin kysymysten kirjalliset vastaukset.

Konetta on pidettävä älykkäänä, jos C erehtyy tuomiossaan.

Tai tilastollisesti: tuodaan sama kone ”rikoksenuusijana” yhä uudelleen saman C eteen eri ihmisten kanssa, ja kutsutaan sitä älykkääksi, mikäli yhteenlaskettu virheiden määrä on samaa luokkaa kuin jos C olisi kuulustelujen sijaan heittänyt kruunaa ja klaavaa.

periaate: kone on ”älykäs” jos sen *syöte-tuloste -käyttäytymistä* ei voida erottaa luotettavasti ihmisen vastaavasta.

Turing ottaa näin *funktionalistisen* kannan älykkyyteen. Myöhemmät tämän kannan edustajat ovat usein siirtyneet tarkastelemaan kokonaisuuden sijasta sen *osien* käyttäytymistä esimerkiksi väittämällä, että vain yksittäisten neuronien (hermosolujen) syöte-tulossignaalit merkitsevät, ei niiden välittämisessä käytetyt biokemialliset mekanismit.

Voisiko funktionalismin hylätä rajaamatta ”älykkyyttä” samalla vain tietynlaisiin eliötyyppeihin (joita saattaa esiintyä vain maapallolla)?

1.1.2. "tekoäly" = "ajatella inhimillisesti"

Ihmisen mielen
miten kuvaisit? Se on
tushipiirroksen
mäntyjen oksistossa
tuulevan tuulen ääni.

Ikkyū (suom. Tuomas Anhava)

Rajusti kärjistäen taulukon 1 vasenta alakulmaa, *kognitiotieteen (Cognitive Science)* menetelmä älykkyyden mallintamiseen on seuraava:

1. Kerätään ihmisryhmältä I *kokeellisen psykologian* menetelmin havaintoaineistot H ja H' sen jäsenten käyttäytymisestä jossakin "älykkyyttä vaativassa" tehtävässä T.
2. Rakennetaan sellainen *laskennallinen malli (ohjelma)* M, joka antaa (likimain) samat tulokset kuin lähde- tai opetusaineisto H.
3. Testataan, antaako M myös (likimain) samat tulokset kuin testiaineisto H'.
4. Jos antaa, niin tämä tukee seuraavia hypoteeseja:
   (a) M ratkaisee tehtävän T "älykkäästi", käyttäytyyhän se samaan tapaan kuin joukon I jäsen; ja kääntäen
   (b) M:n laskennalliset operaatiot muistuttavat niitä mentaalisia operaatioita, joita joukon I jäsen käyttää ratkoessaan tehtävää T.

periaate: kone on "älykäs" jos siinä tapahtuu *samankaltaista informaationkäsittelyä* kuin ihmismielessä.

Luvun 1.1.1 Turingin testi -määritelmään verrattuna ei "onnistumisen" kriteeri ole enää yhtä selvä, mutta toisaalta nyt päästään vetoamaan mallin/koneen M *sisäiseen* rakenteeseen ja toimintaan.

1.1.3. "tekoäly" = "ajatella rationaalisesti"

> **logic** *n.* The art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding.
>
> *A. Bierce*

Filosofiassa ja matematiikassa on tutkittu paljon *logiikkaa*, "oppia oikeasta ajattelusta": sääntöjä, joita järjestelmällisesti seuraamalla ajatteluun ei pääse putkahtamaan virheitä.

Esimerkiksi *Aristoteleen (384–322 eKr.) syllogismit* kuten

$$\text{Sokrates on ihminen.} \tag{1}$$

$$\text{Kaikki ihmiset ovat kuolevaisia.} \tag{2}$$

$$\text{Siispä Sokrateskin on kuolevainen.} \tag{3}$$

tuntuvat "järkeviltä": jos olen jo vakuuttunut väitteistä (1) ja (2), niin johdonmukaisuuden nimissä minun täytyy silloin hyväksyä myös johtopäätös (3). Minun täytyy jopa hyväksyä kaikki *saman muotoiset* perustelut, joissa 'Sokrates', 'ihminen' ja 'kuolevainen' korvataan muilla sanoilla.

periaate: kone M on "älykäs" jos se maailman tilan ja säännönmukaisuuksien kuvauksesta pystyy *päättelemään* onko annettu tavoite saavutettavissa (ja jos, niin miten).

Toisin kuin lukujen 1.1.1 ja 1.1.2 määritelmissä, taulukon 1 oikeassa yläkulmassa pyritään ihmisen jäljittelyn sijasta "oikeaan lopputulokseen". Tämä kuitenkin määritellään koneen M *ulkopuolisin kriteerein* U, kun taas aiemmin tavoite "olla kuin ihminen" oli "sisäistetty".

Kun tavoite on ulkoa annettu, voi "tietämysinsinööri" *(knowledge engineer)* alkaa *suunnittelemaan ja rakentamaan* sen toteuttavaa konetta, jota sitten voisi väittää älykkääksi.

1.1.4. "tekoäly" = "toimia rationaalisesti"

Kun vaaditaan vielä luvun 1.1.3 koneelta M *toimintaa* muodostamiensa rationaalisten ajatusten mukaisesti, saadaan oppikirjojemme [Nil98, RN95] kanta, jonka mukaan tekoäly on *älykkäiden toimijoiden (intelligent agents)* suunnittelemista ja rakentamista.

Toimija (agent) on mikä tahansa erillinen yksikkö A, jonka voidaan ajatella tekevän *havaintoja (percept)* ympäröivän maailman *tilasta (state)* sekä *teoillaan (action)* vaikuttavan siihen.

Informaation kerääminen havainnoilla on sekin tekemistä!

Suorituskykymitta (performance measure) U on luvussa 1.1.3 mainittu toimijan A ulkopuolinen mittari sen (pitkän aikavälin) toiminnan tuloksellisuudelle. Ajatellaan se (numeerisena) funktiona *ympäröivän maailman eri tiloille* S: siivousrobotilla $U(S) < U(S')$, jos S' on puhtaampi maailma kuin S.

Esimerkillinen rationaalinen toimija (ideal rational agent) A_{ert} valitsee jokaisessa tilassa S_{nyt} sellaisen mahdollisen teon T_{nyt}, joka maksimoi suorituskykymitan U *odotusarvon (expected value)*.

Odotusarvon, koska (esimerkiksi autoilevan) toimijan A_{ert}

- lisäksi voi olla *muita* toimijoita
  "Tuleekohan tuolta sivutieltä ketään eteeni?"
- *tieto* nykytilasta S_{nyt} tai tekojen T_{nyt} vaikutuksista voi olla epävarmaa
  "Onkohan tie liukas?"
  "Kiihtyyköhän autoni tarpeeksi ohitukseen?"
- tekojen T_{nyt} *onnistuminen* voi olla epävarmaa
  "Entäs jos pyörät sutivatkin tyhjää kun kaasutan?"

Liikennesääntö "risteysmerkin jälkeen ei saa enää ohittaa" voidaankin nähdä laskutoimituksen *yksinkertaistuksena* karsimalla vaihtoehtoja.

Valinnan T_{nyt} pohjana A_{ert} käyttää

- sisäänrakennettua tietoa

  Liikennesäännöt, . . .
- ennen nykytilaa S_{nyt} keräämäänsä *havaintojonoa (percept sequence)* H_{nyt}.

  Onko auto liukastellut tänään aiemmin? Miten auto on ennen kiihtynyt? . . .

Jonon H_{nyt} perusteella A_{ert} muodostaa oman käsityksensä siitä, millainen S_{nyt} on.

A_{ert} on *itseohjautuva (autonomous)* siltä osin, kuin sen käyttäytymisen on sen aiempien kokemusten H_{nyt} määräämää.

Kellokin on esimerkillinen rationaalinen toimija – jos tavoitteena on vain pyörittää viisareita tahdissa antamatta minkään häiritä niiden kulkua. Se ei havainnoi ympäristöään eikä siis ole lainkaan itseohjautuva, vaan sen "rationaalisuus" on kokonaan kellosepän siihen istuttamaa.

A_{ert} ei saa ummistaa silmiään *olennaiselta* informaatiolta: ennen ohitusta kannattaa katsoa taustapeiliin, ettei käänny takaa tulevan ohittajan eteen – maantieajossa ohituskolarin U-arvo ei liene järin suuri. . .

. . . mutta vuoden 1997 viimeisessä Formula 1 -osakilpailussa Schumacher *toimi järkevästi* kiilatessan Villeneuvea vastaavassa tilanteessa:

Villeneuve pisteittä: Schumacher on mestari.

Villeneuve ohittaa: Schumacher ei ole mestari.

- $U_1 = U_{\text{Schumi}}(\text{Schumacher päästää Villeneuven ohitseen}) \ll 0.$
- $U_2 = U_{\text{Schumi}}(\text{Schumacher kiilaa Villeneuven(kin) ulos}) \gg 0.$
- $U_3 = U_{\text{Schumi}}(\text{Schumacher kiilaa Villeneuven taakseen}) > 0.$

- $U_4 = U_{\text{Schumi}}(\text{Schumacher kiilaa vain itsensä ulos}) \ll 0$.
- Todennäköisyys $P_1 = P(\text{Schumacher kiilaa Villeneuven ulos})$ on hyvin suuri, onhan hän asiansa osaava kuski. . .
- Samoin $P_2 = P(\text{Schumacher kiilaa itsensä ulos})$ on hyvin pieni.

Silloin kiilaamisen odotusarvo on

$$P_1 \cdot U_2 + (1 - P_1) \cdot (1 - P_2) \cdot U_3 + \underbrace{(1 - P_1) \cdot P_2}_{\text{hyvin, hyvin pieni}} \cdot U_4$$

mikä lienee (paljon) suurempi kuin U_1.

Sillä kertaa kuitenkin tuo pessimaalinen vaihtoehto toteutui.

Vaikka jokaisella A_{ert} onkin *oma yksityinen* U, voi kokonaisuus silti maksimoida *yhteistä hyvää*: valitaan yhteiset (liikenne)säännöt siten, että yhteinen hyvä on kannattavin vaihtoehto jokaiselle. Tätä on esim. talous- ja yhteiskuntatieteissä käytetty ns. *mekanismisuunnittelu (mechanism design)*.

A_{ert} ei toisaalta saa jäädä odottelemaan *epäolennaista* informaatiota: jos taustapeilistä näkyy toinen auto, mutta se ei ole vielä laittanut ohitusvilkkua päälle, ei jäädä odottelemaan sitä, vaan ohitetaan itse.

Tekoälyn tehtäväksi nähdään nyt *tietokoneen ohjelmoiminen laskemaan funktio* $A_{\text{ert}}(H_{\text{nyt}}) = T_{\text{nyt}}$.

```
function SKELETON-AGENT(percept) returns action
   static: memory, the agent's memory of the world

   memory ← UPDATE-MEMORY(memory, percept)
   action ← CHOOSE-BEST-ACTION(memory)
   memory ← UPDATE-MEMORY(memory, action)
   return action
```

Algoritmi 1: Älykkään toimijan perusohjelma [RN95, kuva 2.4].

Mikään ei kuitenkaan takaa, että tämä funktio (aliohjelma CHOOSE-BEST-ACTION) olisi mekaanisesti laskettavissa. Muutetaan siis tavoite *täydellisestä (perfect) laskennalliseksi (calculative)* rationaalisuudeksi, ja edelleen käytännössä *rajoitetuksi optimaalisuudeksi (bounded optimality, BO)*, jossa tätä funktiota approksimoidaan mahdollisimman hyvin ylittämättä ennalta annettuja laskentaresursseja. Toimija on siis BO-mielessä älykäs, jos se käyttää sille suodun harkinta-ajan mahdollisimman tehokkaasti, keskittyy olennaiseen.

1.2. Tekoäly ja matematiikka

Logiikka alkoi jo antiikista (luku 1.1.3), mutta vasta 1800-luvun loppupuolelta alkaen se laajeni filosofiasta (meta)matematiikkaan. Haluttiin tarkka muotoilu *todistuksen käsitteelle*: jono väitteitä. . .

1. jotka on esitetty niin täsmällisessä kielessä, ettei yhtäkään niistä voi tulkita kuin yhdellä tavalla; ja
2. jossa jonossa seuraava väitteen voidaan aina nähdä olevan välttämätön edellisten nojalla – kuten (3) väitteistä (1) ja (2).

Tämä oli keskeinen osa *David Hilbertin (1862–1943)* ohjelmaa saattaa matematiikka järkkymättömälle perustalle. Vuonna 1900 hän listasi 23 matematiikan keskeistä avointa ongelmaa, joista viimeinen oli *Entscheidungsproblem*: voidaanko *luonnollisia lukuja koskevien loogisten väitteiden totuus* ratkaista mekaanisesti?

George Boole (1815–64) oli jo 1847 yrittänyt "matematisoida logiikan" antamalla totuusarvoille laskulait lukuina 0 ja 1. Tästä syntyi myöhemmin *lauselogiikka*.

Gottlob Frege (1848–1925) kehitti 1879 *ensimmäisen kertaluvun predikaattilogiikan (First Order Predicate Calculus)*. Sen pohjalta hän yritti 1893–1903 "logisoida matematiikan". *Bertrand Russell (1872–1970)* osoitti kuitenkin 1902 menetelmässä virheen *paradoksillaan*: jos määritellään $P = \{Q : Q \notin Q\}$, niin sekä $P \in P$ että $P \notin P$ ovat mahdottomia.

Russell ja *Alfred N. Whitehead (1861–1947)* korjasivat ongelman *tyyppiteorialla*: P ei olekaan joukko, vaan "luokka". Heidän teossarjansa *Principia Mathematica (1910–13)* lienee kuuluisin yritys saavuttaa tavoite 1.

Tämän osuuden täydensi *Alfred Tarski (1902–83)* vuonna 1933 antamalla yleiset periaatteet, joilla tämän kaltaisten kielten totuus voidaan määritellä.

Tavoite 2 johti puolestaan **laskettavuuden** käsitteeseen: jo *Gottfried W. von Leibniz (1646–1716)* oli haaveillut kielestä, jossa kiistat voitaisiin ratkaista laskemalla. Piti siis määritellä intuitiivinen "laskentasäännön" tai *algoritmin* idea (joka tunnettiin jo antiikissa, ja on nimetty arabimatemaatikko *al-Khwārizmī*n mukaan, jonka *Kitāb al-jabr wa al-muqābalah* vuodelta 825 toi länsimaihin algebran). Tavoite siis oli, että jokainen todistuksen askel voitaisiin *tarkistaa* mekaanisesti.

Ensiksi vuonna 1930 *Kurt Gödel (1906–78)* edisti Hilbertin ohjelmaa huomattavasti *predikaattilogiikan täydellisyyslauseellaan*, jonka mukaan tälle kielelle oli todellakin olemassa

- todistusmenetelmä jolla voi todistaa jokaisen totuuden
- eli mekaaninen todistuksen tarkistaja K.

Tästä logiikasta puuttuu kuitenkin *induktioperiaate*, joten entscheidungsproblem jäi edelleen auki.

Gödel ratkaisi ensin tavoitteen 2 abstraktissa matemaattisessa mielessä *rekursiivisten funktioiden teoriana*. Vuonna 1936 luvussa 1.1.1 mainittu Alan Turing esitti Gödelin työn kanssa yhtäpitävän laskettavuuskäsityksen, joka kuitenkin perustui (periaatteessa) rakennettavissa oleviin mekaanisiin laitteisiin. Tämä *Turingin kone* tuli sittemmin johtavaksi tietokoneen teoreettiseksi muotoiluksi.

Sitten vuonna 1931 Gödel osoitti koko ohjelman mahdottomaksi *epätäydellisyyslauseellaan*, jonka mukaan jos kielessä L voidaan esittää luonnollisten lukujen aritmetiikka, niin siinä voidaan myös lausua tosia väitteitä, joita ei kuitenkaan voida todistaa. Näin Gödel vastasi Hilbertin 23. ongelmaan "ei".

Gödelin päättelyn idea oli, että kielellä L voidaan silloin esittää myös lainauksia sekä algoritmeja (käyttäen eräänlaista salakirjoitusta, nk.

"Gödel-numerointia"). Olkoon sitten C mikä tahansa algoritmi kielen L todistusten tarkistamiseen. Silloin kielellä L voidaan sanoa *valehtelijan paradoksi* "minä olen väite, jonka todistusta C ei pysty tarkistamaan". Turing käytti myöhemmin samaa periaatetta osoittaessaan koneidensa *pysähtymisongelman* ratkeamattomaksi.

Yritys perustaa matematiikka formaaliin logiikkaan siis kariutui. Tekoäly kuitenkin otti sen välineet käyttöönsä:

- Tietämys maailman tilasta pyritään esittämään *kuvailevasti* sitoutumatta ennalta mihinkään tiettyyn käyttöön.

  Matemaattista aksioomaa saa käyttää todistuksessa sen oletuksista johtopäätökseen tai sen johtopäätöksen vastakohdasta sen oletusten vastakohtaan.
- Tietämyksen käyttö on erillisen *päättelyjärjestelmän* reviiriä.

  Ratkaisu aksiooman käyttösuunnasta tehdään vasta todistushetkellä, ei vielä sitä kirjoitettaessa.
- Päättelyjärjestelmä käsittelee *merkkejä, ei merkityksiä*.

  Muutenhan kuulija voisi torjua todistuksen sopimattomana hänen omaan merkitysteoriaansa.

"Älykäs päättely" on silloin käsitteellinen ristiriita. . .

1960-luvun puolivälistä alkaen on teoriassa tietojenkäsittelyssä keskitytty laskettavuuden teorian sijasta **laskennan vaativuusteoriaan**, joka pyrkii arvioimaan laskulaitteiden ajan- ja muistintarpeen kasvua syötteen koon kasvaessa. Perusajatus on, että *polynominen* kasvu voidaan vielä sietää, *eksponentiaalista* ei: koneen kasvattamisen pitäisi aina näkyä myös sen menestyksellä suorittamien laskutehtävien koon selvänä kasvuna.

Vaativuusteorian tärkein uusi käsite on *NP-täydellisyys*. Siinä missä Gödel kysyi 1930 voidaanko *predikaattilogiikkaa ollenkaan* automatisoida, *Stephen Cook* ja *Richard Karp* kysyivät 1971–72 *kuinka tehokkaasti lauselogiikka* voitaisiin automatisoida. Lauselogiikassa nimittäin jokaiselle todelle väitteelle *on olemassa* jokin lyhyt perustely, mutta sen *löytäminen nopeasti* vaikuttaa varsin vaikealta. Laskentaongelmaa sanotaan NP-kovaksi, jos sen nopeasta ratkaisusta saataisiin jalostettua nopea ratkaisu tälle niin sanotulle SATISFIABILITY- eli SAT-ongelmalle, ja NP-täydelliseksi jos sama pätee myös toisinkin päin. (Sen *todistamista*, että nämä ongelmat välttämättä vaativat eksponentiaalisen ajan, pidetään teoreettisen tietojenkäsittelytieteen ja joskus jopa koko soveltavan matematiikan tärkeimpänä avoimena ongelmana.)

Neljäs matematiikan lahja tekoälylle on **todennäköisyyslaskenta**. Luvun 1.1.4 määritelmähän maksimoi *oletusarvoa sille hyödylle*, jonka toimija A_{ert} voi teollaan T_{nyt} saavuttaa, kun U sen "arvomaailmaksi". Tämä on puolestaan *päätösteoriaa (decision theory)* [RN95, luvut 16.1 ja 16.3], jonka *John von Neumann (1903–57)* ja *Oskar Morgenstern (1902–77)* perustivat 1944 ratkaisemaan, miten tehdä päätöksiä sattumanvaraisessa maailmassa.

Tarkastellaan esimerkiksi arpajaisia, jossa on jaossa yhteensä n arpaa á c \$, joista yksi voittaa y \$ ja muut jäävät ilman. Ostaisiko luvun 1.1.4 A_{ert} arvan, kun hän alussa omistaa x \$?

Toimijan A_{ert} *omaisuuden* odotusarvo olisi päätöksestä riippuen joko

$$\begin{aligned} E_{\text{ostaa}} &= \frac{1}{n}(x - c + y) + \frac{n-1}{n}(x - c) \\ &= x + \left(\frac{y}{n} - c\right) \ \$ \text{ tai} \\ E_{\text{ei osta}} &= x \ \$. \end{aligned}$$

Arpajaisten järjestäjä on varmaankin valinnut $y < cn$, muutenhan niistä jäätäisiin tappiolle – joskin arvat kyllä kävisivät kaupaksi. Siis

$$E_{\text{ostaa}} < E_{\text{ei osta}}$$

joten A_{ert} ei milloinkaan sortuisi ostamaan arpaa.

Kuitenkin arvat käyvät kaupaksi; ovatko niiden ostajat siis (aina) *epä*rationaalisia toimijoita?

Eivät välttämättä: A_{ert} saattaa (perustellusti) olla sitä mieltä, että vaikka c \$ tappio *hiukan* harmittaisikin, niin voittaa y \$ tuntuisi silti *paljon* paremmalta kuin turvallinen nykytilanne x \$. Eli

$$U(x-c) \underset{\text{hiukan}}{<} U(x) \underset{\text{paljon}}{<} U(x-c+y).$$

Toimijan A_{ert} *onnellisuuden* odotusarvo onkin nyt

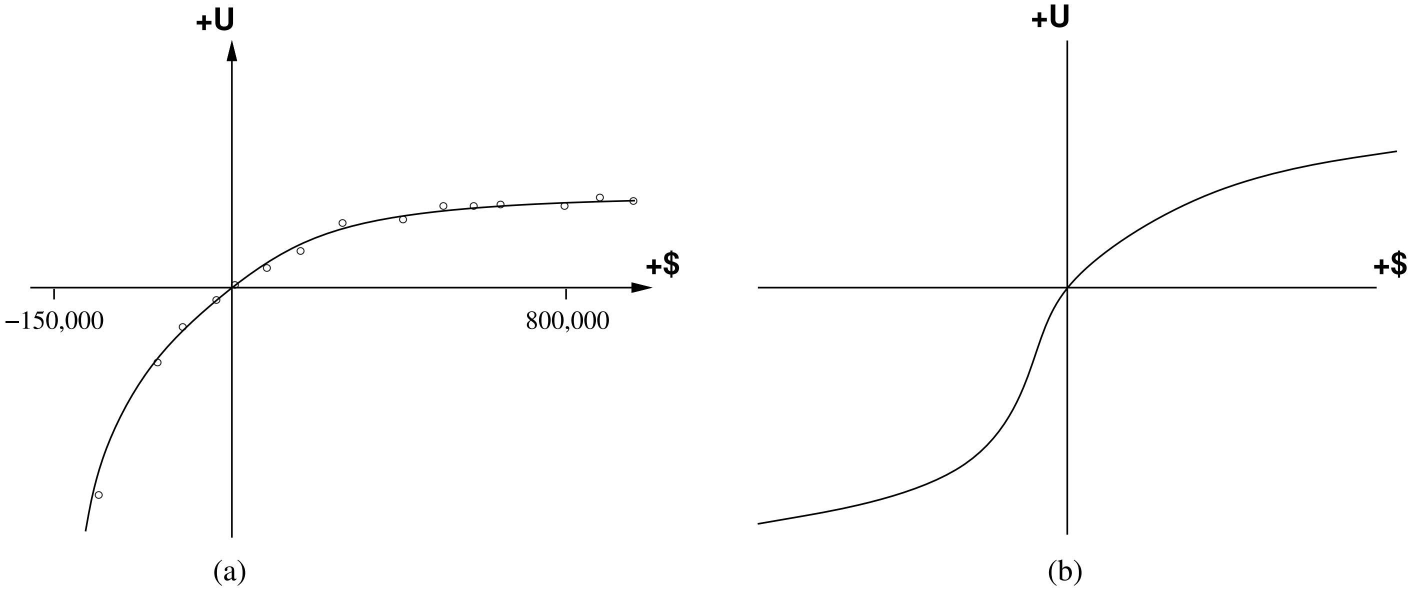
$$\begin{aligned} EU_{\text{ostaa}} &= \frac{1}{n}U(x-c+y) + \frac{n-1}{n}U(x-c) \text{ ja} \\ EU_{\text{ei osta}} &= U(x), \text{ jolloin saattaakin olla} \\ EU_{\text{ostaa}} &> EU_{\text{ei osta}}, \end{aligned}$$

eli A_{ert} saattaisikin ostaa arvan. Tai jos A_{ert} pitää uhkapelistä, niin U sisältää termin ei vain rahalle vaan myös itse osallistumiselle.

Kuvassa 1 (a) on haastattelulla selvitetty erään herra Beardin rahaa koskeva hyötymitta

$$U(x+z) = -263,31 + 22,09\log(z + 150.000\ \$),$$

missä $-150.000\ \$ < z < +800.000\ \$$ on mahdollinen tappio tai voitto. (Onko tässä luvun 1.1.2 tilanne: "älykäs arpajaiskäyttäytyminen" tarkoittaa "toimia kuten keskiverto ihiminen"?)



Kuva 1: Herra Beardin hyötykäsite [RN95, kuva 16.1.].

Herra Beard *välttää riskejä*, koska käyrä on *ylöspäin kupera* [Myr74a, luku 5.9]. Useimmat ihmiset lienevät samanlaisia.

Tällaisen käyrän omistava A_{ert} ei arpaamme ostaisi:

$$\frac{1}{n}U(x-c+y)+\frac{n-1}{n}U(x-c)<U(x) \tag{4}$$

järjestyy uudelleen muotoon

$$U(x-c+y)-U(x)<(n-1)(U(x)-U(x-c)),$$

ja *differentiaalilaskennan väliarvolauseen* [Myr74a, lause 5.4.1] mukaan löytyy $x-c<x_-<x<x_+<x-c+y$, joilla

$$\begin{aligned}(y-c)U'(x_+)&<(n-1)cU'(x_-),\ \text{eli}\\ c(U'(x_-)-U'(x_+))&<cnU'(x_-)-yU'(x_+).\end{aligned} \tag{5}$$

Koska $y<cn$, seuraa epäyhtälö (5) epäyhtälöstä

$$c(U'(x_-)-U'(x_+))<y(U'(x_-)-U'(x_+)),$$

joka puolestaan seuraa epäyhtälöstä

$$c < y \tag{6}$$

funktion U kuperuuden vuoksi.

Epäyhtälö (6) on puolestaan järkevä: miksi A_{ert} ehdoin tahdoin tuhlaisi rahaa ostamalla arvan, joka on kalliimpi kuin sillä mahdollisesti saatava voitto? Tai matemaattisesti, jos U on kasvava, niin epäyhtälö (4) pätee myös silloin kun epäyhtälö (6) ei päde.

Vastaavasti voidaan laskea, että tällainen A_{ert} uskoo sananlaskun "parempi pyy pivossa kuin kymmenen oksalla": jos A_{ert} saa valita, ottaako varmat v \$ tai osallistuuko arvontaan, jossa oletusarvoisesti voittaa *enemmän kuin* v \$, niin A_{ert} valitsee varmat v \$.

Samoin tällainen A_{ert} ottaisi *vakuutuksen*, vaikka se maksaakin enemmän kuin pelätyn vahingon oletusarvo. Tähän perustuu vakuutustoiminta: ihminen liittää kotinsa palamiseen enemmän mielipahaa kuin sen rahallinen arvo ja palamistodennäköisyys edellyttäisivät, ja on valmis maksamaan turvallisuudentunteestaan ylimääräistä. Tämä erotus on vakuutusyhtiön voittomarginaali.

Kuvassa 1 (b) on käyrää laajennettu negatiiviselle puolelle olettamalla, että jossakin vaiheessa tulee *käännepiste*, jonka jälkeen lisätappiot eivät enää paljoa lisää toimijan A_{ert} surua. Eli jos A_{ert} havaitsee, että häntä joka tapauksessa uhkaa $1,000,000$ \$ tappio ellei hän tee mitään, hän saattaa päätyä uhkarohkeaan yritykseen, joka onnistuessaan estää tappion mutta epäonnistuessaan tuplaa sen. Eikä tämä ole tyhmää tai lyhytnäköistä käyttäytymistä: jos olet uppoavan Titanicin kannella, niin *todennäköisesti* tulet kuolemaan, mutta silti sinun kannattaa yrittää *mitä tahansa* joka voisi pelastaa henkesi.

Nämä laskelmat kuitenkin edellyttävät, että A_{ert} pystyy päättelemään, mitkä teot T_{nyt} ovat tällä hetkellä mahdollisia ja miten maailma muuttuu niiden seurauksena. Toisin kuin loogisissa ideaalimaailmoissa, todellisessa maailmassa tämä tietämys ei enää olekaan täsmällistä vaan ainoastaan kerätyista havainnoista H_{nyt} nähtäviä säännönmukaisuuksia. Näin päädytään *Bayesilaiseen tilastolliseen päättelyyn*, jonka periaatteet julkaistiin sen kehittäjän *Thomas Bayesin (1702-61)* kuoltua 1763. Tämä teoria selittää, miten A_{ert} voi päivittää omien maailmaa koskevien *uskomustensa voimakkuusasteita (degree of belief)* uuden tiedon karttuessa.

Esimerkki tällaisesta päättelytavasta on luku 1.1.2: hypoteesit (4a) ja (4b) vahvensivat toinen toistaan, vaikkei kumpaakaan todistettu aukottomasti loogisessa tai matemaattisessa mielessä.

Keskeinen tutkimusongelma onkin:

Miten voitaisiin parhaiten yhdistää logiikka ja Bayesin teoria?

Ratkaisua odotellessa tiivistetään toimijalähtöinen lähestymistapamme tekoälyyn seuraavasti:

- Toimijan A_{ert} perimmäiset tavoitteet on annettu ulkoa suorituskykymittana U, jonka oletusarvoa sen tulisi maksimoida teoillaan T_{nyt} päätösteorian mielessä.

  A_{ert} saattaa "tietää" oman mittansa U ja/tai rakentaa itselleen välitavoitteita johon pyrkiä.

  Mitta U vastaa, kun A_{ert} kysyy itseltään "Mitä minä oikeastaan *tahtoisin* saavuttaa?"
- A_{ert} muodostaa "maailmankuvansa" saamiensa havaintojen H_{nyt} pohjalta. Se kertoo, mitä eri vaihtoehtoiset teot T_{nyt} ovat aiemmin saaneet aikaan, ja millä todennäköisyyksillä. Tämä informaatio tarvitaan mitan U oletusarvojen laskemiseen ja vertailuun.

  Tämä kuva koostuu tilastollisesti havaituista säännönmukaisuuksista, joiden "uskottavuutta" päivitetään Bayesilaisittain havaintoaineiston karttuessa.

  Maailmankuvalla saattaa olla eksplisiittinen esitys, mutta ei välttämättä.

  Maailmankuva vastaa kysymykseen "Mitä minä *voisin* nyt saavuttaa?"

- A_{ert} joutuu päättelemään nykyisen maailmankuvansa perusteella (esimerkiksi) mitkä vaihtoehdot T_{nyt} ovat tällä hetkellä mahdollisia, mitkä aiemmin asetetut välitavoitteet ovat vielä kesken. . . Tähän yhden tavan antaa formaali logiikka, jossa kuitenkin (yleensä) unohdetaan todennäköisyydet ja uskomusten asteet.

  Tämä komponentti tarjoaa *muistin*, jonka sisällön käsittelytapoja ei välttämättä tarvitse kiinnittää etukäteen suunnittelupöydällä.

  Tämä kurssi keskittyy tämän komponentin menetelmiin:

  – symboliseen ja loogiseen tietämyksen esittämiseen
  – etsintään ja päättelyyn tästä tietämyksestä.

  Muille kursseille jäävät esimerkiksi

  – oppiminen havainnoista (koneoppinen)
  – tilastollinen päättely (kolme käsitettä -kurssit)
  – toiminta päättelyn pohjalta (robotiikka).

1.3. Tekoäly ja filosofia [RN95, luku 1.2]

Sana "teko*äly*" viittaa filosofiassa pohdittuihin ongelmiin. Tehdään siis lyhyt katsaus niihin ajatusrakennelmiin, jotka ovat edesauttaneet alan syntyä.

Ajattelua ja päättelyä on tutkittu jo antiikissa, esimerkiksi Aristoteleen syllogismeja (luku 1.1.3). Tällöin syntyi (?) ajatus siitä, että sääntöjoukko voi ohjata ihmisen mieltä – ainakin osittain; Aristoteleen ajattelussa oli mukana myös intuitiivinen äly ja tekijän sanaton tieto.

Mikä siis on oman mielemme ja meitä ympäröivän fysikaalisen maailman välinen suhde? Päädytään *ontologiaan*, filosofian alueeseen joka kysyy "mitä *pohjimmiltaan* on olemassa?"

- Jos sekä mieli että maailma ovat kumpikin olemassa erillään, päädytään *dualismiin*, jonka tunnetuin edustaja lienee ollut *René Descartes (1596–1650)*, latinaksi *Cartesius*.

  Mutta miten mieli voi silloin vaikuttaa fyysiseen maailmaan – ohjata sitä syy-seuraus -suhteilla?
- Jos mielikin nähdään fyysisen maailman tuloksena, päädytään *materialismiin*, jonka varhainen muotoilija oli Leibniz (luku 1.2).

  Mutta miten silloin voidaan selittää se, että ihmisellä näyttää olevan vapaa tahto, jolla hän voi valita mitä tehdä, eikä vain mennä luonnonlakien mukaan kuin putoava kivi?

  Usein kuulee väitettävän, että nykyinen (luonnon)tieteellinen maailmankuva olisi materialistinen. Näin pitkälle ei voine yleistää; lähinnä uskotaan, että *ainakin* fysikaalinen maailma on varmasti olemassa. Se, voidaanko siitä saada (varmaa) *tietoa*, onkin jo vaikeampi kysymys. . .

  Tekoälytutkimuksen yleisin lähtökohta lieneekin suunnilleen, että *mieli on informaation käsittelyä, aivot taas eräs fysikaalinen laite jolla tätä käsittelyä voidaan suorittaa.*

- Näiden kahden ääripään väliin sijoittuu monen tekoälyä arvostelevan filosofin omaksuva kanta, jonka mukaan vaikka mielellä onkin fysikaalinen perusta ("Brains Cause Minds", sanoo J. Searle), *ei se silti ole selitettävissä* pelkästään palauttamalla sen operaatiot fysikkaan.

  Voi esimerkiksi hyväksyä, että yksittäiset hermosolut toimivat fysiikan lakien mukaan, mutta väittää silti, ettei niiden *yhteistoiminnan tulosta* voi enää ymmärtää fysiikan lakien kautta. (Fysiikassakin ajatellaan yksittäisten hiukkasten noudattavan deterministisiä laskulakeja, mutta suurina määrinä siirrytäänkin puhumaan tilastollisesti entropiasta.)

- Filosofiassa on esitetty myös *idealistisia* kantoja, kuten *George Berkeleyn (1685–1753)* kanta, että on vain ääretön henki (= Jumala) ja äärellisiä henkiä (= ainakin ihmismielet, entä muut?), ja "fysikaalinen maailma" onkin pelkkiä ideoita, joita näillä hengillä on [Saa85, sivut 195–208].

  Helposti ajatellaan, että nämä kannat kuuluvat "selvästi arkikokemuksen vastaisina" jo menneisyyteen. Mutta itse asiassa niiden vaikutus näkyy edelleen: miten mieli saa kaipaamaansa informaatiota ulkomaailmasta, ellei tällaisina "ideoina" – ja koskeeko saatu informaatio sitten ulkomaailmaa itseään vai mielemme ideoita siitä?

Näin päädytään *tieto-oppiin (epistemologiaan)*, joka kysyy "Miten saamme informaatiota/tietoa? Onko (periaatteessakaan) mahdollista saada varmaa tietoa? Mikä on *synnynnäisten ideoiden* osuus ymmärryksessämme, vai onko mieli syntyessään kuin tyhjä taulu (Tabula Rasa) johon kokemus kirjoittaa paitsi yksittäiset havainnot myös niiden säännönmukaisuudet?"

Miten esimerkiksi saisimme tietää että "kaikki korpit ovat mustia"? Kärjistäen

Empiristi: "Kun näemme korpin toisensa jälkeen, ja ne kaikki ovat mustia, tämä sääntö alkaa tuntua yhä vahvemmin ja vahvemmin todelta."

(Tämä argumentti on sukua Bayesin teorialle luvusta 1.2. Varhaisia empiristejä olivat Berkeleyn lisäksi *Francis Bacon (1561–1626)*, *John*

Locke (1632–1704) – Tabula Rasan ehdottaja – ja *David Hume (1711–76)*.)

Rationalisti: "Mutta näin et saa *varmaa* tietoa! Voihan toisella puolen maapalloa olla valkea korppi, jota et koskaan tule nähneeksi. Tai voi olla *periaattessa* mahdollista, että sellainen syntyisi.

Ja sitäpaitsi sinulla pitää olla jo jokin *korppiuden käsite* valmiina, jotta tietäisit mitä katsoa.

Parempi on todistaa tämä sääntö loogisesti tuosta käsitteestä."

Empiristi: "Ja mitenhän sinä tuon käsitteesi mieleesi sait? Et ainakaan oppimalla, siinähän joutuisi tarkkailemaan lintuja! Vai oletko muka syntynyt kaikki mahdolliset käsitteet valmiina päässäsi?"

⋮

Immanuel Kant (1724–1804) alkoi sovitella näitä äärikantoja yhteen esittämällä, että ihmisellä on synnynnäisiä *kategorioita* – kuten syy-seuraus -suhteen käsite – joiden avulla havaintoja maailmasta voidaan alkaa jäsentämään [Saa85, sivut 227–54]. Nykykielellä [Pin97] sama ajatus kuuluisi "evoluutio on ohjannut ihmisaivoille sellaisen rakenteen, että ne käsittelevät havaintojen informaatiota sellaisilla tietyillä tavoilla, jotka synnyttävät mielen".

Tekoälyn toimijamääritelmässämme (luku 1.1.4) tämä ajatus näkyy siinä, että me suunnittelemme toimijamme perusrakenteen – eli ohjelmoimme sen, eli asetamme sen kategoriat – ennen kuin päästämme sen itseohjautuvasti maailmaan oppimaan havainnoistaan. Tekoäly ei siis ole *puhtaasti* empirismiä. . .

. . . mutta lähes: tämän vuosisadan alkupuolen metamatematiikan (luku 1.2) vastine filosofiassa oli *looginen positivismi*, joka myöhemmin lieveni *loogiseksi empirismiksi* ja loi pohjaa tämän vuosisadan anglosaksiselle analyyttiselle filosofialle. (Esimerkiksi Tarski ja Gödel tekivät sekä metamatematiikkaa että filosofiaa.) Tällä koulukunnalla oli 2 pääteesiä:

Todennettavuusteesi: Annetun väitelauseen *merkitys* (myöhemmin:

merkityksellisyyden/ymmärtämisen ehto) = sen *todentamisen menetelmä*.

Käännettävyysteesi: Jokainen merkityksellinen väitelause voidaan *kääntää* sellaiselle "puhtaalle" kielelle, jossa puhutaan ainoastaan havainnoista.

Turingin testi -määritelmä tekoälylle (luku 1.1.1) on tällainen: väitteen "kone M on älykäs" merkitykseksi annetaan koejärjestely, jossa tuomari C havainnoi syytettyjä M ja ihminen. Väite "tämä kone on älykäs" kääntyy puolestaan muotoon "tämä kone selviäisi tuosta koejärjestelystä".

Myös *Einsteinin suhteellisuusteoria* oli loogisen positivismin motivoimaa: miltä maailma näyttää, jos *havaintojen tekonopeudelle* oletetaan yläraja, ja sitä lähestytään?

Ilman käännettävyysteesiä älykkäillä toimijoillamme A_{ert} olisi *jo periaatteessa* rajoittunut ymmärrys: silloin A_{ert} voisi kohdata (tai joutua muodostamaan) käsitteen, jonka merkitystä se ei tuntisi "kategorioidensa" eikä havaintojensa pohjalta. Eikä sillä ole muuta!

Valitessamme tekoälyn määritelmäksemme älykkäät toimijat luvussa 1.1.4 teimme samalla filosofisen valinnan siitä, mikä *ohjaa* toimintaa – tässä luvussahan on vasta puhuttu siitä, mitä on olemassa, ja miten siitä saadaan tietoa. Valitsimme *utilitarismin*, jonka filosofiassa muotoili *John Stuart Mill (1806–73)* vuonna 1863.

Sen mukaan on aina syytä pyrkiä "maksimoimaan hyötyä". Tämä "hyöty" *ei* kuitenkaan välttämättä tarkoita seuraavaa:

- Sitä, että aina olisi yksikäsitteinen hyödyn mitta kuten raha luvussa 1.2.
- Sitä, että tämä periaate *selittäisi* toiminnan: kuva 1 antaa kyllä herra Beardin hyötyfunktion, mutta ei kerro *mistä syystä* se on muodostunut sellaiseksi kuin on. Sen pohjalta voi vain ennustaa hänen tulevaa käytöstään.

- Tai sitä, että A_{ert} aina maksimoisi vain *omaa* etuaan. Se voi maksimoida myös yhteistä etua – tai jopa maksimoimalla omaa etuaan maksimoida samalla yhteistä etua.

  Esimerkkinä olkoot sosiaaliset hyönteiset kuten mehiläiset. Pesässä valtaosa naaraista (= työläiset) jättävät vapaaehtoisesti lisääntymisen kuningattarelle, ja keskittyvät hoitamaan sen jälkeläisiä.

  Tämä näyttää olevan vastaesimerkki "itsekkäälle evoluutioteorialle", jonka mukaan jokainen eläin pyrkii lisääntymään, koska sen jokainen geeni pyrkii kopioitumaan.

  Mehiläiset ovat kuitenkin *haplodiploideja*:

	koiras	**naaras**
ihminen	äidin X, isän Y	äidin X, isän X
mehiläinen	kuningattaren X, kuhnurin Y	kuningattaren X

Taulukko 1: Ihmisen ja mehiläisen sukukromosomit

  Työläinen hoitaa siis tulevaa kaksossisartaan hoitaessaan kuningattaren naarasmunaa. Se taas on yhtä tehokasta kopiointia kuin oma lisääntyminen. Kuningattaren koirasmunan hoitamiseen kuluvasta vaivasta vain puolet menee työläisen oman X-kromosomin kopiolle, ja toinen puoli isäkuhnurin Y-kopiolle. Silloin työläisen kannattaa jakaa aikansa suhteessa $3 : 1$ naaras- ja koirasmunien välillä.

  Kuningattarelle puolestaan naaras- ja koirasmunat ovat yhtä arvokkaita, kun lasketaan niiden kopiointitehokkuutta lapsenlapsituotannossa, joten hänen näkökulmastaan ajankäyttö saisi olla $1 : 1$.

  Todellisessa pesässä työläisten ajankäyttö vaihteleekin tuolla välillä riippuen siitä, ovatko työläiset vai kuningatar niskan päällä.

  Näin siis usean itsekkään toimijan yhteisössä voi syntyä (ulkoapäin katsoen) epäitsekästä yhteistyötä, kuten luvussa 1.1.4 jo todettiin.

1.4. Tekoäly, psykologia ja kielitiede

1960-luvulta alkaen on vallalla ollut *kognitiivinen* psykologia, joka tarkastelee tekoälyn tavoin aivoja informaation käsittelijänä. Monet tämän koulukunnan jäsenet lähtevätkin siitä, että kognitiivisten teorioiden pitäisi olla luvun 1.1.2 tapaan ”tietokoneohjelmien kaltaisia” eli mallintaa kognitiiviset prosessit hyvinmääriteltyinä informaationkäsittely- ja muuntotoimituksina. 1943 Kenneth Craik muotoili tämän kannan 3 askeleena:

1. Ulkoa tuleva *ärsyke (stimulus)* pitää muuntaa *sisäiseen esitysmuotoon (internal representation)* . . .
2. jota toimijan *kognitiiviset prosessit* käsittelevät tuottaen tuloksina uusia sisäisiä esitysmuotoja. . .
3. jotka sitten muunnetaan takaisin toiminnaksi.

Eli älykkäiden toimijoiden A_{ert} teoriana!

Noin vuosina 1920–60 varsinkin Yhdysvalloissa oli kuitenkin vallalla *behaviorismi*, joka oli syntynyt vastalauseeksi sellaisten subjektiivisten ”kyökkipsykologian” sanojen kuten ”uskomukset”, ”tavoitteet” ja ”järkeily” aiemmalle käytölle psykologisissa tutkimuksissa.

Behavioristien oma tutkimusohjelma oli taas kovasti luvun 1.3 loogisen positivismin oloinen: sallittiin vain tarkat objektiiviset mittaukset ja havainnot koe-eläinten reaktioista niille annettuihin ärsykkeisiin.

Craikin teoria taas mahdollisti näiden sanojen objektiivisen määrittelyn sopivien kognitiivisten tilojen ja prosessien niminä – palautti ne jälleen tieteellisiksi.

Vuonna 1957 johtava behavioristi *B.F. Skinner* julkaisi kirjan, jossa hän kuvasi behavioristisen käsityksen *kielen oppimisesta*. Nuori kielitieteilijä *Noam Chomsky* kirjoitti siitä kriittisen arvostelun [Cho97], jossa hän osoitti, ettei behaviorismi pysty kuvaamaan *kielellistä luovuutta*: sitä, että lapsi(kin) pystyy ymmärtämään ja tuottamaan sellaisia oikein muodostettuja ilmauksia, joita hän ei ole aikaisemmin tavannut. Behaviorismi oli tiensä päässä – se ei yleistynyt monimutkaiseen käyttäytymiseen.

Chomskyn oma *generatiivisten kielioppien* teoria taas selitti kielellisen luovuuden kognitiivisella mallilla, joka mukaan (ihmis)mielellä olisi erityinen *kielikyky* (erinomainen johdatus tähän teoriaan on [Pin94]):

1. Se erottaa puheen muista kuulohavainnoista *kielen sääntöjen* avulla.
   Sitten se "kääntää" sen samoilla säännöillä *mentaleesin* eli "mielen kielen" ilmaisuksi, joka säilyttää puheen merkityksen (*syvärakenteen* eli *deep structure*).
2. Muu ajattelu saa syötteenään tuon mentaleesi-ilmauksen.
   Ajattelun ei silti tarvitse olle *pelkästään* kielellistä.
3. Kielikyky muuntaa käyttäen samoja sääntöjä vastakkaiseen suuntaan ajattelun tuloksen jälleen luonnolliselle kielelle, ja panee äärielimet liikkeelle tuottamaan puhetta.
   Itse äänen tuotto vaikuttaa yllättävän hankalalta askeleelta:
   - Vauvaiän jälkeen (vain) ihmisen kurkunpää muuttuu paremmin puheen tuotantoon sopivaksi – johtaen "väärään kurkkuun"!
   - Lapsi oppii viittomakielen 24–36 kk:n "kieliräjähdystä" aiemmin.

Teoria olettaa, että aivoissa on (evoluution tuloksena [NKN01]) *universaalikielioppi*, joka sisältää kaikki ne kielioppisäännöt, joita missään luonnollisessa kielessä voi esiintyä.

Tämän vahvan oletuksen voi heikentää oletukseksi, että aivoissa on säännöt "vierekkäisten" symboleiden liittämiseksi toisiinsa ja liitosten purkuun [Car97, Dea98].

Lapsi oppii kielensä havaitsemalla mitkä näistä säännöistä ovat päällä/pois päältä ympäristössä. Empiristin "Tabula rasa" -kielenoppimisteorioita häiritsee *syötteen vähyys (dearth of input)*: 2-vuotias lapsi on kuullut niin vähän puhetta, ettei hän ole ehtinyt muodostaa alusta sääntöjä joita hän alkaa menestyksellä soveltamaan.

1.5. Tekoälyn historiaa

Tekoälyn synty (1943–56)

Ensimmäisen tekoälyartikkelin kirjoittivat neurofysiologit *W. McCulloch* ja *W. Pitts* vuonna 1943 [MP43] yhdistämällä 3 aiemmin erillistä tieteenhaaraa:

- aivojen *hermosolujen* (neuronien) toiminnasta saadun tiedon;
- formaalin logiikan; ja
- laskettavuuden teorian.

He ehdottivat myös, että hermosolujen *verkot* voisivat *oppia*. *Donald Hebb* esitti 1949 sopivan säännön päivittää solujen välisten kytkentöjen voimakkuuksia.

1950-luvun alussa alalla toimi jokunen (lähinnä) yhdysvaltalainen tutkija:

- Vuonna 1950 fysikaalisen informaatioteorian luoja *Claude Shannon* kirjoitti shakkiohjelmia – vain tietokone puuttui. . .
- Vuonna 1953 Alan Turing (luvut 1.1.1 ja 1.2) teki samaa Iso-Britanniassa, kunnes hallitus kielsi häneltä tietokoneiden käytön moiseen humpuukiin.
- Vuonna 1951 *Marvin Minsky* ja *Dean Edmonds* rakensivat ensimmäisen hermoverkkokoneen Princetonin yliopiston matematiikan laitoksella.

  John von Neumann (luku 1.2) puolusti Minskyn väitöskirjaa arvostelijoilta, jotka eivät pitäneet sitä matematiikkana.

Samoin Princetonissa vaikuttanut *John McCarthy* siirtyi valmistuttuaan Darthmouth Collegeen, jossa hän, Minsky, Shannon ja *Nathaniel Rochester* (IBM) järjestivät 1956 kesäkoulun. 10 osallistujan joukossa olivat

Trenchard More (Princeton), *Arthur Samuel* (IBM) sekä *Ray Solomonoff* ja *Oliver Selfridge* (MIT).

Muita pidemmällä olivat kuitenkin *Allen Newell* ja *Herbert Simon* (Carnegie Tech, nyt Carnegie-Mellon U. eli CMU): heidän ohjelmansa *Logic Theorist* teki samanlaisia loogisia todistuksia kuin Principia Mathematica (luku 1.2) sisälsi. Hilbertin ohjelma kantoi siis sittenkin hedelmää!

Newellin ja Simonin perusajatuksena oli, että *kone etsii ratkaisua annettuun ongelmaan omaamansa tiedon ohjaamana*.

Alkuinnostus, suuret toiveet (1952–69)

Tänä aikana moni tekoälyn vastaväite tyyppiä ”X on mahdotonta tietokoneelle” kumottiin antamalla vastaava ohjelma — kunhan X oli tarpeeksi hyvin rajattu ongelmaperhe, ja esimerkit riittävän pieniä senaikaisille tietokoneille.

Newell ja Simon jatkoivat ohjelmalla *General Problem Solver (GPS)*, joka käytti samanlaisia ongelmanratkaisutapoja kuin ihmiset. Tämä kognitiotieteellinen tekoälytutkimus (luku 1.1.2) on jatkunut CMU:ssa aina näihin päiviin.

Samuel kirjoitti vuodesta 1952 alkaen *oppivia* ohjelmia tammipeliin (checkers), jotka aikanaan ohittivat laatijansa taidot. Rochesterin johdolla jatkettiin Logic Theoristin aloittamaa työtä teoreemien todistamisessa logiikan säännöillä.

McCarthy siirtyi MIT:iin ja vuonna 1958 teki historiaa kolmesti:

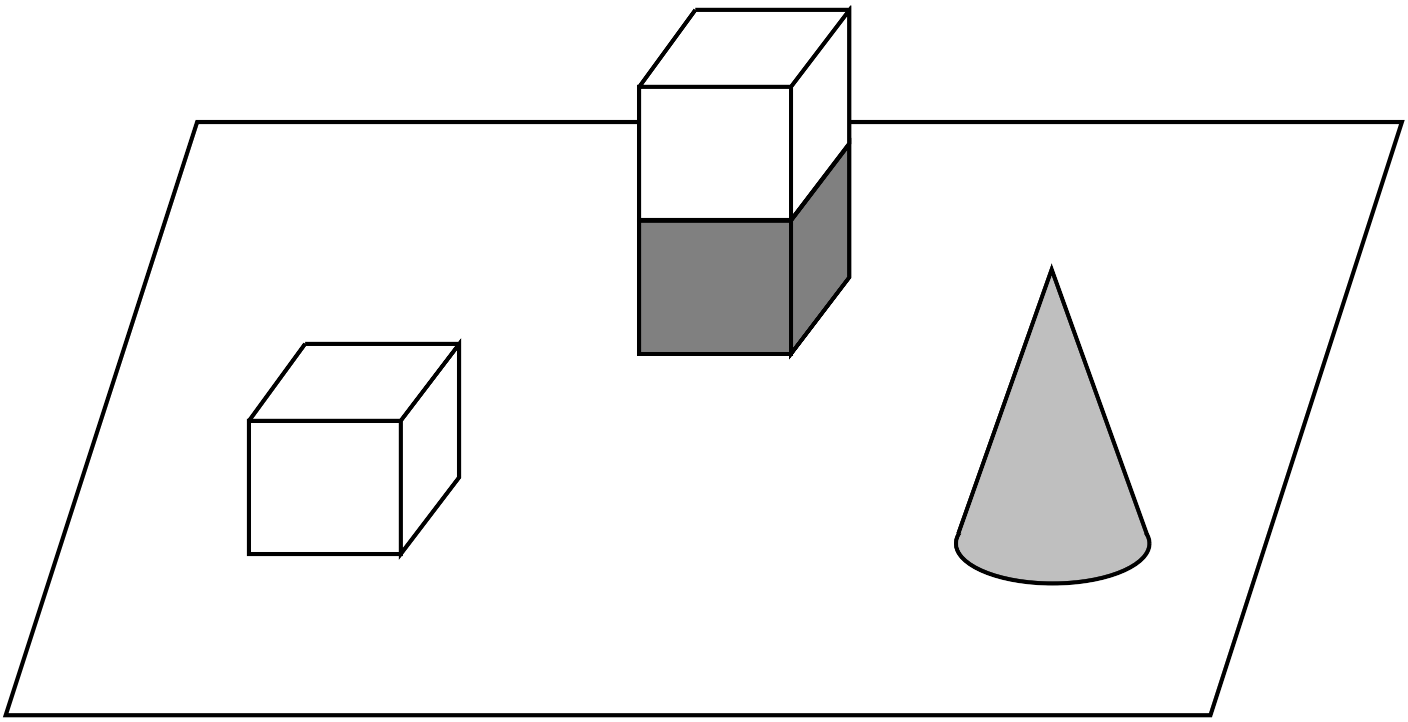
- Hän määritteli uuden *AI lab*oratorion ensimmäisessä raportissa ohjelmointikielen *Lisp* (*Lis*t *p*rocessing), joka on edelleen elossa tärkeimpänä tekoälykielenä — ja toiseksi vanhimpana vielä käytettynä kielenä heti vuonna 1957 esitellyn FORTRANin (FORmula TRANslator) jälkeen.
  Kielen perustietorakenne *lista* oli peräisin jo Logic Theoristia varten kehitetystä ohjelmointikielestä IPL.

- Yhdessä muiden MIT:n tietokoneihmisten kanssa hän ideoi *osituskäytön (time sharing)*.
- Hän esitteli — muttei toteuttanut — ehkä ensimmäisen *täydellisen* tekoälyohjelman *Advice Taker*. Perusidea oli Newellin ja Simonin, mutta nyt aksiomatisoitavana olikin matematiikan sijasta *arkimaailma*. Ohjelmaan pystyi myös lisäämään uusia aksiomeja ilman uudelleenohjelmointia, eli erotti toisistaan *tietämyksen esittämisen logiikalla* ja *näiden esitysten käsittelyn deduktiivisilla päättelysäännöillä*.

Samana vuonna myös Minsky siirtyi MIT:iin ja alkoi yhteistyö, joka alkoi rakoilla kun "scruffy" Minsky painotti toimivia ohjelmia logiikan kustannuksella ja "neat" McCarthy taas painotti tietämyksen esittämistä ja loogista päättelyä. Nämä kaksi kantaa ovat ottaneet mittaa toisistaan myöhemminkin — ja täydentäneet toisiaan.

Vuonna 1963 McCarthy perusti Stanford Research Instituten (SRI) ja jatkoi siellä Advice Taker -ideaansa.

MIT:ssa Minskyn oppilaat kehittivät puolestaan ohjelmia käsittelemään (usein simuloituja) *mikromaailmoja*:



Kuva 2: Palikkamaailma (blocks world) [RN95, kuva 1.3].

Palikkamaailmassa tutkittiin menestyksellisesti esimerkiksi *konenäköä*, *oppimista*, *luonnollisen kielen ymmärtämistä* ja *moniosaisten suunnitelmien tekoa*.

Innostus laantuu, toiveet karisevat (1966–74)

Tutkijoiden *ylioptimistiset* odotukset ja ennusteet jäivät toteutumatta, kun ratkaistavat *tehtävät kasvoivat ja monimutkaistuivat* (skaalautumisongelma):

- Monet varhaiset ohjelmat eivät olleet kovinkaan "älykkäitä" vaan suoriutuivat pienistä esimerkkitehtävistään yksinkertaisin syntaktisin manipulaatioin.

  Esimerkiksi luonnollisten kielten kääntämistä toisikseen pidettiin helppona: ohjelmoitaisiin Chomskyn kielioppiteoria ja kerättäisiin elektroninen sanakirja. Kuitenkin tekstin "ymmärtäminen" vaatii *ennakkotietämystä* sen aiheesta — maailmasta, ihmisistä,. . . — moniselitteisyyksien ratkomiseksi:

  $$\text{"Olen nähnyt Helga-neidin } \left\{ \begin{array}{c} \text{kun minä olin} \\ \text{kun hän oli} \end{array} \right\} \text{ kylvyssä, ihanaa!"}$$

- Monet etsintäongelmat tiedetään nyt NP-koviksi (luku 1.2).

  Vaikka ohjelmassa olisikin mekanismi ratkaisemaan tehtävä *periaatteessa*, ei tuo mekanismi välttämättä ratkaise sitä *käytännössä*.

  Kyvyttömyys hallita tämä *kombinatorinen räjähdys* oli (muodollisena) syynä *Lighthill-raporttiin*, joka lopetti Iso-Britannian hallituksen tuen tekoälytutkimukselta kahta yliopistoa lukuunottamatta.

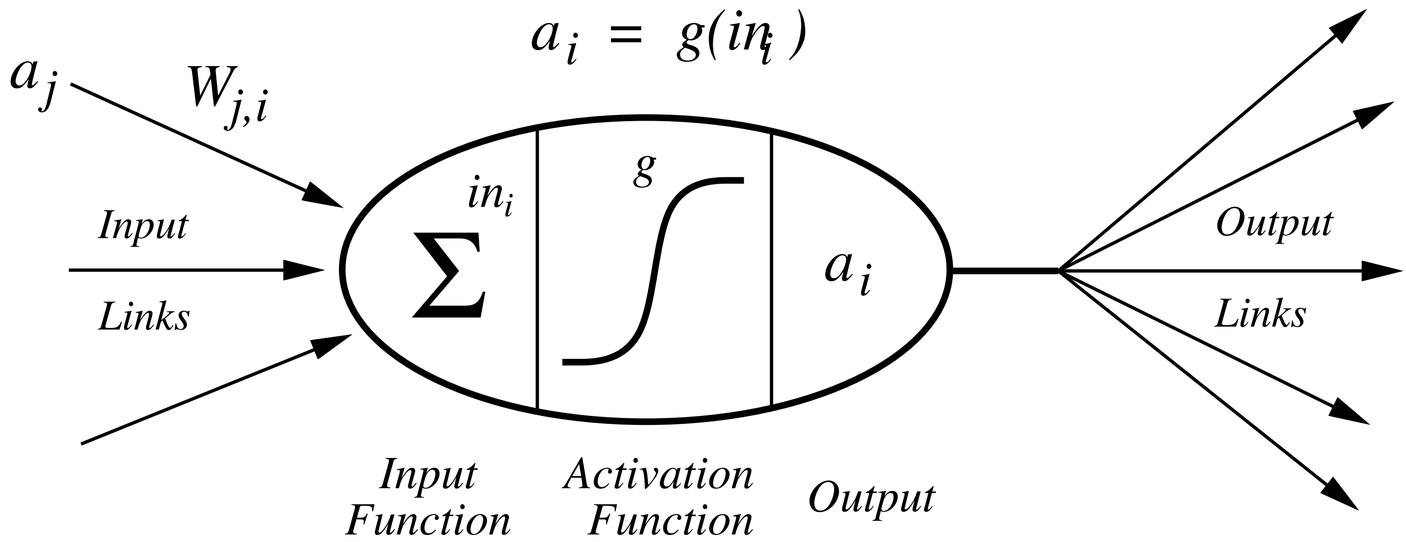
- Hermoverkkotutkimuksessa johdettiin teoreettisia tuloksia käytetyistä tekniikoista:

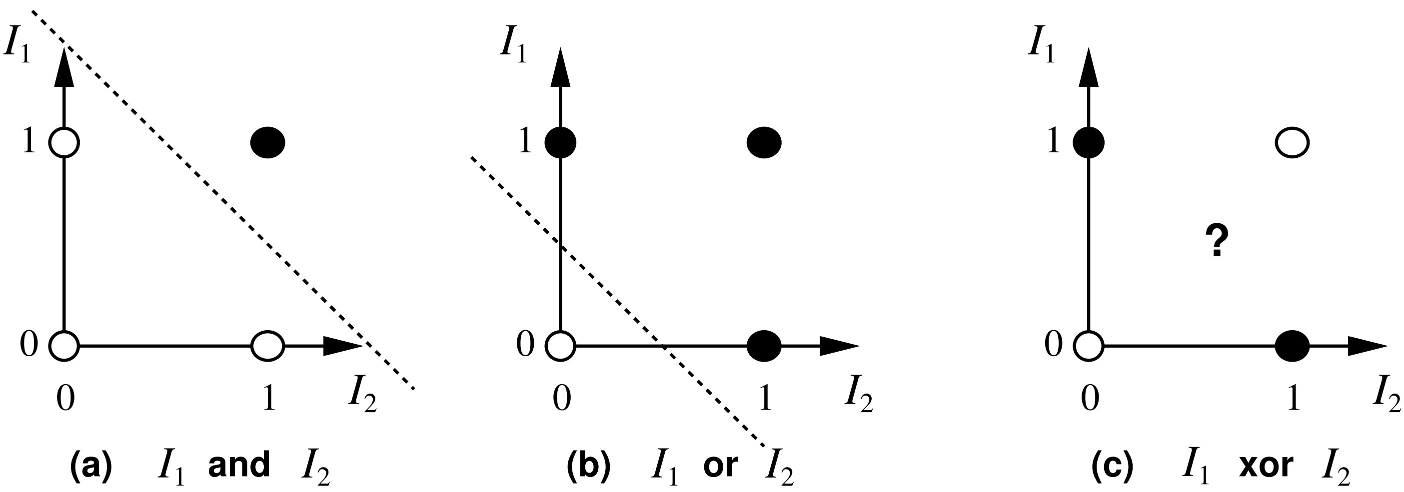
  1. *Frank Rosenblatt* osoitti vuonna 1962 *perceptronien konvergenssilauseen*, jonka mukaan hänen *oppimisalgoritminsa* pystyi sovittamaan *yhden* hermosolun kytkentäpainot vastaamaan mitä tahansa sellaista syötettä, jolle sopivat painot olivat olemassa.

     *Moni*soluisia verkkoja tutkittiin myös, ja uskottiin, että niillekin saataisiin pian samanlaisia tuloksia.

2. Vuonna 1969 (aiempi hermoverkkotutkija) Minsky ja *Seymour Papert* julkaisivat kuitenkin kirjan *Perceptrons*, jossa he osoittivat että nämä sopivat painot olivat olemassa vain hyvin yksinkertaisille syotteille.

   Vaikka monisoluiset verkot ratkaisevatkin tämän ongelman, ei niille tunnettu vielä oppimisalgoritmeja. Kirjan sisältämän terävän kritiikin (jota 2. painos 1988 lievensi) tuloksena tietojenkäsittelyllisen hermoverkkotutkimuksen rahoitus kutistui olemattomiin. . .





Kuva 3: Perceptron ja Boolen logiikka [RN95, kuvat 19.4 ja 19.9].

3. . . . peittäen alleen sen, että sellainen algoritmi oli juuri keksitty (tosin kansantaloustieteessä)! Hermoverkkojen tutkimus koki renesanssin vasta 1980-luvun loppupuolella, kun se keksittiin uudelleen.

Tietämyspohjaiset järjestelmät (1969–79)

Newell ja Simon olivat uskoneet *heikkoihin menetelmiin*, joissa itse etsintäkone oli yleiskäyttöinen ja syötteenä annetut säännötkin kuvailivat yleispäteviä periaatteita. Näin saadut tuloksetkin olivat usein heikkoja.

Alettiin tutkia vaihtoehtoja, joissa valittiin ensin sovellusalue ja sitten kirjoitettiin hyvin sovellusaluekohtaisia sääntökokoelmia. Ei siis mallinnettu enää "yleistä" älykkyyttä vaan sitä suorempaa päättelyä jota *asiantuntija (expert)* tekee käyttäessään omaa erityisosaamistaan.

DENDRAL, joka päätteli näytteen molekyylirakenteen massaspektrometrin antamista mittaustuloksista, lienee ollut ensimmäinen *tietämysintensiivinen ohjelma*, jonka asiantuntemus johtui suuresta kokoelmasta erityistapauksia koskevia sääntöjä.

MYCIN puolestaan diagnosoi verisairauksia noin 450 säännön voimin lähes yhtä hyvin kuin alan erikoislääkärit. Se erosi DENDRALista kahdessa suhteessa:

- DENDRALin sääntökanta voitiin johtaa kyseeseen tulevista kemian teorioista; MYCINin pohjaksi ei ollut tarpeeksi tarkkaa lääketieteellistä teoriaa, vaan se kerättiin *haastattelemalla* alan spesialisteja.
- MYCINin sääntöihin oli liitettävä *varmuuskertoimia (certainty factors)* heijastamaan tarjolla ollutta vain likimääräistä, ei varmaa sovellusalueen tietämystä. Niistä tuli suosittu tapa mallintaa epävarmuutta sääntöpohjaisissa asiantuntijajärjestelmissä.

  Jos annetun säännöstön ja siitä tehtävien päätelmien rakenne on tietyllä tapaa yksinkertainen (kuten onneksi MYCINissä) niin varmuuskertoimet ovat eräs Bayesin teorian (luku 1.2) erikoistapaus; mutta *yleisesti* niiden lisääminen saattoi antaa aivan harhaanjohtavia tuloksia, joten niiden käyttöa ei enää suositella [RN95, luku 15.6].

Epävarmuuden mallinnukseen (varsinkin lääketieteellisessä diagnoosissa) käytettiin jo tuolloin myös perinteistä todennäköisyyslaskentaa ja utiliteettiteoriaa. Esimerkiksi probabilistinen päättelykone PROSPECTOR ehdotti koeporauksia paikassa, josta löytyi laaja molybdeeniesiintymä.

Luonnollisen kielen käsittelyä tutkittiin joko rajoittumalla yksinkertaisiin ympäristöihin (*Terry Winograd*in SHRDLU palikkamaailmassa) tai lisäämällä tarvittavaa taustatietoa ja päättelyä sen pohjalta (*Roger Schank* oppilaineen Yalessa).

Tosielämän sovelluksia varten kehitettiin myös uusia tiedonesitystapoja:

Prolog on logiikkaan perustuva ohjelmointikieli, joka syntyi Ranskassa ja Iso-Britanniassa (Skotlannissa) luonnollisen kielen käsittelyn tarpeista.

Planner on myöskin logiikkaan perustuva yhdysvaltalainen perhe kieliä monimutkaisten suunnitteluongelmien esittämiseen ja ratkaisuun.

kehykset (frames) ovat Minskyn *rakenteinen* vaihtoehto logiikalle, jossa tietämys esitetään hierarkisina rakenteina ja niiden solmujen välisinä suhteina. *Olio-ohjelmointi* voidaan nähdä tämän vaihtoehdon tuomisena tietämyksen esittämisestä ohjelmointiin.

Tekoälyteollisuuden synty (1980–88)

Alan kaupallinen liikevaihto kasvoi vuoden 1980 muutamasta miljoonasta vuoden 1988 2 miljardiin dollariin.

Japani käynnisti 1981 kansallisen "viidennen sukupolven tietokonehankkeen": pyrittiin Prolog-tekoälykoneeseen jossa olisi esimerkiksi täysimittainen luonnollisen kielen tunnistus. Yhdysvalloissa tämä nähtiin uhkana, joka kasvatti tekoälytyön kiinnostusta ja rahoitusta. Iso-Britanniassa *Alvey-raportti* käynnisti "uuden" tutkimusalueen nimeltä "älykkäät tietämyspohjaiset järjestelmät".

Samalla tuotettiin tekoälytehtäviin sopivia laite- ja ohjelmointiympäristöjä (Lisp-koneet ja -ympäristöt). Nämä markkinat ja niillä toimineet yritykset eivät ole kuitenkaan enää hengissä: tavallisilla mikrotietokoneilla saavutettiin lähes samat edut paljon halvemmalla.

Hermoverkkojen paluu (1986–)

Hermoverkkojen tutkimus oli jatkunut tietojenkäsittelyn ulkopuolella:

Fyysikot kuten *John Hopfield* mallinsivat niitä ja niiden vuorovaikutuksia alunperin kiinteän aineen atomeille kehitetyillä menetelmillä.

Psykologit kuten *David Rumelhart* ja *Geoff Hinton* mallinsivat niillä (ihmis)muistia.

1980-luvun puolivälin tienoilla monisoluisten verkkojen oppimisalgoritmi keksittiin uudelleen — ainakin 4 kertaa. . . — ja sitä sovellettiin moniin tietojenkäsittelyn ja psykologian oppimisongelmiin.

Nykytila (1987–)

Tekoälytutkimus on vakiintumassa "normaalitieteen tilaan", jossa "neat"-tutkijat testaavat ja kehittävät edelleen jo olemassaolevia teorioita todellisiin tilanteisiin. (Uusi "scruffy-vallankumous" voi olla tarpeen kunhan teorioiden rajat tulevat vastaan.)

Esimerkiksi puheen- ja käsialantunnistus on edennyt kaupalliseen käyttöön saakka, kun otettiin käyttöön *piilo-Markov-mallit* todennäköisyyslaskennasta aikaisempien *ad hoc*-ratkaisujen sijaan. Samoin epävarman tiedon hallinnassa on edetty korvaamalla varmuuskertoimet Bayesin ja päätösteorian menetelmin.

Varsinainen tavoite – "aito" tekoäly – näyttääkin olevan (vähintään) yhtä kaukana nyt kuin alkuinnostuksen vuosina; tällä välin olemme oppineet paljon sen hankaluuksista. . .

Kurssimme "älykäs toimija" -määritelmä tekoälylle onkin "kattokäsite" yhdistämään käsitteellisesti tekoälyn eri osa-alueet, ei lähitulevaisuuden tuotekehittelyä.

1.6. Tekoälyn omasta filosofiasta [RN95, luku 26]

Tekoälyn filosofia jakaa sitä koskevat väitteet kahteen päätyyppiin:

Heikko tekoäly kysyy "voidaanko kone saada käyttäytymään *ikään kuin* se olisi älykäs?"

Tämän väitteen arvostelijat esittävät seuraavanlaisia kantoja:

- On asioita, joihin tietokoneet eivät pysty, ohjelmoitiinpa ne miten tahansa.
- Tietyt tavat suunnitella älykkäitä ohjelmia tulevat epäonnistumaan.
- Näiden ohjelmien laatiminen on käytännössä mahdotonta.

Vahva tekoäly kysyy "onko tällaisella koneella *tietoisuus (consciousness)*?"

- Turingin määritelmä (luku 1.1.1) väitti, että heikosta tekoälystä seuraa vahva.
- Näin voi uskoa uskomatta heikkoon: jos voitaisiin, niin olisi.

Turing luetteli myös useita (heikon) tekoälyn vastaisia argumentteja [Tur50, luku 6], joista osa elää tänäänkin, esimerkiksi:

Matemaattinen vastaväite lähtee Gödelin (tai Turingin omista) epätäydellisyystuloksista (luku 1.2), joiden mukaan mikään formaali systeemi — kuten tietokone — ei voi ymmärtää tiettyjä matemaattisia väittämiä jotka koskevat sitä itseään.

Gödel on ne kuitenkin todistanut ja muut matemaatikot todistuksen ymmärtäneet. Siis he ovat tehneet koneelle mahdottoman älykkyyttä osoittavan ja vaativan toimituksen.

Eräs vastaus on, että vaikka formaali systeemi — matemaatikko tai tietokone — pystyykin käsittelemään *itseään "yksinkertaisempien"* systeemien väittämiä, se ei pystykään tekemään samaa omalle väitteelleen. Tämä erottaa "kielen" *jota* tarkastellaan ja "metakielen" *jossa* tarkastellaan.

Roger Penrose esitti tämän vastaväitteen kirjassaan "Keisarin uudet aatteet" vuonna 1989 [Pen89]. Penrose. . .

- halusi toisaalta pitää kiinni aivotoimintojen ja sitä kautta myös tietoisuuden fysikaalisesta perustasta,
- mutta toisaalta halusi kieltää niiden mekaanisen laskettavuuden.

Hän päätyi arvelemaan, että aivoissa tapahtuvat kvanttimekaaniset ilmiöt ovat tärkeitä tietoisuudelle. (Liekö lääke tautia pahempi?)

Käyttäytymisen epämuodollisuus on vastaväite, jonka mukaan inhimillinen käyttäytyminen on niin monimutkaista, ettei sitä voi vangita sääntöihin.

Näin on ajatellut esimerkiksi *Hubert Dreyfus* [DD88].

Kannan edustajilla on kuitenkin jälleen kaksijakoinen ongelma:

- Antaa vastaesimerkki älykkyyttä vaativasta toiminnasta, jota ei voi kuvailla säännöin.
- Antaa vaihtoehtoinen selitys omalle esimerkilleen ilman että samalla tulee itse kehittäneeksi sille säännöt.

Tunnetuin vastaväite vahvalle tekoälylle lienee *John Searlen "kiinalainen huone" (Chinese Room) -ajatuskoe* [Sea80, Sea90, CSC90][Bod90, §4]:

- Istun kopissa, jossa on vain
  - yksi luukku nimeltä 'sisään'
  - toinen luukku nimeltä 'ulos'
  - ja valtava kirja jonka kullakin sivulla on kiinalainen kirjoitusmerkki

  enkä minä osaa kiinaa
- mutta osaan tehdä seuraavaa:
  - kun 'sisään'-luukusta näytetään minulle jokin merkki $\mathfrak{A}$
  - niin etsin kirjasta sitä vastaavan aukeman $\boxed{\mathfrak{A}}\boxed{\mathfrak{B}}$
  - ja näytän 'ulos'-luukusta vastaavan merkin $\mathfrak{B}$.
- Läpäisköön tämä koppi Turingin testin keskustelemalla kiinaksi täysin ihmisen tavoin.

  Kuka tai mikä kopissa
  - osaa kiinaa?
  - on tietoinen (keskustelluista asioista)?

Searlen mukaan ei kukaan eikä mikään!

- Searlen johtopäätökset:
  - Ohjelmaa suorittava tietokone ei ole tietoinen — huonehan on eräänlainen Turingin kone.
  - Tietoisuutta ei voi koodata ohjelmaksi — kirjahan on eräänlainen ohjelma.

Searle haluaa tämän vuoksi eroon tietoisuuden Turing-määritelmästä pelkkänä syöte/tulos-vastaavuutena.

Toisaalta Searle (kuten Penrose) haluaa pitää kiinni tietoisuuden fysikaalisesta perustasta aivotoiminnassa: "brains cause minds".

Searlen haluja voi toisaalta pitää ristiriitaisina *aivoproteesikokeen (the Brain Prosthesis Experiment)* nojalla:

- Oletetaan lääketieteen kehittyneen niin pitkälle, että yksittäinen neuroni voidaan korvata täsmälleen saman syöte/tulos -käyttäytymisen omaavalla mikropiirillä ilman että aivojen toiminta häiriintyy leikkauksessa.
- Vaihdetaan saman potilaan kaikki neuronit *yksi kerrallaan*. Miltä potilaasta itsestään tuntuu, häviääkö hänen tietoisuutensa? Searlen mukaan häviää; tutkitaan mitä tästä seuraa.
- Koska yksittäisten aivonosien syöte/tulos -käyttäytyminen ei muutu, niin koko potilaan havaittava käyttäytyminen ei muutu yllättäen.

  Tällöin potilaan *oma tahto katoaa yhtäkkiä*, muuten hän ehtisi huudahtaa "apua!"

  Siis on jokin "kriittinen massa" neuroneita, jotka juuri ja juuri riittävät oman tahdon olemassaoloon.
- Kun kaikki vaihdot on suoritettu, on saatu alkuperäisten aivojen kanssa syöte/tulos -käyttäytymiseltään identtiset "sähköaivot", joilta Searle kuitenkin kieltää tietoisuuden.

  Silloin potilaan alkuperäistenkään aivojen tietoisilla tiloilla ei ollut mitään vaikutusta hänen käyttäytymiseensä, koska nämä tilat ovat nyt hävinneet mutta käyttäytyminen säilyi.

- Peruutetaan lopuksi tämä leikkaus jälleen neuroni kerrallaan, ja kysytään sen jälkeen "miltä leikkaus tuntui?"

  Koska potilaan neuronien syöte/tulos -käyttäytymistä ei taaskaan häiritty, hänen pitäisi muistaa tuntemukset myös siltä ajalta jolloin aivoissa ei ollut yhtään neuronia.

  Searlen mukaan sinä aikana ei aivoissa kuitenkaan ollut ketään kokemassa noita tuntemuksia.

Toisaalta todellinen neurotutkimus ei ehkä salli näin teräviä tulkintoja:

- Tietoisuuden "kriittinen neuronimassa" ei ehkä olekaan kiinteä suure: tietoisuus saattaa tarvita kullakin hetkellä käytössään olevien aktiivisten neuronien lisäksi *vapaita lisäneuroneita* jotka voivat aktivoitua tulevaisuudessa samalla kun osa nykyisistä passivoituu. [TE88]

  Tietoisuuden kulloinenkin sijainti voi siis "matkailla" aivoissa ajan kuluessa, ja tarvita siihen vapaata kulkutilaa.

- Aivot ovat myös yksi *elin* joka on yhteydessä muuhun ruumiiseen myös muuten kuin lähettämällä ja vastaanottamalla hermoimpulsseja.

  Tietoisuus saattaa olla jopa *evoluution sivutuote* matkalla kohden yhä tarkempaa eliön itsesäätelymekanismia [Dam99].

  Tietoisuutta ei silloin ehkä voi tutkia erottamalla aivoja muusta ruumiista tai edes ei-tietoisista aivoprosesseista.

2. Suunnitelmia älykkäiksi toimijoiksi

— Mikä tuo on? minä huudahdin haltioissani.

— Karuselli, Fredrikson sanoi. Etkö muista. Minähän tein sinulle konepiirustuksen. Läpileikkauksen ja kaikki.

— Mutta ei se tuolta näyttänyt, minä intin. Tässä on hevosia, musiikkia ja viirejä ja punaista kultaa!

— Ja rataspyöriä, Fredrikson sanoi. *Tove Jansson: Muumipapan urotyöt*

Meille on annettu tehtäväksi suunnitella automaattinen taksikuski (= sen ohjausohjelma). Sopimusehdot on kirjattu seuraavaksi *PAGE-kuvaukseksi:*

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Kuva 4: Taksikuskin PAGE-kuvaus [RN95, kuva 2.6].

Saadaan 5 erilaista toimijaluokkaa riippuen siitä, miten Algoritmin 1 (kalvo 8) eri osat — *muisti* ja *parhaan teon valinta* — päätetään toteuttaa.

2.1. Taulukko-ohjatut toimijat

Voitaisiin yrittää *taulukoida etukäteen* paras teko kullekin havaintojonolle:

```
function TABLE-DRIVEN-AGENT(percept) returns action
   static: percepts, a sequence, initially empty
           table, a table, indexed by percept sequences, initially fully specified

   append percept to the end of percepts
   action ← LOOKUP(percepts, table)
   return action
```

Kuva 5: Taulukko-ohjattu toimija [RN95, kuva 2.5].

Teoriassa tällainen optimaalinen (ääretön) taulukko LOOKUP on kuhunkin tehtävään kyllä olemassa, mutta. . .

- Esimerkiksi shakkipelin (jossa havaintomaailma on yksinkertainen ja jonot äärellisiä) taulukossa on noin 35^{100} alkiota.
  Minne se mahtuisi?
- Suunnittelijan tulisi kyetä myös *löytämään* taulukko.
- Tulos ei ole autonominen, koska kaikki on laskettu valmiiksi. Jos maailma muuttuu yllättäen (shakkipelin sääntöjä muutetaan) niin toimija olisi hukassa, tarvittaisiin uusi taulukko.
- Vaikka toimijaan lisättäisiinkin *oppimismekanismi*, niin useimmiten kestäisi aivan liian kauan ennen kuin toimija olisi oppinut taulukon (useimmin käytytyt osat). Mutta toimija olisi nyt autonominen!

Taulukko-ohjattu toimija siis toteuttaa älykkään toimijan määritelmän, mutta se voidaan usein toteuttaa myös muilla, käytännöllisemmillä tavoilla.

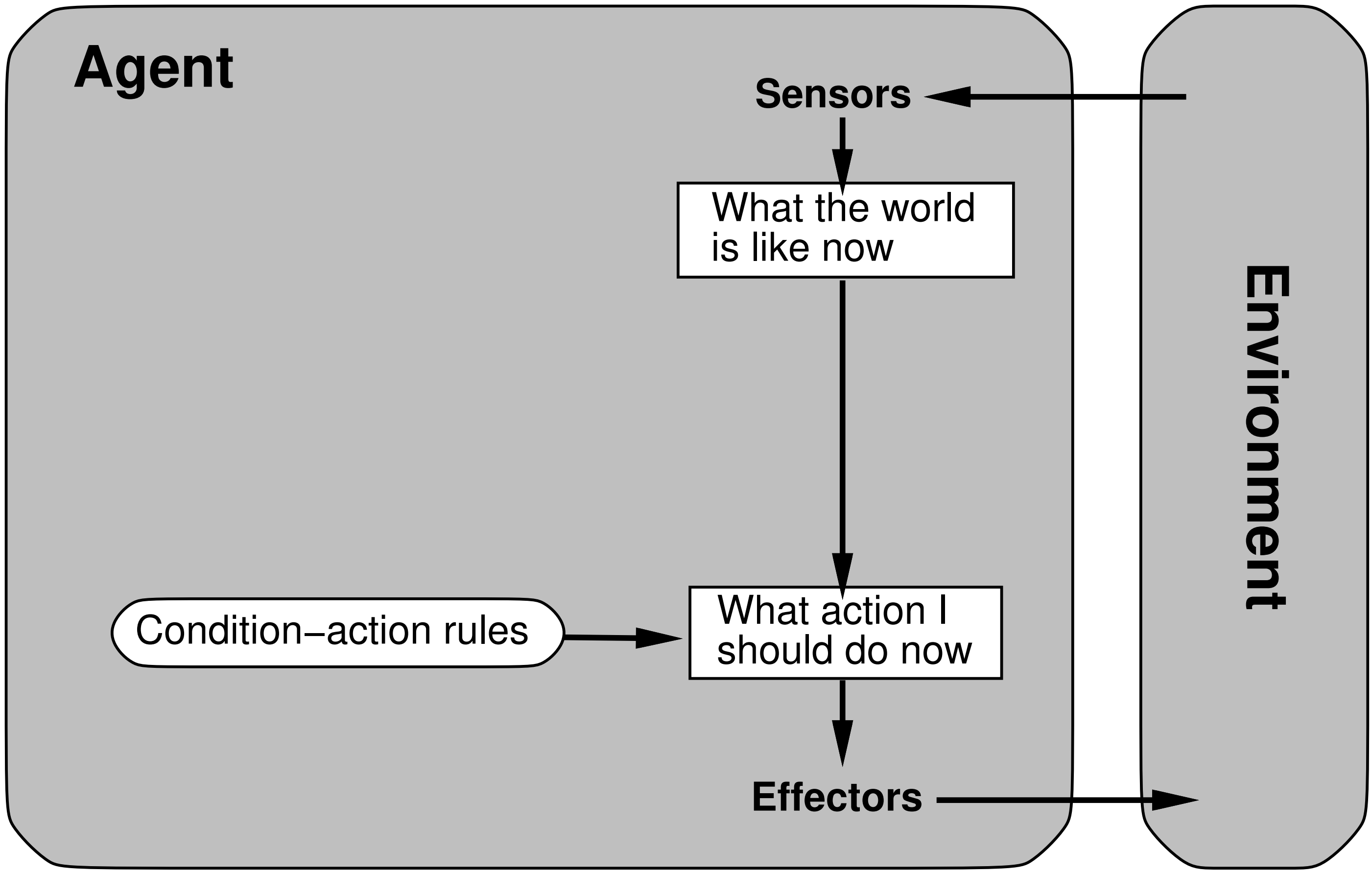
Luvun 1.6 aivoproteesikoe voitaisiin tehdä myös taulukolla mikropiirin sijaan — jolloin taulukko olisi tietoinen!

2.2. Muistittomat heijastetoimijat

Luvun 2.1 optimaalista taulukkoa LOOKUP voidaan kutistaa ottamalla käyttöön "vaistonvaraisia" *jos-niin-sääntöjä*

if edellä ajava auto jarruttaa **then**
  aloita jarrutus
end if

esittämään havaintojonojen odotettavissa olevia säännönmukaisuuksia. Tulos on "behavioristinen" toimija sääntöinään ärsyke-refleksi-parit.



function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
  static: *rules*, a set of condition-action rules

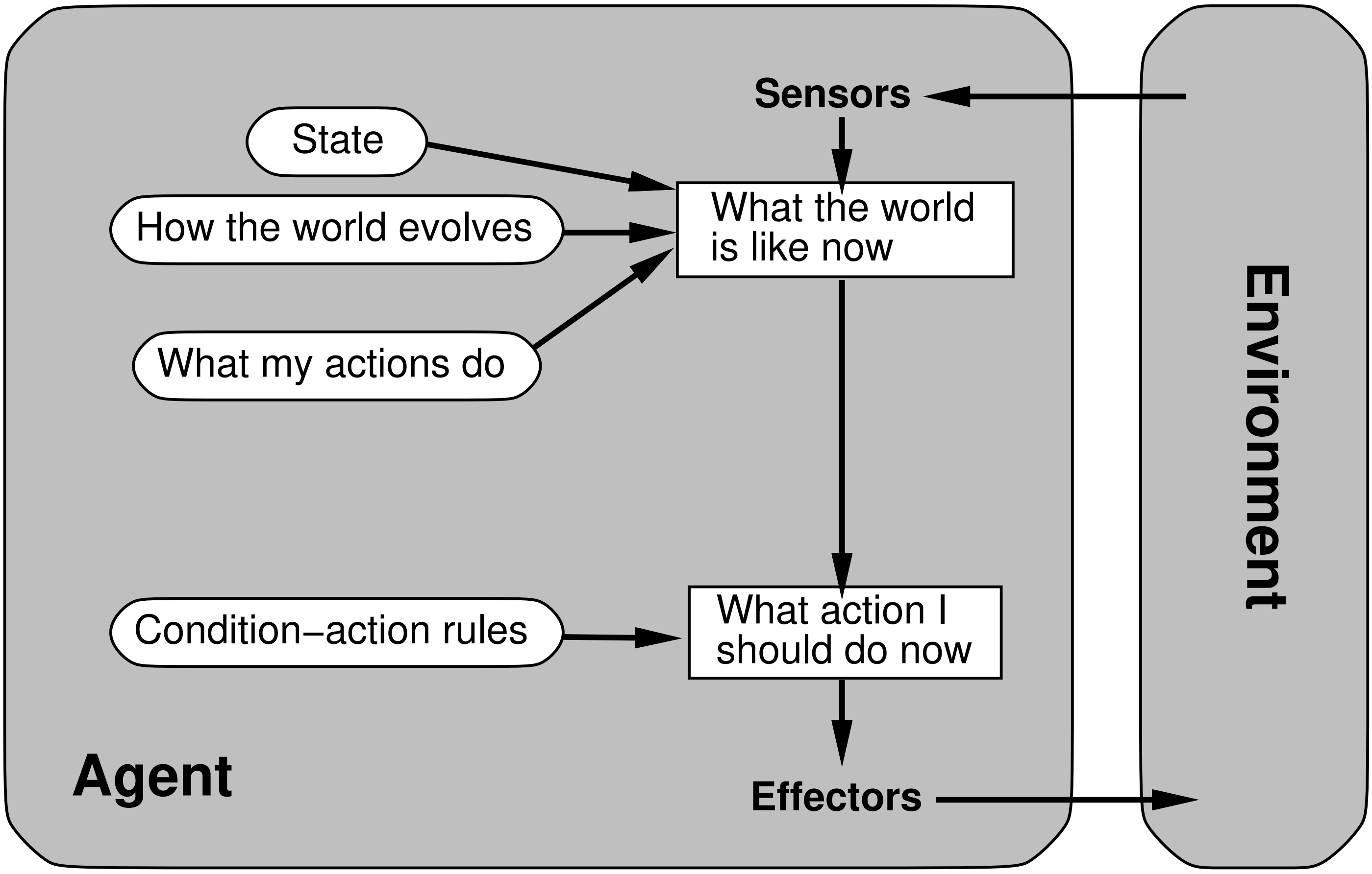
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state, rules*)
  action ← RULE-ACTION[*rule*]
  return *action*

Kuva 6: Muistiton heijastetoimija [RN95, kuvat 2.7 ja 2.8].

2.3. Muistavat heijastetoimijat

Luvun 2.2 toimijan sääntöjen valinnassa voi tarvita tietoa ympäristön *muutoksista*: miten erottaa kääntyvä auto jarruttavasta jonka toinen takavalo on palanut?

Ympäristö muuttuu sekä toimijan omien tekojen seurauksena että siitä riippumatta. UPDATE-STATE päivittää toimijan nykyistä "maailmankuvaa", ja sen pohjalta valitaan käytettävä sääntö.



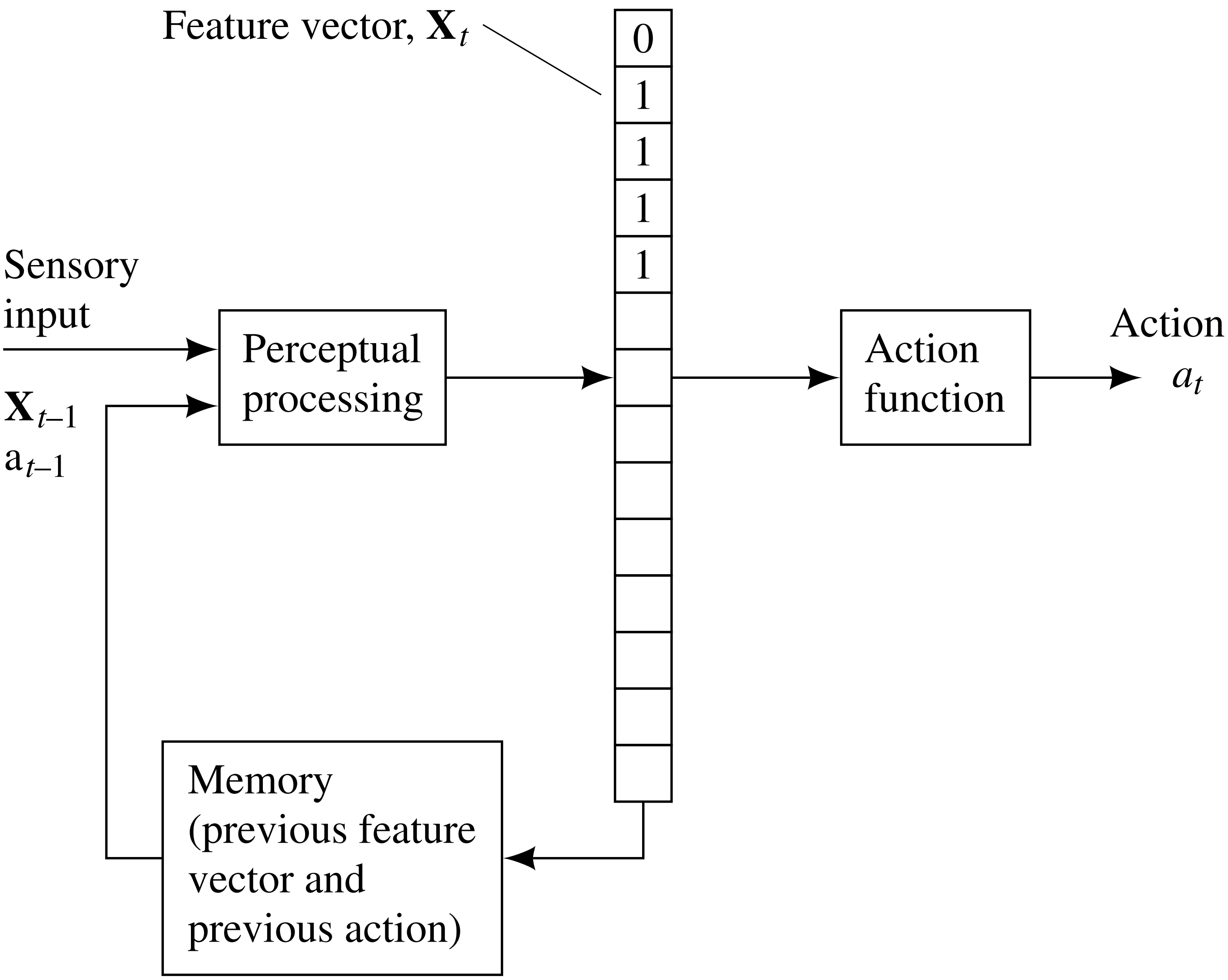
function REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
  static: *state*, a description of the current world state
    rules, a set of condition-action rules

  state ← UPDATE-STATE(*state, percept*)
  rule ← RULE-MATCH(*state, rules*)
  action ← RULE-ACTION[*rule*]
  state ← UPDATE-STATE(*state, action*)
  return *action*

Kuva 7: Muistava heijastetoimija [RN95, kuvat 2.9 ja 2.10].

Havainnoista voi myös eristää *piirteitä (features)* ja muistaa ne. Saadaan tilakone jolla on

- syötteenä seuraava havainto
- sisällään (äärellinen) piirremuisti
- tulosteena seuraava teko.

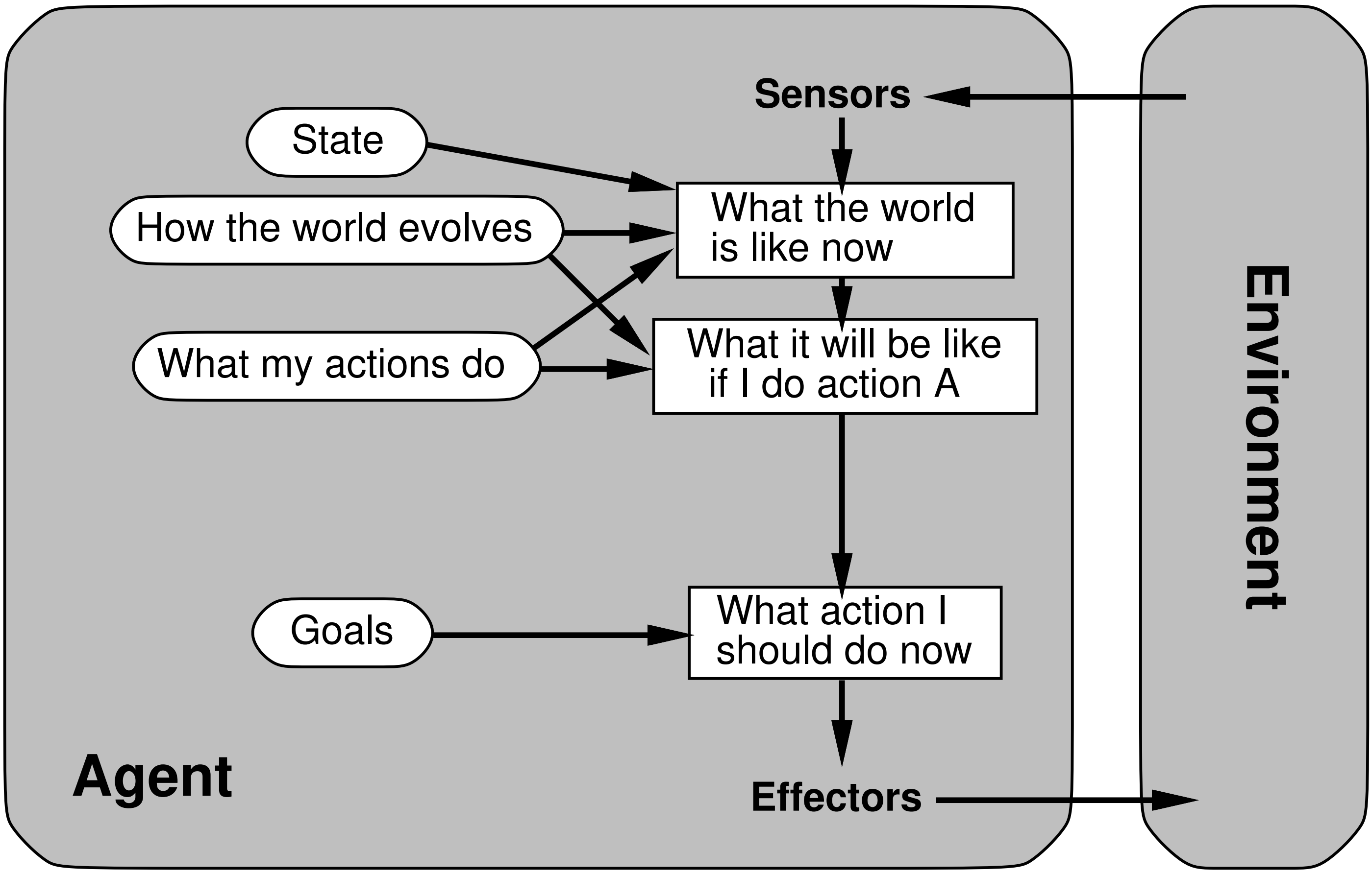


Kuva 8: Piirrevektorin muistava tilakone [Nil98, kuva 5.1].

McCulloch ja Pitts esittivätkin (implisiittisesti) äärelliset automaatit muotoillessaan loogisen teoriansa oppimattomille hermoverkoille (luku 1.5).

2.4. Päämäärähakuiset toimijat

Luvun 2.3 toimijat ovat edelleen heijastetoimijoita. Kuskilla on myös *tavoitteita* jotka ohjaavat tehtäviä valintoja — kuten päästä asiakkaan sanomaan osoitteeseen.



Kuva 9: Päämäärähakuinen toimija [RN95, kuva 2.11].

Toimija, joka "tietää" tavoitteensa, ei enää ole heijastetoimija:

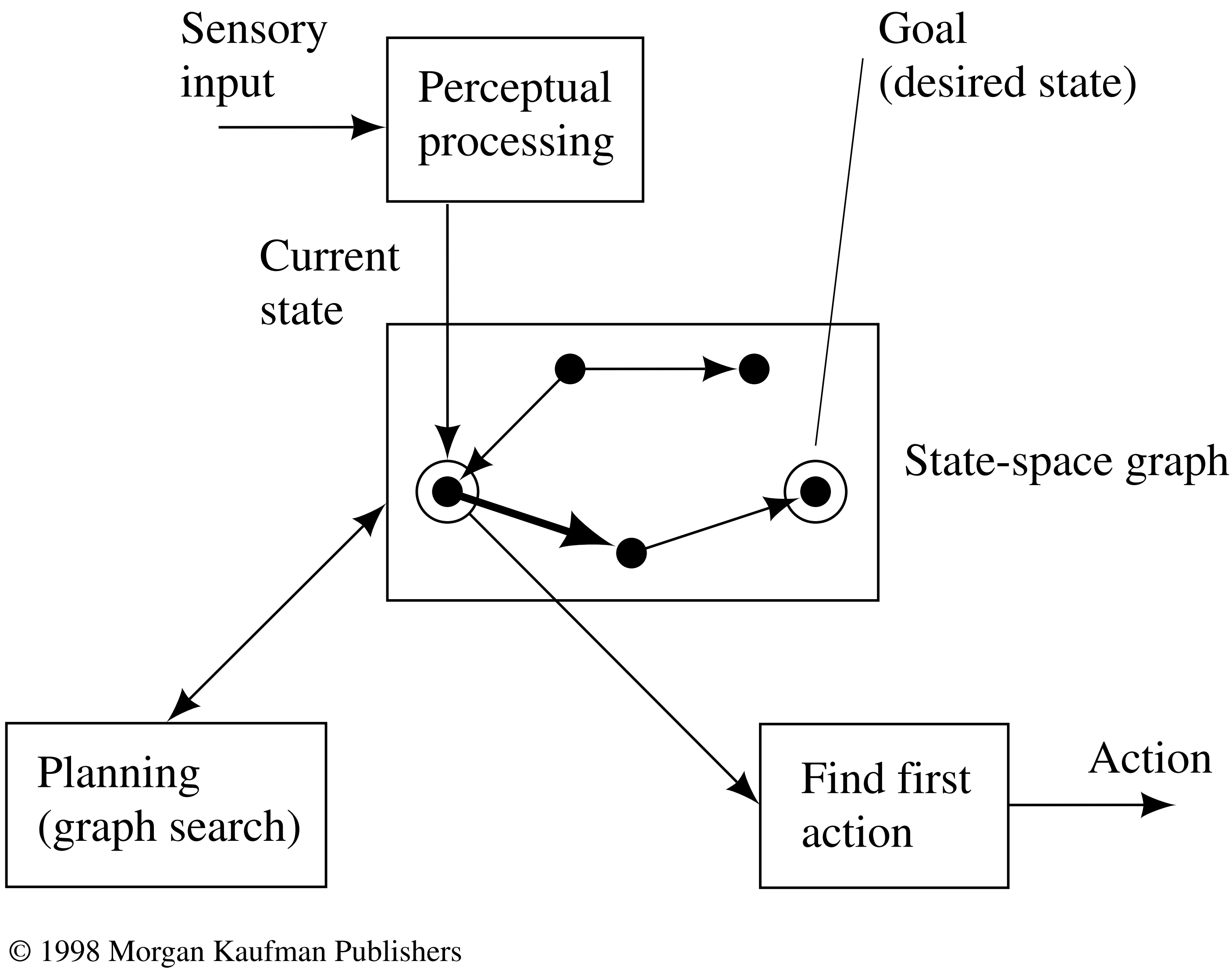
- Samat säännöt saattavat soveltua vaikka tavoitteet vaihtuisivatkin; tavoite voi muuttaa sitä ehtoa, jonka perusteella tekoja pidetään hyvinä tai huonoina.
- Sama tavoite saattaa olla toimijan saavutettavissa vaikka säännöt muuttuisivatkin; toimija voi vaihtaa menetelmää.
- Toimija järkeilee ehdollisesti: "Mitä tästä teostani seura*isi*? Ve*isi*kö se lähemmäs tavoitettani?"

Etsintä on keskeinen tekoälyn alue, koska sen menetelmät tarjoavat tapoja päätellä, onko annettu tavoite saavutettavissa annetuilla säännöillä.

Suunnittelu on toinen keskeinen alue, koska sen menetelmät tarjoavat tapoja osittaa kokonaistavoite osatavoitteiksi ja niiden välisiksi riippuvuuksiksi.

Suunnittelu voidaan usein nähdä etsinnäksi — suunnitelman etsinnäksi.

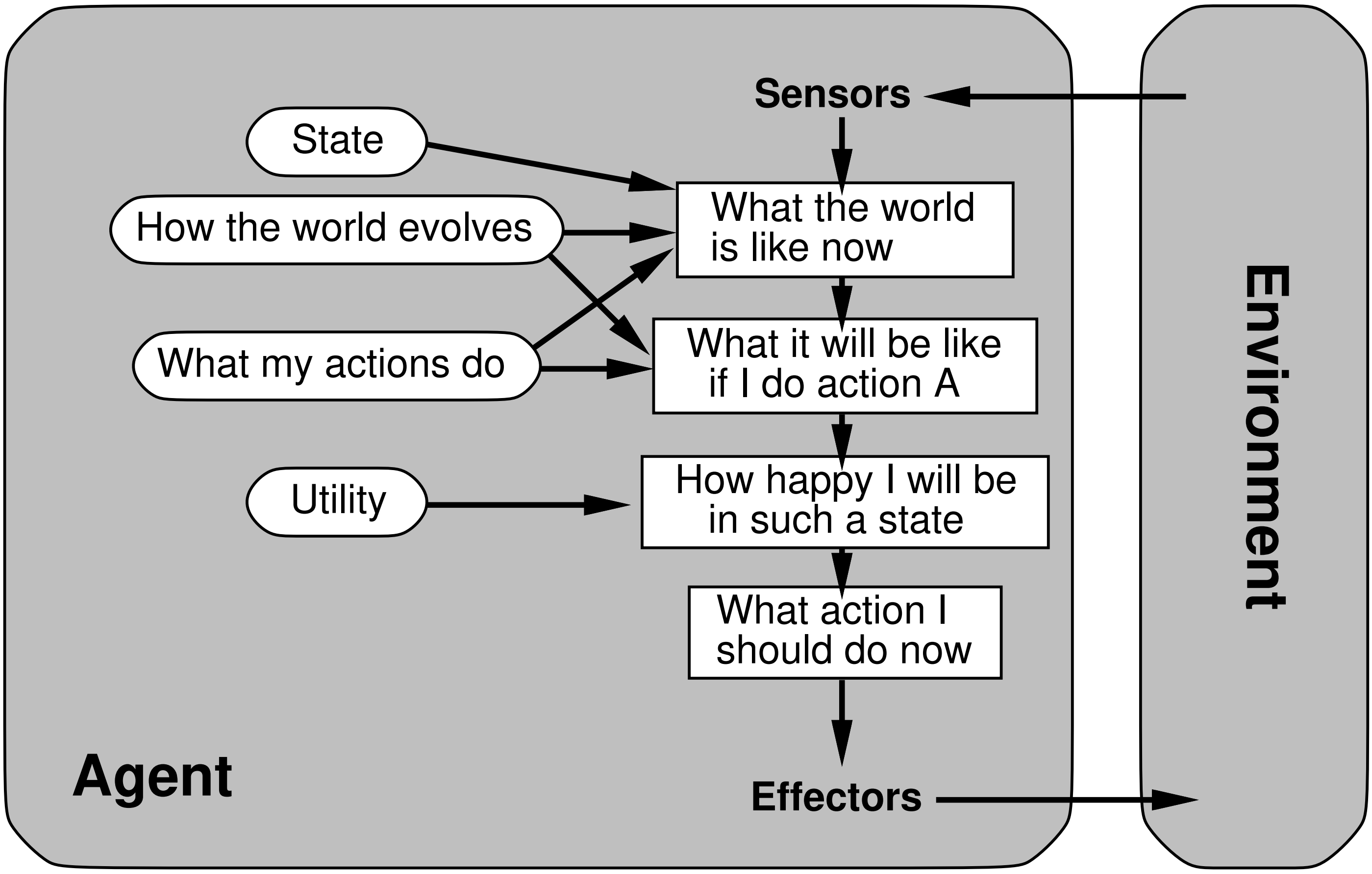
Kurssimme käsitteleekin tämän vuoksi etsintää ja suunnittelua symbolisin välinein.



Kuva 10: Etsimällä suunnittelevan toimijan rakenne [Nil98, kuva 10.1].

2.5. Hyötyohjatut toimijat

Luvun 2.4 päämäärähakuiselle toimijalle kelpaa periaatteessa mikä tahansa jono tekoja, joka lopulta saavuttaa tavoitteen. Mutta asiakas ei ehkä halua ”turistikierrosta” vaan suoremman reitin.



Kuva 11: Hyötyä maksimoiva toimija [RN95, kuva 2.12].

Eri vaihtoehdoista voidaan valita ”paras” vertaamalla ”mahdollisten maailmankuvien” S hyötyjä $U(S)$. Se on myös tapa valita, kun. . .

- kaikkia tavoitteita ei voi saavuttaa yhtä aikaa, tai
- tavoitteiden saavuttaminen on epävarmaa.

Hyötyfunktio U voi myös *ohjata etsintää*.

Hyötyohjattu toimija voidaan joskus toteuttaa luvun 2.4 tehokkaampana päämäärähakuisena toimijana. (Kääntäen, jokaisella rationaalisella toimijalla on jokin maksimoitava hyötyfunktio.)

2.6. Ympäristöoppia

Toimijan suunnitteluun vaikuttavat tietenkin myös sen aiotun toimintaympäristön ominaispiirteet. Ympäristöjä voidaan luokitella seuraavasti:

saavutettavissa vai ei? Ympäristö on saavutettavissa jos kukin havainto sisältää kaiken (tarpeellisen) informaation sen nykytilasta.

Silloin toimijan ei tarvitse käyttää muistiaan ympäristön seuraamiseen.

deterministinen vai epädeterministinen? Ympäristö on deterministinen jos sen nykytila ja toimijan siinä valitsema teko määräävät täysin sen seuraavan tilan.

Silloin toimijan ei *periaatteessa* tarvitse toimia epävarman tiedon pohjalta. Mutta jos ympäristö ei olekaan saavutettavissa, tai jos sen kaikkien ominaisuuksien seuraaminen on liian vaikeaa, niin se saattaa *näyttää toimijasta* epädeterministiseltä.

jaksoittainen vai ei? Ympäristö on jaksoittainen, jos toimijan "elinkaari" voidaan jakaa erillisiin jaksoihin. Kukin jakso koostuu siitä, että toimija havainnoi ensin ympäristöään, ja tekee sitten tekonsa. Eri jaksot eivät vaikuta toisiinsa.

Jaksoittaisissa ympäristöissä toimijan ei tarvitse suunnitella tulevaisuuttaan, koska kunkin jakson paras teko riippuu vain sen aluksi tehdyistä havainnoista.

staattinen vai dynaaminen? Ympäristö on dynaaminen, jos se saattaa muuttua toimijan miettiessä mitä tehdä seuraavaksi.

Staattisissa ympäristöissä toimijan ei tarvitse seurata ympäristöä koko ajan eikä välittää ajan kulumisesta.

Jos ympäristö ei muutu, mutta ajan kulumisesta pitääkin välittää, on se *semistaattinen*.

diskreetti vai jatkuva? Ympäristo on diskreetti, jos se koostuu rajallisesta määrästä selkeästi toisistaan erottuvia mahdollisia havaintoja ja tekoja.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Kuva 12: Erilaisia ympäristöjä [RN95, kuva 2.13].

Yleensä toimijat suunnitellaan ennalta sovittuun *ympäristöluokkaan*, joissa jotkut peruspiirteet pysyvät toisten vaihdellessa. Esimerkiksi shakkiohjelma toimii ympäristöissä, joissa

- vastapelaaja vaihtelee
- mutta pelin perussäännöt (laudan ruutujen määrä, nappuloiden luvalliset siirrot,. . .) pysyvät.

Toimijan menestyksellisyyttä arvioidaan sen perusteella, miten onnistuneesti se toimii ympäristöluokkansa eri jäsenissä.

Jos ympäristöluokassa on jokin luonteva todennäköisyysmitta, niin voidaan tarkastella myös toimijan keskimääräistä/odotettavissa olevaa menestystä.

3. Ongelmanratkaisumenetelmiä [RN95, Nil98, osa II]

Tarkastellaan päämäärähakuisia toimijoita (luku 2.4), joiden tavoitteena on löytää sellainen tekojono, joka saavuttaa päämäärän ympäristön nykytilasta alkaen. Siis *ensin suunnitellaan, sitten tehdään* (jos viitsitään).

(Tekojonon) *etsintä* on siis keskeinen ongelma(nratkaisumenetelmä)!

Erotetaan toisistaan

ympäristön tila eli se miten asiat todellisuudessa ovat

etsinnän tila eli se mitä ympäristön tilasta on selvitetty etsinnän kuluessa.
Pois jätetään siis ne ympäristön piirteet

- joilla ei ole merkitystä nykyisessä etsintäongelmassa
- tai joista etsijä ei voi saada varmaa tietoa.

Esimerkki: *imuriympäristö*, joka koostuu 2 vierekkäisestä ruudusta, joissa voi olla likaa ja/tai itse imuri. Imurin teot:

L: Jos imuri on nyt oikeanpuoleisessa ruudussa, niin se siirtyy vasemmanpuoleiseen ruutuun.

R: Jos imuri on nyt vasemmanpuoleisessa ruudussa, niin se siirtyy oikeanpuoleiseen ruutuun.

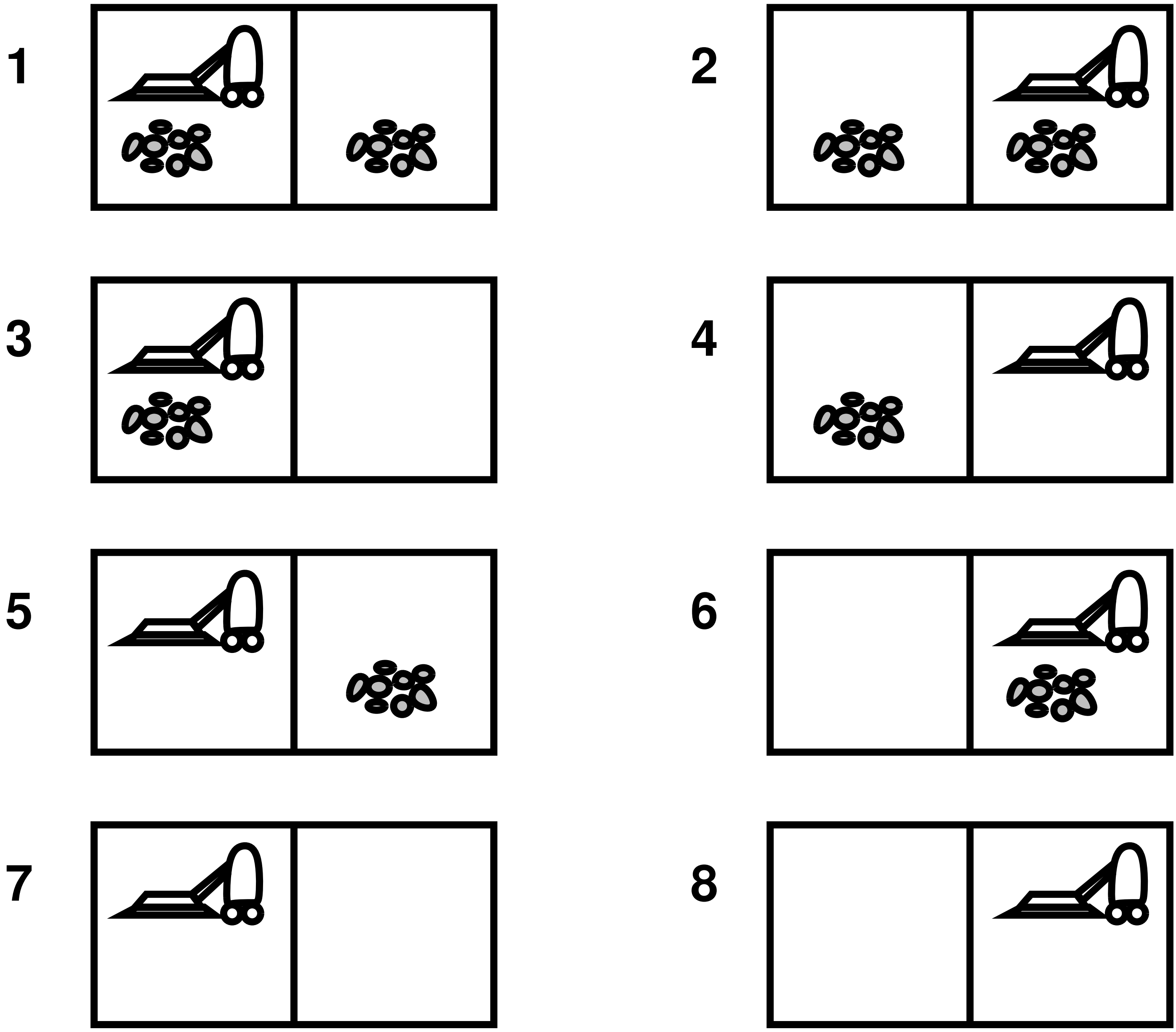
S: Jos siinä ruudussa, missä imuri on nyt, on likaa, niin se imetään pois.

Tavoitteena on imeä pois kaikki lika.

Tällä *ympäristöllä* on $2 \times 2^2 = 8$ mahdollista tilaa (kuva 13):

- Imurilla on 2 mahdollista ruutua.
- Ruudussa joko on tai ei ole likaa.

Tavoite on päästä tilaan 7 tai 8.



Kuva 13: Imuriympäristön mahdolliset tilat [RN95, kuva 3.2].

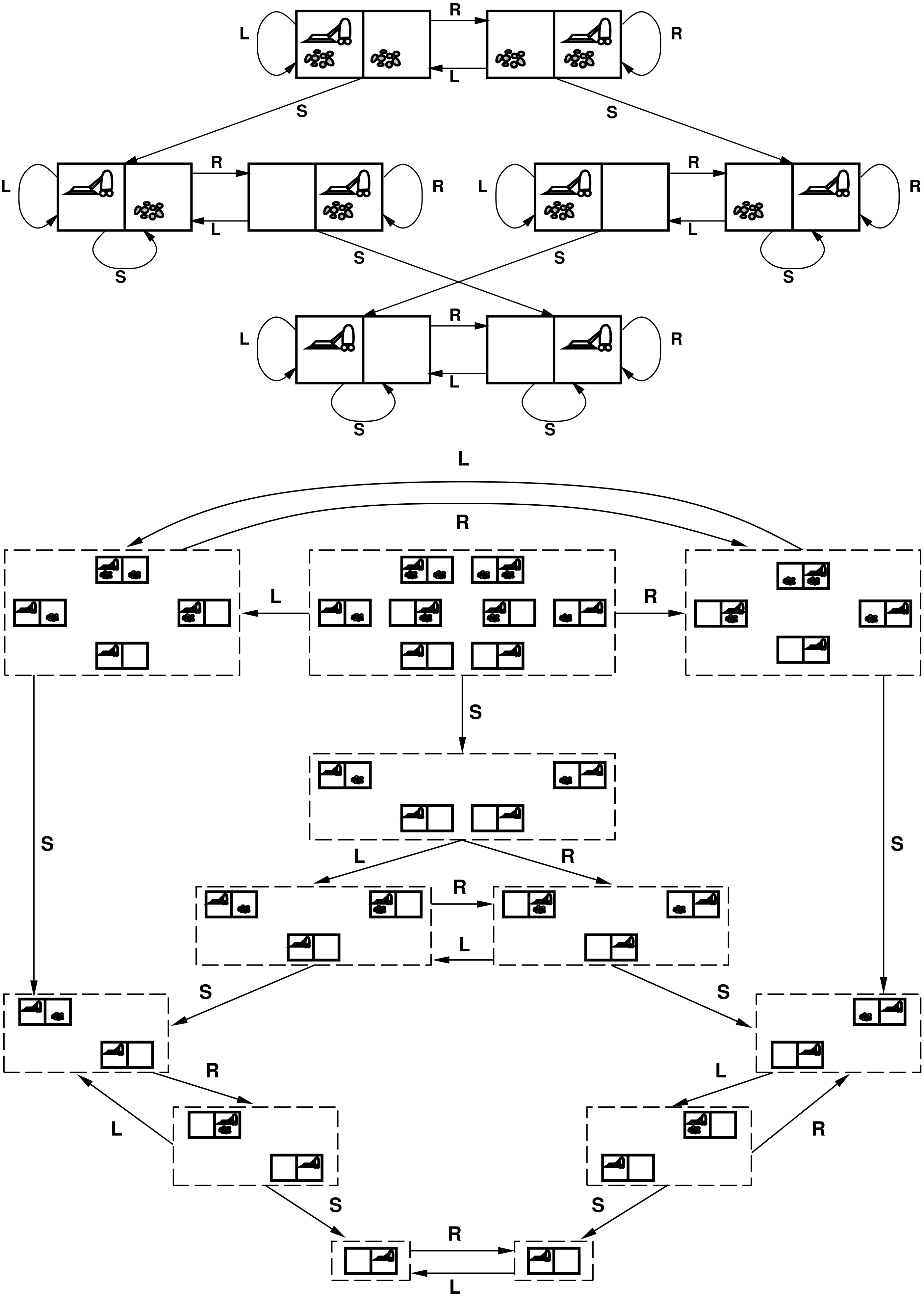
- Jos ympäristö on imurin *saavutettavissa* ja *deterministinen*, niin silloin etsinnän tilat yhtyvät ympäristön tiloihin (kuvan 14 yläpuolisko).

  Kyseessä on *yksitilainen (etsintä)ongelma*: jokaisessa etsinnän tilassa on tasan yksi ympäristön tila.

  Puhutaan *etsintäavaruudesta*: etsinnän tiloista ja niiden välisistä tekoreiteistä.

  Etsijä kulkee tässä avaruudessa kohti tavoitetilaa, ja ratkaisu on näin kuljettu *polku*.

  Silloin imuri voi esimerkiksi havaita ympäristön olevan tilassa 5, ja päätellä että tekojonolla **RS** se pääsee tavoitetilaan 8.



Kuva 14: Yksi- ja monitilainen ongelma [RN95, kuvat 3.6 ja 3.7].

- Entä jos ympäristö ei olekaan saavutettavissa — jos vaikka imuri ei aisti sijaintiaan eikä likaa?

  Silloin jokaiseen avaruuden tilaan kuuluvat *kaikki mahdolliset* ympäristön tilat — *moni*tilainen ongelma (kuvan 14 alapuolisko).

  Imuri käynnistyy avaruuden tilassa $\{1, \ldots, 8\}$; kaikki ympäristön tilat ovat vielä mahdollisia.

  Jos imuri aloittaa teolla **S**, niin sen jälkeen on mahdotonta että imurin oma ruutu (kumpi se onkin) olisi likainen; mahdolliset ympäristön tilat ovat $\{4, 5, 7, 8\}$.

  Tavoite on päästä avaruuden tilaan S, jonka *kaikki* jäsenet ovat siistejä, eli $(\emptyset \subsetneq) S \subseteq \{7, 8\}$.

  Esimerkiksi tekojono **RSLS** saavuttaa tavoitteen pääsemällä avaruuden tilaan $\{7\}$.

  Vertaa epädeterministisen automaatin simulointi!

- Epädeterministinen ympäristö voidaan mallintaa samoin: kaaret ovat silloin muotoa

$$P \xrightarrow{t} \{q : \text{jossakin } p \in P \text{ teosta } t \text{ saattaa seurata } q\}$$

  missä p ja q ovat jälleen ympäristön tiloja. Nyt verkon solmujen voi ajatella edustavan toimijan *tietoa* ympäristöstään.

- Jos ympäristö ei ole saavutettavissa eikä deterministinen, voi ratkaisu jäädä löytymättä:
  - Puuttukoon imurista sensori, joka kertoisi onko *naapuri*ruutu likainen.
  - Vuotakoon imuri siten, että **S** saattaa pudottaa likaa aikaisemmin puhtaseen ruutuun.

  Silloin seuraava tekojono *ei* olekaan ratkaisu:

$$\{1, 3\} \xrightarrow{\mathbf{S}} \{5, 7\} \xrightarrow{\mathbf{R}} \{6, 8\} \xrightarrow{\mathbf{S}} \{6, 8\}$$

  Oikeanpuoleisessa ruudussa saa imuroida *vain jos siinä on likaa*.

  Ympäristöä on nyt havainnoitava löydetyn ehdokastekojonon *suorituksen aikana*, jolloin *yksi etukäteen löydetty* jono ei enää riitäkään varmaksi ratkaisuksi.

Tällaista ongelmaa kutsutaan *varautumisongelmaksi (contingency problem)*: toimijan on varauduttava siihen, että kun hän alkaa toteuttaa valitsemaansa toimintasuunnitelmaa, niin sen kuluessa voi ilmetä jotakin sellaista, mikä vaatii suunnitelman muuttamista.

Valtaosa tosielämän ongelmista on tätä laatua: taksikuksi ei sulje silmiään päätettyään mitä reittiä yrittää viedä matkustajansa perille.

Varautumisongelmia lähestytään kahdella tapaa:

Vuorottelemalla *(interleaving)* suunnitelman etsintää ja toteutusta:
  Etsitään toimintasuunnitelma (ainakin) siihen saakka kunnes ympäristöä on taas havainnoitava, ja toteutetaan se.
  Sitten havainnoidaan, ja jatketaan tulosten perusteella etsintää.
  Hyötynä on, että toimeen päästään tarttumaan vaikkei kaikkea informaatiota olisikaan saatavilla.
  Haittana on, että toimija voi valita suunnitelman, joka näyttää aluksi hyvältä, koska sitä ei ole mietitty "loppuun asti".
    Tai suunnitelman, joka on arka yllättäville muutoksille ympäristössä.
  Tuttu vuorottelun erikoistapaus ovat kahden hengen *pelit*: kumpikin pelaaja miettii useita siirtoja eteenpäin ajatellen "mitä tuo toinen tekisi, jos minä itse tekisin näin?"
  Jossakin vaiheessa tehdään siirto — vaikkei miettiminen olisikaan loppunut — ja katsotaan toisen vastaus siihen ennen kuin jatketaan miettimistä muuttuneessa pelitilanteessa.
  Hyötynä on, että päästään siirtämään laskematta kaikkia vaihtoehtoja (mikä olisi käytännössä mahdotonta).
  Haittana on, että valittu pelistrategia olisi voinut osoittautua kehnoksi jos olisi mietitty "vielä pari siirtoa eteenpäin".

Suunnittelulla *(planning)*, joka tuottaa yhden tekojonon sijasta *havaintojen ehdollistamia jonoperheitä* tai "ohjelmia".
  Jos imuri vuotaa mutta tunnistaa lian ruudussaan (havainnoilla $\mathbf{k}/\mathbf{e}$), niin hyvä suunnitelma on havaintoihin sopiva säännöllisen lausekkeen

$$(\mathbf{k}.\mathbf{S})^*.\mathbf{e}.\mathbf{R}.(\mathbf{k}.\mathbf{S})^*$$

  mukainen jono. Tai ohjelmana:

```
while k do
   S
end while
R
while k do
   S
end while
```

Usein tällainen ratkaisu löytyy *osittamalla* tavoite välitavoitteiksi: tässä kumpikin **while** saavuttaa välitavoitteen ”siivoa nykyinen ruutu”. Vertaa silmukkainvariantit!

Hyötynä on, että ennen tekojen valintaa on hankittu ”kokonaiskuva” tilanteesta, mikä välttää lyhytnäköiset ratkaisut.

Haittana on, että tämän kokonaiskuvan muodostaminen on vaikeaa, kallista ja tehtävä ennen kuin päästään toimeen.

- Vaikein autonomisen toimijan ympäristö on sellainen, jossa se ei tiedä etukäteen edes omien tekojensa vaikutuksia. Silloin sen on *kokeiltava* ”omia voimiaan” ja *opittava* näin erottamaan toisistaan ne ympäristön muutokset, jotka johtuvat sen teoista, ja ne jotka tapahtuvat siitä huolimatta. Nämä ongelmat [RN95, luku 20] ohitetaan tällä kurssilla.

Määritellään jatkossa tarkasteltavat *etsintäongelmat (search problem)* koostumaan seuraavista osista:

Alkutila *(initial state)* on se etsinnän tila, jossa etsintää suorittava toimija tietää nyt olevansa.

Etsijä ei välttämättä tiedä ympäristön tilaa; voihan kyseessä olla monitilainen etsintäongelma.

Operaattorit *(operator)* joilla mallinnetaan ne *teot*, jotka kussakin etsinnän tilassa ovat etsijälle mahdollisia.

Operaattori kuvaa kunkin teon t kertomalla, mihin etsinnän tilaan Q_t päästään tekemällä t etsinnän nykytilassa P.

Voidaan olettaa, että kullekin P ja t on enintään yksi kohdetila Q_t: muuten kyseessä onkin epädeterministinen ympäristö, joten vaihdetaan monitilaiseen etsintäongelmaan ja kerätään jokainen vaihtoehtoinen Q_t yhteen kuvan 14 alapuolen tapaan.

Joskus tämä etsintäongelman osa muotoillaan **seuraajafunktio**na *(successor function)*

$$S(P) = \{\text{tila } Q_t\colon Q_t \text{ saadaan tilasta } P \text{ jollain teolla } t\}\ .$$

Etsintäavaruus *(search/state space)* on alkutilan ja operaattoreiden määräämä (mahdollisesti ääretön) suunnattu verkko, jonka

solmut ovat ne etsinnän tilat, joihin päästään alkutilasta jollakin jonolla tekoja, ja

kaaret ovat operaattorit $P \xrightarrow{t} Q_t$ kaikilla solmuilla P ja teoilla t

Esimerkkeinä olkoot kuvan 14 kaksi verkkoa.

Maaliehto *(goal test)* kertoo kustakin etsinnän tilasta, onko tavoite siinä saavutettu.

Etsintäongelman **ratkaisu** *(solution)* on mikä tahansa polku (tai sitä vastaava tekojono) sen alkutilasta johonkin maaliehdon täyttämään solmuun.

Polkuhinta *(path cost)* on funktio, joka laskee, paljonko annetun polun (= tekojonon) suorittaminen toimijalle ”maksaisi” resursseina.

Tällä kurssilla koko polun

$$\mathcal{P} = Q_0 \xrightarrow{t_1} Q_1 \xrightarrow{t_2} Q_2 \xrightarrow{t_3} Q_3 \xrightarrow{t_4} \cdots \xrightarrow{t_n} Q_n$$

$$\mathrm{hinta}(\mathcal{P}) = \sum_{1 \leq i \leq n} \mathrm{hinta}(t_i)$$

on sen kunkin teon $\mathrm{hinta}(t_i) > 0$ summa.

Kaikkiin ongelmiin ei liity hintoja: silloin valitaan $\mathrm{hinta}(t) \equiv 1$ ja toivotaan lyhyttä ratkaisua.

Etsintäongelmien ratkaisumenetelmää M voidaan arvioida (ainakin) neljällä tavalla:

- Löytääkö M ratkaisun aina kun sellainen on olemassa?

Silloin M on *täydellinen (complete)*.

Vai voiko M epähuomiossa sivuuttaa ratkaisun

– ja pysähtyä vastaukseen "ei ole"

– tai olla pysähtymättä.

- Löytääkö M aina hinnaltaan halvimman ratkaisun *(online cost)*?

  Silloin M on *optimaalinen (optimal)* eli sen ehdottama tekojono on toimijalle "vähiten rasittava".

  Tai kuinka lähelle parasta ratkaisua M (yleensä) pääsee?

- Kuinka paljon *etsintä*resursseja (= muistia ja prosessoriaikaa) M kuluttaa *(offline cost)*?

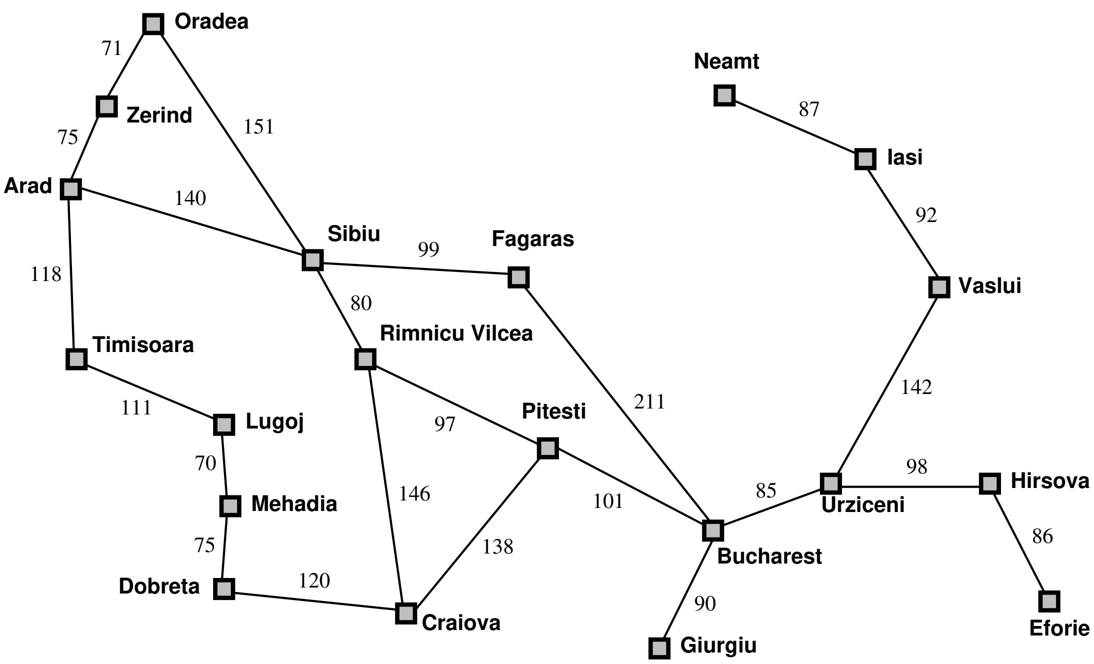
  Eli paljonko toimija käyttää voimiaan "miettimiseen"?

- Kuinka paljon M kuluttaa

$$\textit{kokonais}\text{resursseja} = \text{offline cost} + \text{online cost}$$

  (mahdollisesti painotettuna löydetun ratkaisun käyttökertojen lukumäärällä).

Tarkastellaan ongelmaa valita ajoreitti Aradista Bukarestiin.



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

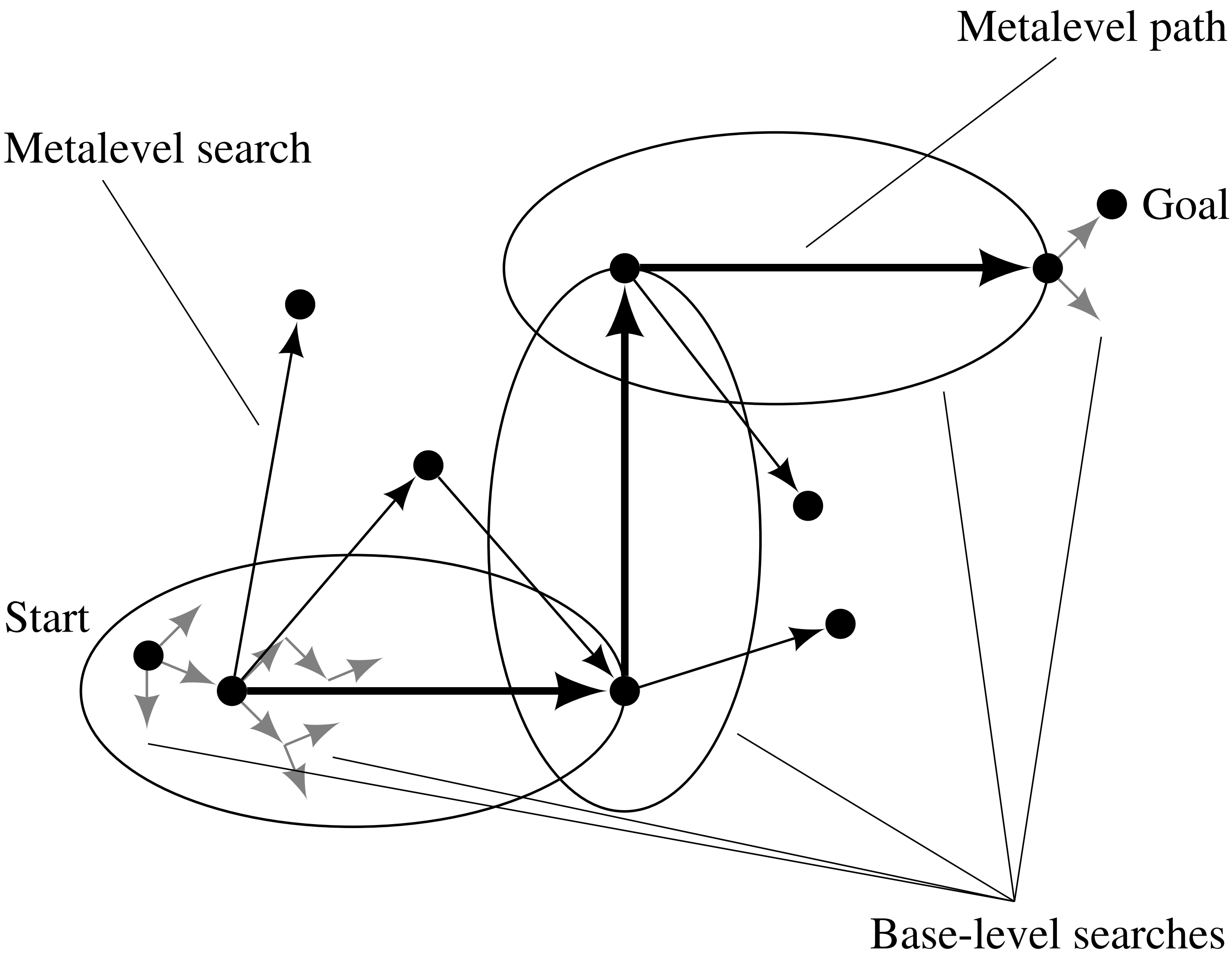
Kuva 15: Osa Romanian tieverkkoa. [RN95, kuva 4.2].

Alkutila on se, jossa olemme Aradissa.

Operaattorit valitaan viemään meidät kaupungista johonkin naapurikaupunkiin.

Tällä tasolla *pelkistämme (abstract)* ongelman kuvauksesta pois tarpeettomat ympäristön piirteet.

Pelkistämme pois myös tarpeelliset mutta liian yksityiskohtaiset piirteet: millaiset ohjaustoimenpiteet tarvitaan auton saamiseksi tästä parkkiruudusta kadulle, kadulta moottoritielle,. . . Ne voidaan selvittää *alemman pelkistystason* etsintäongelmana, kunhan ensin päätetään, mihin naapurikaupunkiin seuraavaksi suuntaamme. Tämä on *hierarkista* suunnittelua.



Kuva 16: Hierarkinen etsintä [Nil98, kuva 10.3].

Onnistunut abstraktio on välttämätöntä (muttei riittävää) onnistuneeseen etsintään!

Ja vaikeaa: pitäisikö esimerkiksi ottaa mukaan autokorjaamojen määrä kullakin reitillä — varmuuden vuoksi?

Maaliehto on, että olemme Bukarestissa.

Polkuhinta valitaan kuvan 15 informaation perusteella lyhimmäksi kokonaismatkaksi.

Ratkotaan tällaisia etsintäongelmia seuraavalla yleisellä algoritmi(skeema)lla:

function GENERAL-SEARCH(*problem*, QUEUING-FN) **returns** a solution, or failure

nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[*problem*]))
loop do
  if *nodes* is empty **then return** failure
  node ← REMOVE-FRONT(*nodes*)
  if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **then return** *node*
  nodes ← QUEUING-FN(*nodes*, EXPAND(*node*, OPERATORS[*problem*]))
end

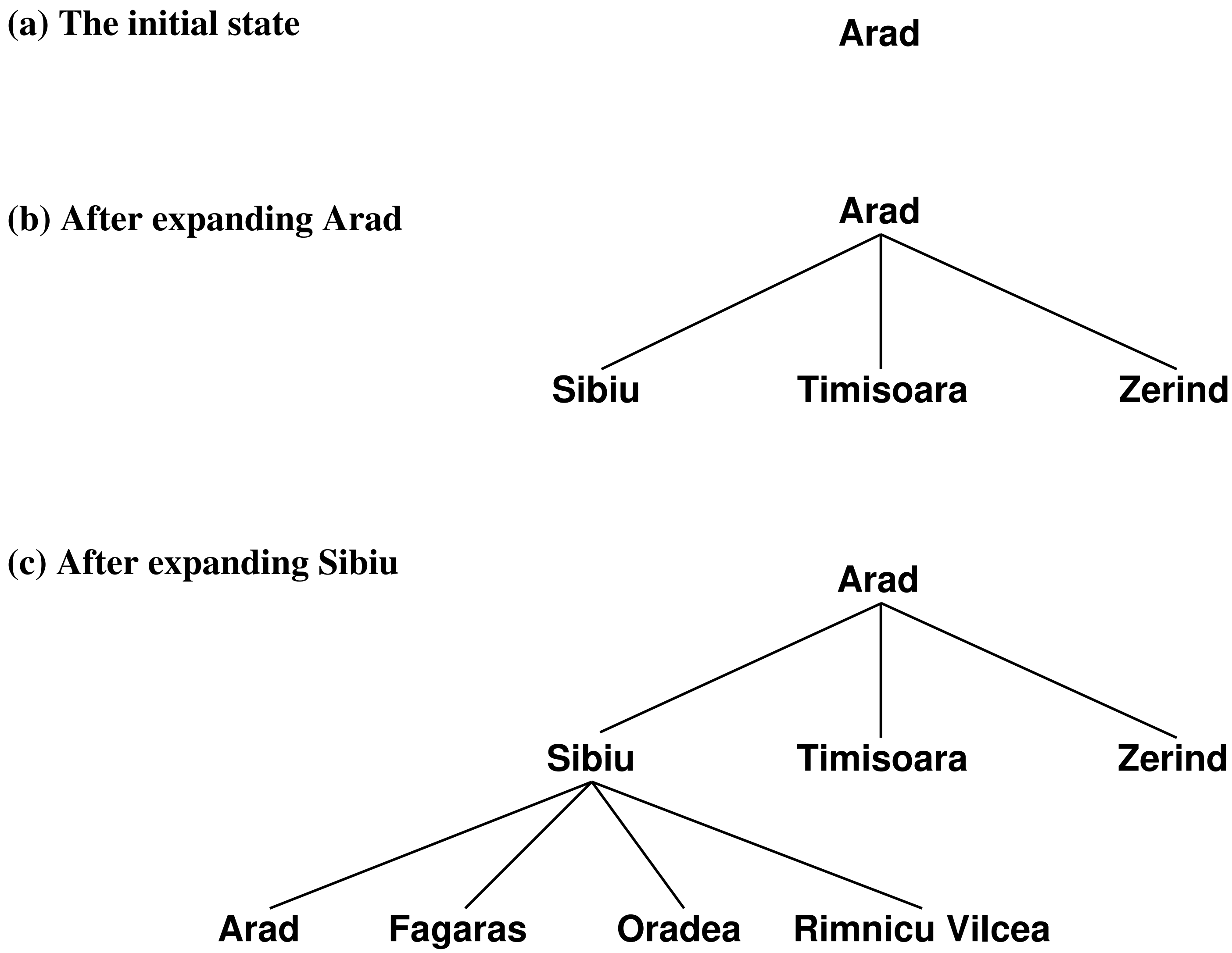
Algoritmi 2: Yleinen etsintäalgoritmi [RN95, kuva 3.10].

Parametrina saatu aliohjelma QUEUEING-FN tallettaa *työmuistiin* nimeltä *nodes* etsinnän eri vaihtoehtoja, joista REMOVE-FRONT poimii aina seuraavan tarkasteltavaksi.

Työmuistin ominaisuuksien valinnalla päätetään *etsintästrategia (search stratgy)* eli se järjestys, jossa eri vaihtoehtoja harkitaan.

Sovelletaan etsintäalgoritmia 2 ongelmaamme:

1. Aloitamme Aradista (kuva 17a), joka ei ole Bukarest, joten se on *laajennettava (expand)* naapureikseen (kuva 17b).
2. Strategiamme valitkoon naapurikaupungeista Sibiun kokeiltavaksemme (kuva 17c).
3. Jatkamme siitä kuten askeleessa 1, jne.



Kuva 17: Etsintäpuun kasvu. [RN95, kuva 3.8].

Strategian jäljittäminen tuottaa *etsintäpuun (search tree)*:

juuri on alkutila

solmut vastaavat etsintäavaruuden *polkuja*

lapset saadaan laajentamalla isäsolmun polkua yhdellä teolla

lehdistä valitaan yksi kullakin kierroksella laajennettavaksi strategian määräämässä järjestyksessä.

- Etsintäpuu kertoo, miten annettu etsintäalgoritmi käyttäytyy annetussa etsintäavaruudessa.

  Se on rekursiokutsupuun yleistys tapaukseen, jossa vaihtoehtojen käsittelyjärjestys seuraa työmuistin toiminnasta (ohjelmakoodin sijaan).

- Sen ei tarvitse olla algoritmia suorittavan tietokoneen muistissa:
  - jo käsiteltyä ei ole pakko muistaa
    (vaikkapa jos sen mahdollinen uudelleen käsittely ei haittaa)
  - vielä käsittelemätöntä saa unohtaa
    (vaikkapa jos se voidaan palauttaa takaisin käsiteltäväksi myöhemmin).
- Se voi olla *ääretön*, jos
  - avaruus on ääretön
    (jo ympäristö voi olla ääretön)
  - avaruudessa on kehä.

Jatkossa tarkastellaan erilaisia etsintäalgoritmeja antamalla vastaava QUEUEING-FN ja tarkastelemalla näin syntyviä etsintäpuita.

3.1. Systemaattisia etsintämenetelmiä [RN95, luku 3] [Nil98, luku 8]

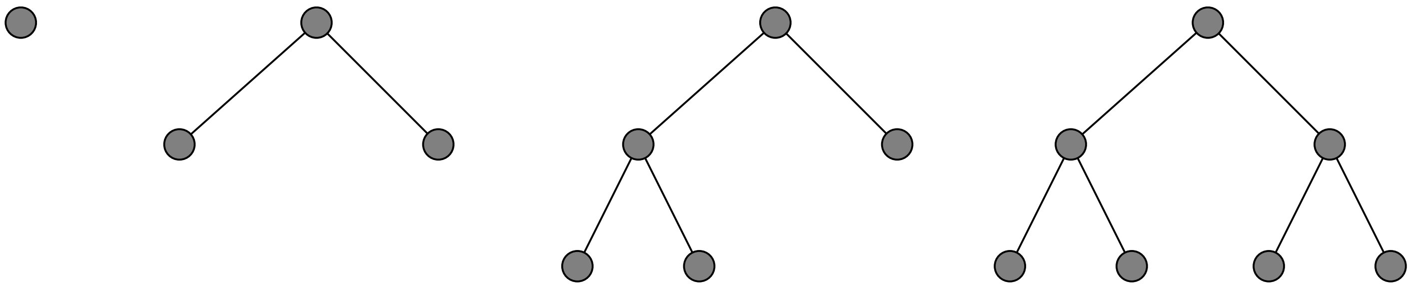
> "Ihaa, joka on ystäväni, on kadottanut häntänsä. Ja hän on ruvennut Murjottamaan. Voisitko ystävällisesti neuvoa minulle miten löytäisin sen?"
>
> "Hmm", sanoi Pöllö. "Tällaisissa tapauksissa Tavanomainen Proseduuri lienee seuraava."
>
> "Mitä tarkoittaa Tavanomainen Proosakuuri?" kysyi Puh. "Sillä minä olen Pieniälyinen Karhu jolle kummalliset sanat ovat Harmi."
>
> "Se tarkoittaa Mitä Tehdään."
>
> "Hyvä jos se tarkoittaa sitä", sanoi Puh nöyrästi.
>
> "Mitä Tehdään lienee seuraava: täytyy julistaa systemaattiset etsinnät. Sitten —"
>
> "Hetki vain", sanoi Puh käpälä pystyssä. "*Mitä* täytyy tehdä? Aivastelit juuri kun olit sanomaisillasi sen."
>
> *A.A. Milne: Nalle Puh*

Etsintämenetelmä on systemaattinen, jos se käy etsintäpuuta läpi jossakin systemaattisessa järjestyksessä hyödyntämättä mitään tietoa itse etsintäongelmasta. Sellaista etsintää kutsutaan myös *sokeaksi (uninformed/blind search*.

Vaikka systemaattinen etsintä onkin tehotonta, niin joskus ongelmastakaan ei tiedetä mitään.

3.1.1. Leveyssuuntainen etsintä (Breadth-First Search)

Jos Algoritmin 2 QUEUEING-FN on *jono (queue)*, niin saadaan (tunnetusti) etsintäpuun leveyssuuntainen läpikäynti:



Kuva 18: Leveyssuuntainen etsintä [RN95, kuva 3.11].

- Se on *täydellinen*.
- Se on myös *optimaalinen, jos* polkuhinta on solmun syvyyden ei-vähenevä funktio.

  Näin on kun kaikki operaattorit maksavat yhtä paljon.
- Ongelmana on *eksponentiaalinen (ajan ja) tilan tarve*:
  - Olkoon $b > 1$ avaruuden solmujen (suurin) *haarautumisaste (branching factor)* (= naapurien lukumäärä).
  - Olkoon lyhimmän ratkaisun (= polun) pituus eli avaruuden *halkaisija (diameter)* d tekoa.
  - Läpikäydyn puun koko (= laajennusten määrä) on silloin enintään

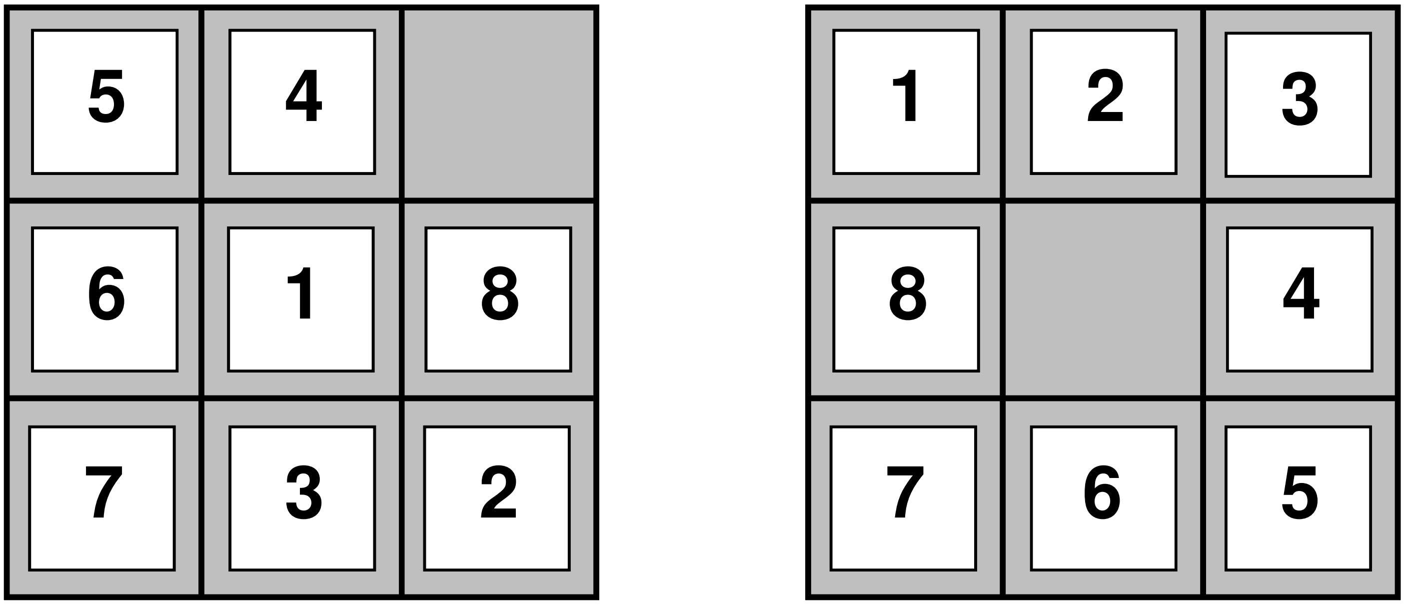
$$T_d = \sum_{0 \leq k \leq d} b^k = \frac{b^{d+1} - 1}{b - 1} \tag{7}$$

solmua, joista enintään b^d on kerrallaan jonossa. (Geometrinen sarja [Myr74b, luku 11.3].)

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Kuva 19: Leveysetsinnän resurssit ($b = 10$) [RN95, kuva 3.12].

Eräs tekoälyn (opetuksen) suosittu leluongelma on lasten liukupeli:



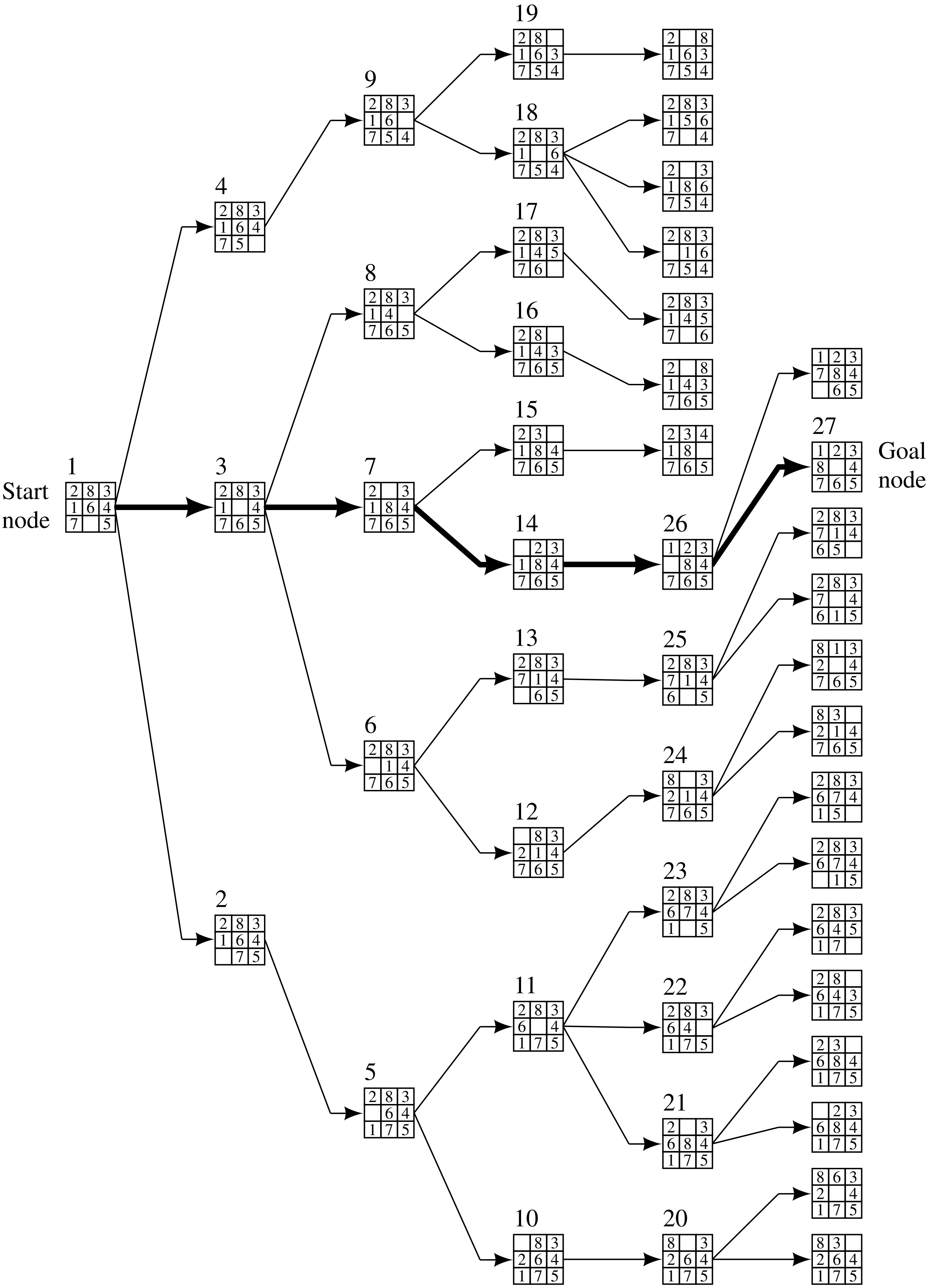
Start State **Goal State**

Kuva 20: 8-liukupeli [RN95, kuva 4.7].

Esimerkiksi kuvan 20 vasemmassa tilanteessa sen oikean yläkulman tyhjään ruutuun voi (yhtenä tekona) liu'uttaa

- joko vasemmanpuolisen naapurin $\boxed{4}$
- tai alapuolisen naapurin $\boxed{8}$

ja tyhjä paikka siirtyy sen tilalle.

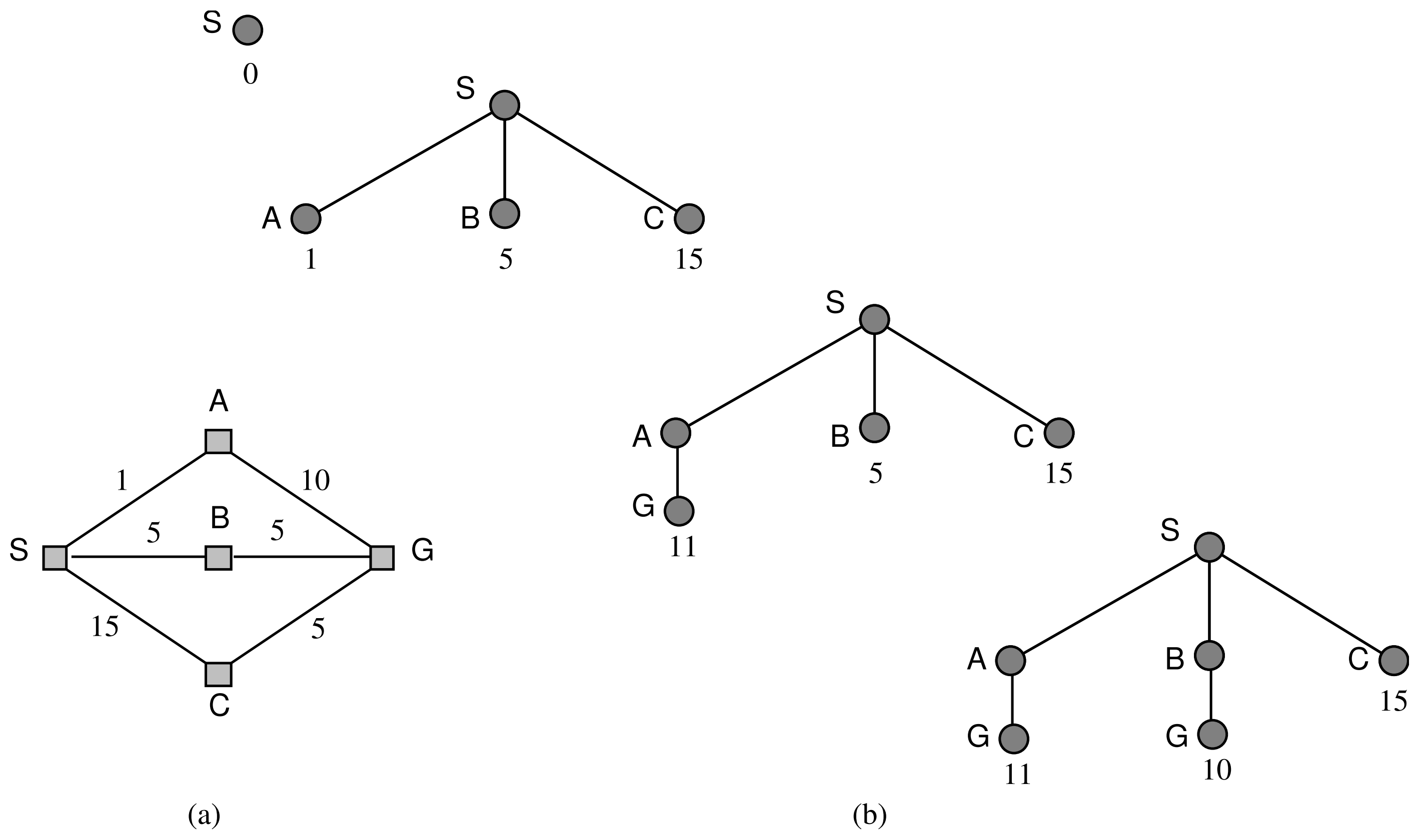


Kuva 21: Leveysetsintä 8-liukupelissä [Nil98, kuva 8.2].

3.1.2. Halvin ensin (Uniform Cost Search)

Luvun 3.1.1 algoritmi löytää sellaisen ratkaisun, jossa on *vähiten tekoja*, mutta jos eri teot ovat eri hintaisia ei se ole välttämättä *paras* valinta.

Algoritmin 2 QUEUEING-FN olkoon siis *keko (heap)* kuhunkin solmuun liittyvän polun (= tekojonon) kokonaiskustannuksen suhteen, joista valitaan aina pienin. Etsitään vaikkapa reitti solmusta S solmuun G:



Kuva 22: Halvin reitti ensin [RN95, kuva 3.13].

Tämä etsintäalgoritmi on *täydellinen ja optimaalinen, jos* polun kokonaishinta ei koskaan laske sitä myöten edettäessä.

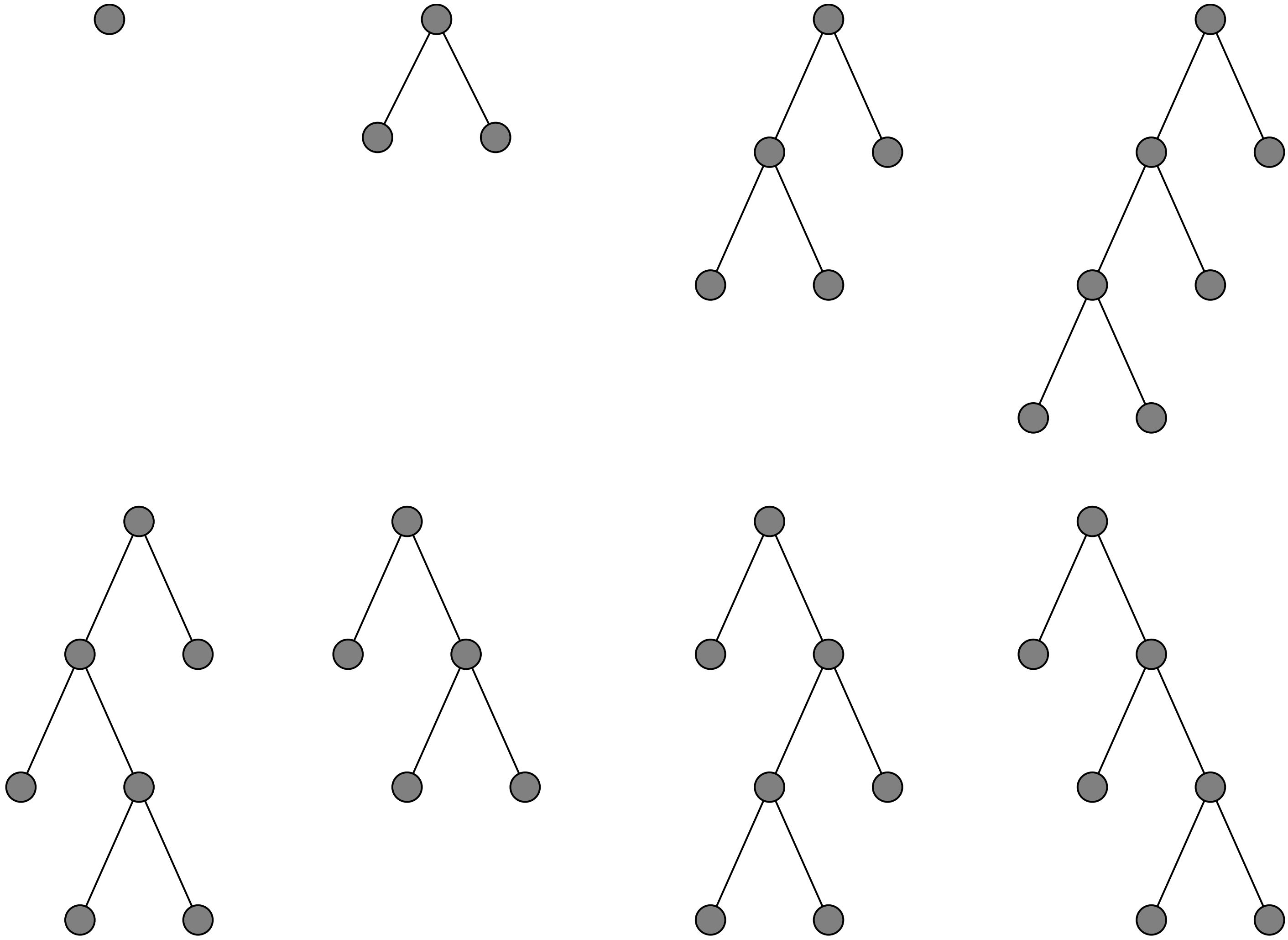
Luentojemme summautuvilla polkuhinnoilla tämä ehto on

$$\mathrm{hinta}(t) \geq 0$$

eli että mikään teko t ei tuota "voittoa" tekijälleen. Siis se on voimassa.

3.1.3. Syvyyssuuntainen etsintä (Depth-First Search)

Jos Algoritmin 2 QUEUEING-FN on *pino (stack)*, niin saadaan (tunnetusti) etsintäpuun syvyysyyntainen läpikäynti:



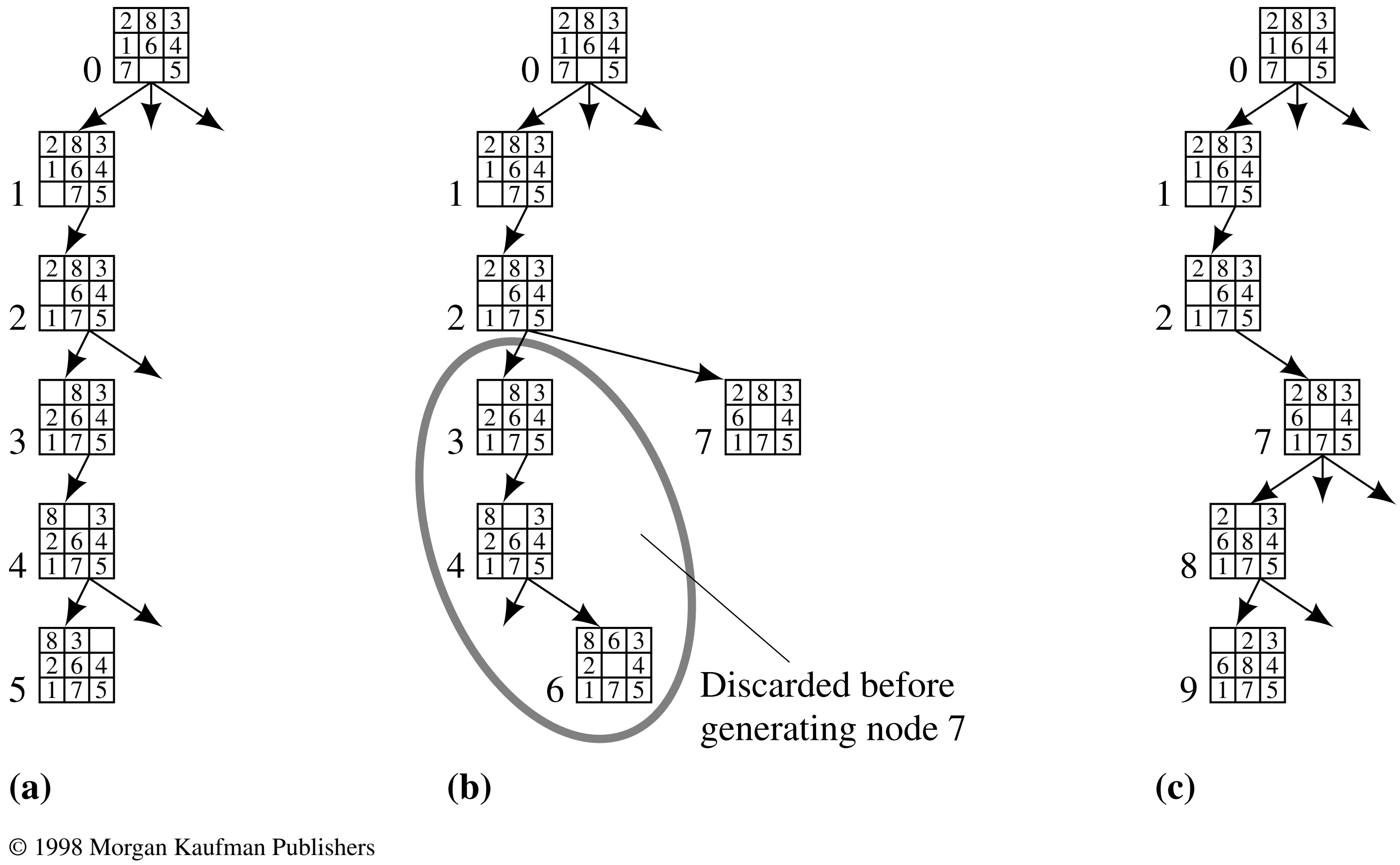
Kuva 23: Syvyyssuuntainen etsintä [RN95, kuva 3.14].

Tämä etsintäalgoritmi tarvitsee *paljon vähemmän muistia* kuin luvun 3.1.1 menetelmä: jos m on etsintäavaruuden pisimmän tekojonon pituus (= etsintäpuun syvyys), niin pinossa on kerrallaan vain $O(bm)$ solmua.

Tämä algoritmi vie pahimmillaan $O(b^m)$ etsintäaskelta, mutta jos avaruudessa on monta maaliehdon täyttävää solmua, se saattaa löytää yhden käymättä läpi kuin murto-osan koko avaruudesta.

Tämä algoritmi *ei ole optimaalinen*, eihän se edes pidä kirjaa eri polkujen kustannuksista.

Tämä algoritmi *ei edes pysähdy, jos* m on ääretön: se seuraa ensimmäistä tapaamaansa etsintäpuun ääretöntä haaraa yhä syvemmälle ja syvemmälle.



Kuva 24: Syvyyssuuntainen etsintä 8-puzzlessa [RN95, kuva 8.3].

Jos etsintäongelman ratkaisijan ohjelmoi *rekursiivisena* niin päätyy johonkin syvyyshaun varianttiin: rekursiokutsupino toimii silloin (näkymättömänä) hakupinona.

Ja kääntäen: jos valitsee menetelmäkseen syvyyshaun, niin sen voi usein *toteuttaa helposti* rekursiolla.

3.1.4. Syvyysrajoitettu etsintä (Depth-Limited Search)

Yksi tapa saada luvun 3.1.3 algoritmi varmasti pysähtymään on lisätä siihen etukäteen yläraja l tarkasteltavien polkujen *pituudelle* (eli seurattavien haarojen syvyydelle): jos nykyinen solmu ylittää rajan, niin sitä ei enää laajennetakaan lapsikseen.

- Tämä syvyysrajoitettu etsintä ei edelleenkään ole optimaalinen.
- Jos $l \geq d$, niin se on täydellinen, muuten taas ei.
- Se vie $O\left(b^l\right)$ etsintäaskelta.
- Se pitää pinossaan korkeintaan $O(bl)$ solmua kerrallaan.

Samaa ideaa voi soveltaa myös antamalla yläraja ℓ polkujen *hinnalle*:

- Pysähtyminen on taattu jos ei ole ääretöntä polkua jonka hinta pysyisi silti äärellisenä.
  Luentojemme summatuvilla polkuhinnoilla tämä tarkoittaa
  $$\mathrm{hinta}(t) \geq \delta$$
  missä $\delta > 0$ on kaikkien tekojen hintojen alaraja.

3.1.5. Iteratiivinen syventäminen (Iterative Deepening)

Luvun 3.1.4 algoritmin soveltamisessa on vaikeaa löytää tarpeeksi muttei liian suuri yläraja l. Jätetään siis tämäkin etsintäalgoritmin huoleksi:

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution sequence
  inputs: *problem*, a problem

  for *depth* $\leftarrow 0$ **to** ∞ **do**
    if DEPTH-LIMITED-SEARCH(*problem*, *depth*) succeeds **then return** its result
  end
  return failure

Algoritmi 3: Iteratiivinen syventäminen [RN95, kuva 3.15].

Tämä algoritmi yhdistää lukujen 3.1.1 ja 3.1.3 hyvät puolet:

- Se on *täydellinen* ja *optimaalinen* kuten luvun 3.1.1 leveyssuuntainen etsintäkin.
- Sen muistintarve on yhtä vaatimaton kuin käyttämällään luvun 3.1.3 syvyyssuuntaisella etsinnälläkin.
- Se tekee tosin useampia etsintäaskeleita kuin luvun 3.1.1 leveyssuuntainen etsintä, koska sama etsintäpuun solmu *laajennetaan monta kertaa*:
  1. Ensin käydään läpi koko korkeuden 0 puu löytämättä vastausta. Siihen kuuluu $T_0 = 1$ solmua (eli vain juuri) yhtälön (7) nojalla.
  2. Sitten läpikäydään korkeuden 1 puun

     $$T_1 = \frac{b^2 - 1}{b - 1} = b + 1$$

     solmua, joista 1 (pelkkä juuri) tutkittiin (muttei vielä laajennettu) jo kerran vaiheessa 1.
  3. Sitten korkeuden 2 puu, jossa juuri käsitellään kolmannen kerran, sen lapset toistamiseen, ja lapsenlapset ensimmäistä kertaa.

Kun näin jatketaan ratkaisusyvyyteen $depth = d$ saakka, on käsiteltyjen solmujen kokonaismäärä

$$\begin{aligned}
\sum_{0 \leq k \leq d} T_k = \sum_{0 \leq k \leq d} \frac{b^{k+1}-1}{b-1} \\
&= \frac{1}{b-1}\left(\left(\sum_{0 \leq k \leq d} b^{k+1}\right) - \left(\sum_{0 \leq k \leq d} 1\right)\right) \\
&= \frac{1}{b-1}\left(b\left(\sum_{0 \leq k \leq d} b^{k}\right) - (d+1)\right) \\
&= \frac{bT_d-(d+1)}{b-1}. \qquad (8)
\end{aligned}$$

- Toisaalta yhtälöstä (8) nähdään, että tämä algoritmi on vain noin

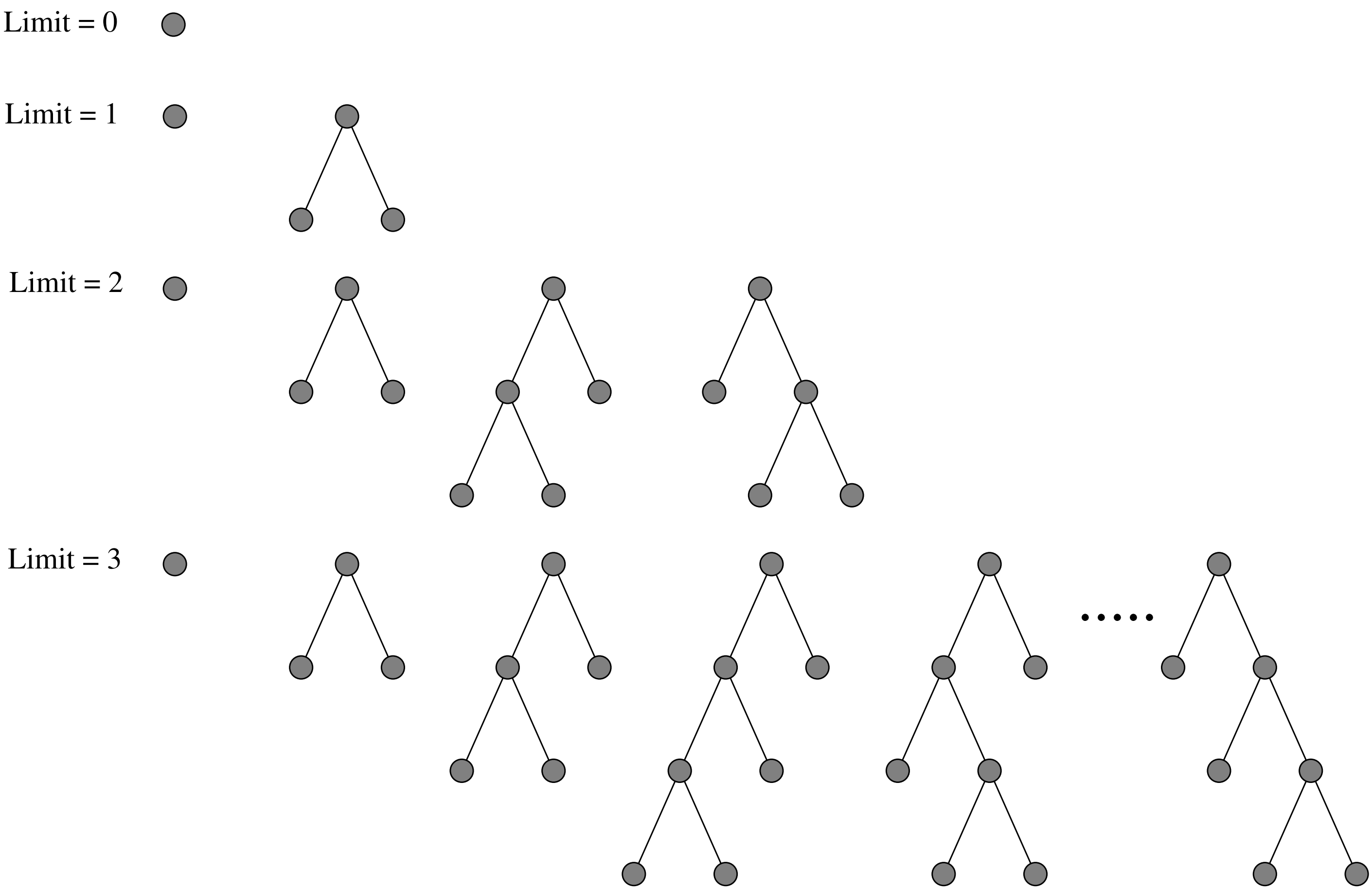
$$\frac{b}{b-1}$$

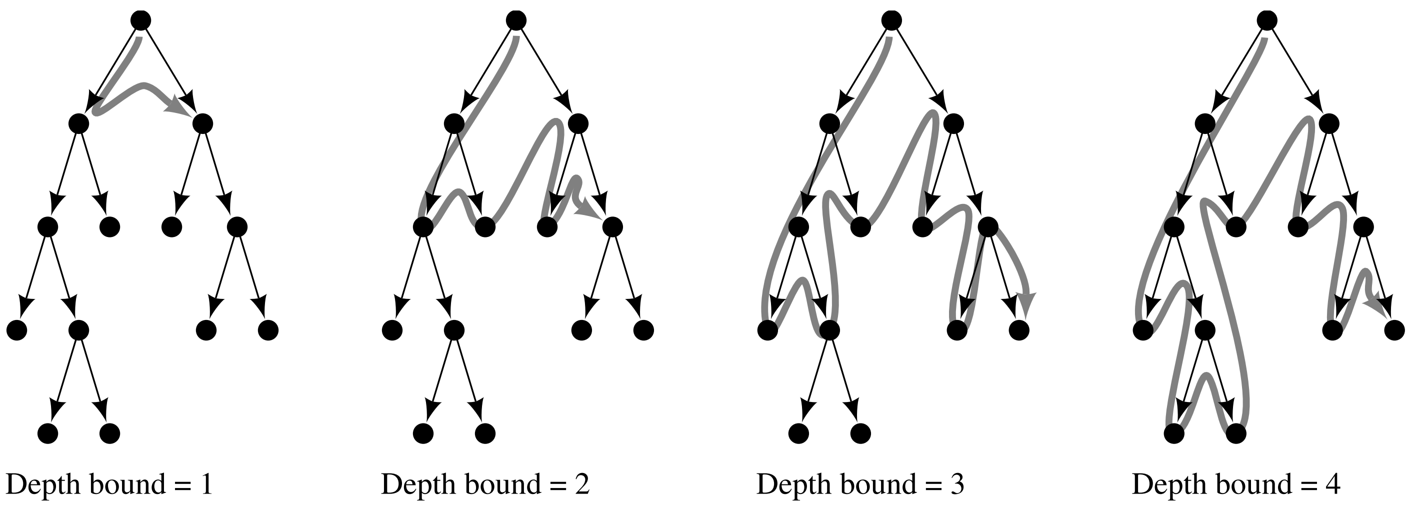
kertaa hitaampi kuin luvun 3.1.1 leveyssuuntainen etsintä:
  - Jos $b = 2$ niin se vie noin kaksinkertaisen ajan, mutta ero kutistuu kun b kasvaa.
  - Kun $b = 10$ ja $d = 5$ niin yhtälö (7) antaa $111\ 111$, ja yhtälö (8) antaa $123\ 456$, jolloin eroa on enää noin 11%.
  - Yleisestikin *suurin osa etsintäpuun solmuista on lehtiä*: jo binääripuissa on aina lehtiä 1 enemmän kuin sisäsolmuja, ja tämä ero vain kasvaa haarautumisasteen b mukana.
  - *Sisä*solmujen käsitteleminen moneen kertaan ei siis vaikuta merkittävästi kokonaistyömäärään.

Iteratiivinen syventäminen onkin paras systemaattinen etsintämenetelmä suurille avaruuksille, joiden halkaisijaa d ei tunneta etukäteen.

Jos samaa ideaa käytetään hintarajan ℓ määräämiseen, niin rajaa voi kasvattaa

- joko systemaattisesti hinta-alarajan δ verran
- tai dynaamisesti pienimmäksi pituudeksi jolla jokin tavattu solmu jäi lapsettomaksi.

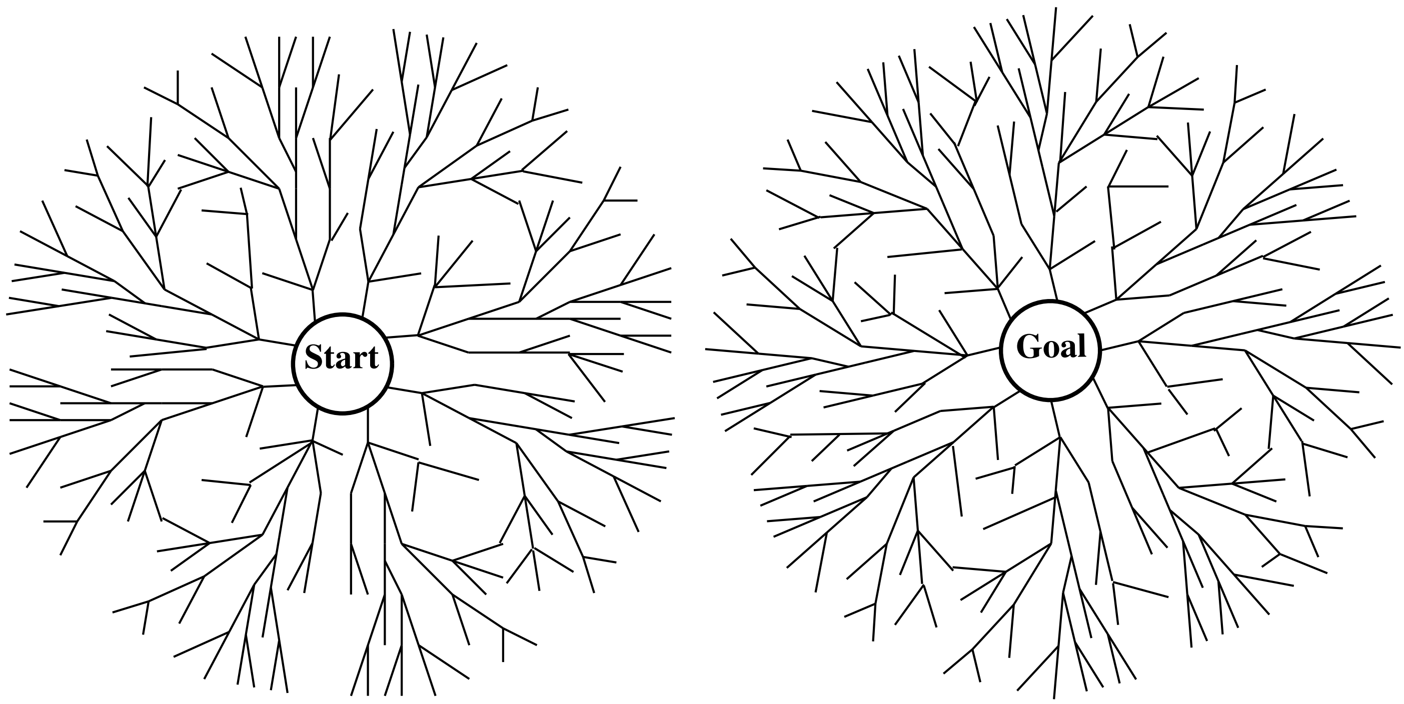




Kuva 25: Syventämisen eteneminen [RN95, kuva 3.16][Nil98, kuva 8.5].

3.1.6. Kaksisuuntainen etsintä (Bidirectional Search)

Entä jos käynnistetään yhtä aikaa yksi etsintä *eteenpäin alkutilasta* ja toinen etsintä *taaksepäin maalitilasta*, ja lopetetaan niiden kohdatessa?



Kuva 26: Kaksisuuntainen etsintä [RN95, kuva 3.17].

Periaatteessa voimme saavuttaa ratkaisun $O(2b^{d/2})$ askeleessa, koska kumpikin puolisko tulee toista puoliväliin vastaan.

Kun $b = 10$ ja $d = 6$, niin yhtälön (7) mukaan syvyyshaku laajentaa $1\ 111\ 111$ solmua, tämä algoritmi taas vain $2\ 222$.

Tekemällä 2 pienempää puuta karsimme lehtien määrää; luvun 3.1.5 havaiinon nojalla tämä kannattaa.

Ensin on kuitenkin ratkaistava seuraavat pulmat:

- Mitä tarkoittaa "etsiä taaksepäin"?
  Intuitiivisesti seurataan etsintäavaruuden nuolia takaperin.
- Onko siis jokaisella operaattorikaarellamme

$$P \xrightarrow{t} Q$$

vastakaari

$$P \xleftarrow{t^{-1}} Q$$

joka palauttaa tilanteen sellaiseksi, joka vallitsi ennen tekoa t?

Silloin nuolien seuraaminen takaperin voidaan tulkita etsinnäksi eteenpäin käyttäen vastatekoja t^{-1}.

Muuten voi olla vaikea määritellä miten nuolien seuraaminen takaperin oikein muuttaakaan etsinnän tilaa.

Esimerkiksi kuvan 15 kaltaisissa reititysongelmissa teko on peruutettavissa, ellei se kulje yksisuuntaista tietä.

- Entä jos maalitiloja on monta?

  Jos ne on annettu *eksplisiittisenä* äärellisenä listana, niin voidaan

  – etsiä taaksepäin ikään kuin se ongelma olisi monitilainen, tai

  – aloittaa yksi taaksepäin etsintä kustakin maalitilasta.

  Mutta jos onkin annettu *implisiittinen* maaliehdon kuvaus, niin pitäisi jotenkin järkeillä millaisista etsinnän tiloista saattaa seurata jokin maalitila. Tämä voi olla vaikeaa.

- Miten havaitaan, että etsinnät ovat kohdanneet?

  Aikaraja $O(b^{d/2})$ edellytti, että kohtaaminen voidaan tunnistaa vakioajassa. Tähän pyritään käytännössä usein *hajautustaululla (hash table)*.

- On valittava (ei välttämättä yksi ja sama) etsintäalgoritmi kummallekin suunnalle.

  Jos ainakin toinen näistä on luvun 3.1.1 leveyssuuntainen etsintä, niin kohtaamisen havaitseminen helpottuu.

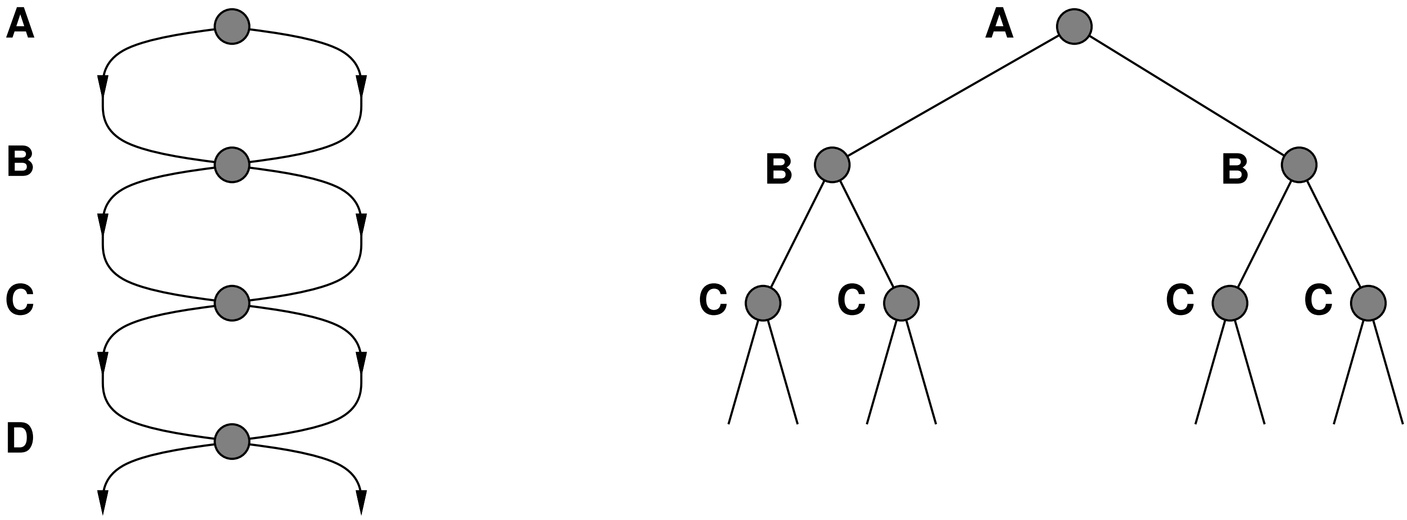
Verrataan lopuksi esiteltyjen algoritmien ”strategisia mittoja”:

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Kuva 27: Systemaattisten etsintöjen vertailu [RN95, kuva 3.18].

3.1.7. Toistuvien tilojen eliminoimisesta

Etsinnän kuluessa saatamme löytää toistamiseen saman tilan, emmekä tietenkään halua tuhlata resursseja käsittelemällä sitä — ja etenkään siitä alkavaa koko alipuuta — uudelleen:



Kuva 28: Eksponentiaalinen eliminointi [RN95, kuva 3.19].

On ongelmia, joiden huolellinen määrittely ehkäisee tilojen toistumisen jo ennalta.

Toisaalta vaikkapa peruutettaville teoille näin ei ole.

Etsintäalgoritmia voidaan muuttaa siten, ettei se enää laajennakaan mitään sellaista etsinnän tilaa, joka on jo esiintynyt. . .

1. . . . sillä polulla, joka vie etsintäpuun nyt laajennettavasta lehtisolmusta sen juureen.

   Tämä polku on silloin pidettävä muistissa.

   Jokaista tilaa laajennettaessa sen polku on ensin käytävä läpi.

   Jos algoritmina on variantti luvun 3.1.3 syvyyssuuntaisesta etsinnästä, niin tämä polku löytyy pinosta.
2. . . . milloin tahansa etsinnän kuluessa.

   Muistintarve on O(avaruuden koko).

Hakurakenteena käytetään usein hajautusta.

Silti tämä vaihtoehto saattaa olla raskas.

Se on kuitenkin aggressiivisempi toiston torjuja kuin tapa 1.

Toisaalta eri poluilla olevien samojen tilojen rooli voi riippua myös polkuhinnasta: Jos kuvan 15 autoilutehtävässä minimoidaan vain kuljettujen teiden lukumäärää, niin reitit

(a) Arad – Sibiu – Rimnicu Vilcea – Craiova – Dobreta ja
(b) Arad – Timisoara – Lugoj – Mehadia – Dobreta

ovat "samat": molemmissa päädytään samaan kaupunkiin 4 askeleessa.

Jos taas minimoidaan kokonaismatkaa, niin reitti 2a on 486 km pitkä, mutta reitti 2b vain 374 km. Oletetaan, että reitti 2a löydetään ensin, ja sen jälkeen reitti 2b jätetään huomiotta "samana": saatamme päätyä väärään vastaukseen!

Silloin "samuus" on määriteltävä siten, että se ottaa huomioon myös kertyneen polkuhinnan.

Tällöin vähenee poistettavissa olevan toiston määrä.

Kokonaiskustannukset ja -hyödyt riippuvat tietenkin siitä, kuinka paljon toistoa etsintäongelmassamme esiintyy:

- Kuvan 20 mukaisessa 8-liukupelissä on $d \approx 20$ ja $b \approx 3$.
- Sen etsintäpuussa on siis noin 5.2×10^9 solmua yhtälöstä (7).
- Niistä vain $9! = 362\ 880$ on erilaisia.
  Polkuhinnoista (siirtomääristä) ei tarvinne pitää huolta.
  Näin pieni määrä mahtuu varmaankin jo keskusmuistiin.
  (Toisaalta virtuaalimuistiin joutuminen hidastaa.)
- Siis toistuvien tilojen poisto hävittää 99.99% etsintäpuusta!
  Eli kannattanee tässä tapauksessa.

3.2. Informoituja etsintämenetelmiä [RN95, luku 4] [Nil98, luku 9]

Luvun 3.1 algoritmit ovat usein tehottomia, koska ne eivät ota huomioon ratkaistavan etsintäongelman erityispiirteitä. Parempi tulos saadaan algoritmeilla, joita suunnittelija voi "ohjata" kohti tavoitetta jollakin *ongelmakohtaisella lisäinformaatiolla*.

Luvun 3.1.2 algoritmi käytti etsintäongelman määrittelyyn kuuluvaa polkuhintafunktiota g. Se valitsi aina laajennettavaksi sellaisen etsintäpuun solmun n, jossa "tähän mennessä toimijalle suunniteltu työmäärä"

$$g(n) = \text{juuresta solmuun } n \text{ vievän polun hinta}$$

on pienin. Lisäinformaatiota tuo puolestaan funktio

$$h^*(n) = \text{solmusta } n \text{ lähimpään ratkaisuun vievän polun hinta}$$

joka antaa "tästä eteenpäin jäljellä olevan työmäärän".

Tällaisten funktioiden ohjaama etsintämenetelmä on **"paras ensin" -etsintä (best-first search)**:

function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

Algoritmi 4: Paras ensin -etsintä [RN95, kuva 4.1].

Toteutus on kuten luvussa 3.1.2, paitsi että keko järjestetäänkin nyt ohjausfunktion EVAL-FN solmuille antamien arvojen mukaan.

Oikean funktion h^* *tarkkoja* arvoja ei yleensä ole tarjolla. Siksi käytetäänkin jotenkin etsintäongelmasta johdettua *heuristista arviofunktiota* h.

Sanalla "heuristiikka" (kr. *heurisko*, "keksiä"; vrt. "Heureka!") on useita eri merkityksiä:

- Tässä yhteydessä se tarkoittaa algoritmista tekniikkaa, joka saattaa *parantaa algoritmin suorituskykyä*.
- Se voi tarkoittaa myös sellaista yksinkertaistavaa oletusta, jonka avulla jokin vaikea *algoritminen ongelma helpottuu* — ratkeamattomasta ratkeavaksi, NP-täydellisestä polynomiseksi,. . .

  Esimerkiksi tyydytään tarkastelemaan vaikean yleisen tapauksen sijasta sellaisia yksinkertaisempia tapauksia jotka kattavat kaikki "käytännössä vastaan tulevat" tilanteet. . .
- Joskus se tarkoittaa jopa *algoritmisen ongelmanratkaisun vastakohtaa*, "luovaa inhimillistä" säännöistä vapaata ongelmanratkaisua!

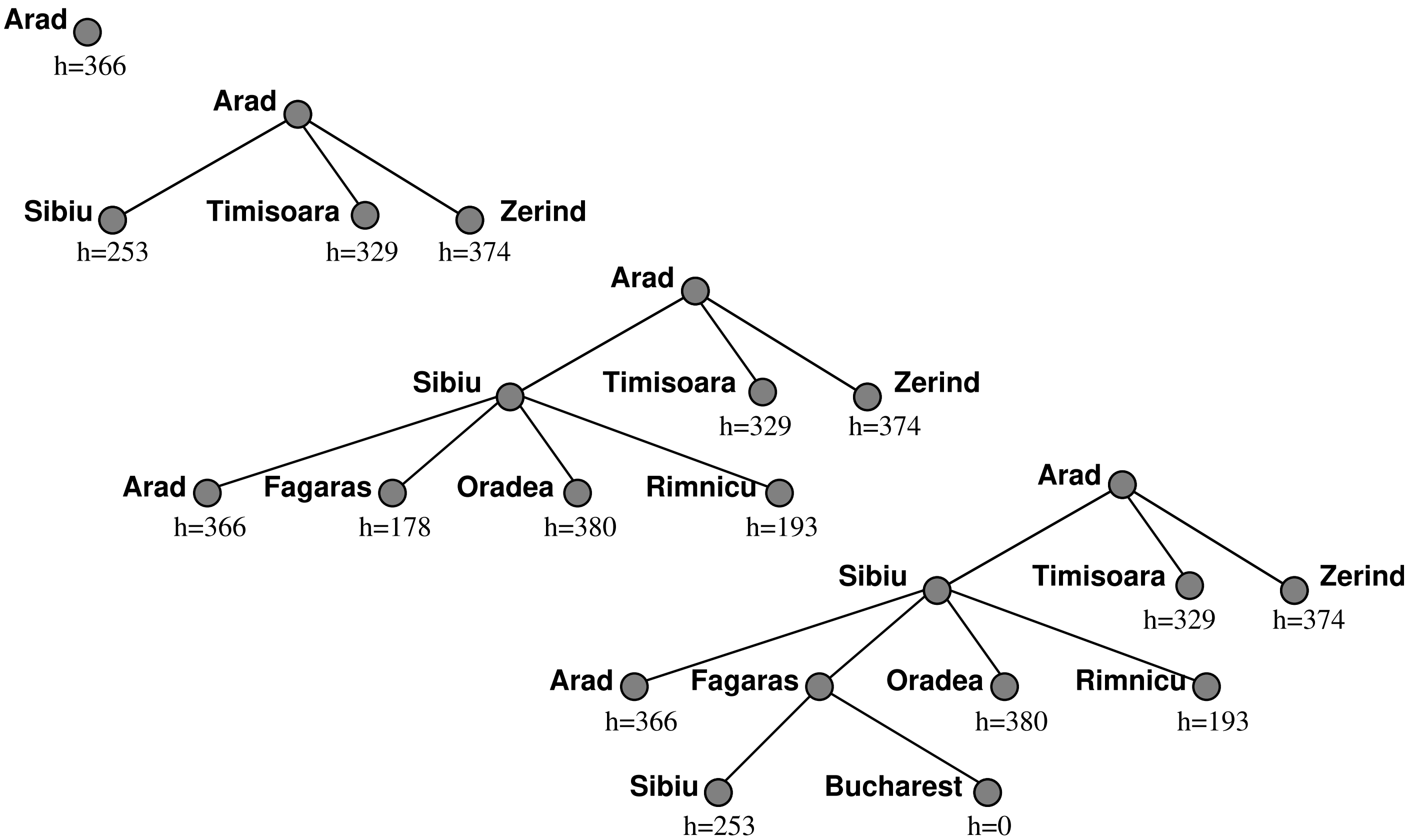
Yleisen systemaattisen etsintälgoritmin 2 tavoin yleisestä informoidusta etsintäalgoritmista 4 saadaan yksittäisiä algoritmeja kiinnittämällä laajennusjärjestyksen määräävä funktio $\textsc{Eval-Fn}$ eri tavoin.

Esimerkiksi luvun 3.1.2 algoritmi (joka ei vielä ollut informoitu eli ei käyttänyt funktiota h) saadaan valitsemalla $\textsc{Eval-Fn} = g$, eli ongelman polkuhinta juuresta nykyiseen lehtisolmuun.

Esimerkkinä informoidusta etsinnästä käytetään kuvan 15 tehtävää löytää reitti Aradista Bukarestiin. Lisäinformaatioksi h on kuvan oikeassa laidassa listattu kuinka kaukana Bukarest on kustakin kaupungista maantieteellisesti eli "linnuntietä". (Itse reitti kiemurtelee taas tarjolla olevia maanteitä.)

3.2.1. Ahne etsintä (Greedy Search)

Valitaan ensin Algoritmissa 4 $\textsc{Eval-Fn} = h$. Saatu algoritmi valitsee laajennettavaksi sen etsintäpuun lehden, joka kulloinkin *näyttää olevan lähimpänä tavoitetta*:



Kuva 29: Ahneesti Aradista Bukarestiin [RN95, kuva 4.3].

Tämä algoritmi on luvun 3.1.2 algoritmin "peilikuva":

- siellä pysyteltiin mahdollisimman lähellä lähtösolmua
- täällä rynnätään mahdollisimman lähelle maalia.

Ahne algoritmi käyttäytyy *usein muttei aina* ikään kuin luvun 3.1.3 syvyyssuuntainen etsintä, johon on lisätty "suuntavaistoa":

- Algoritmi pyrkii jatkamaan samaa polkua yhä eteenpäin ja eteenpäin, mutta nyt polkua ei jatketakaan "ensimmäisestä" lapsisolmusta vaan siitä — ei välttämättä (mutta usein) lapsi- — solmusta jonka arvioitu etäisyys tavoitteesta on pienin.

- Algoritmi muuttaa kulkuaan vasta kun tämä polku päätyy umpikujaan tai alkaa etääntyä tavoitteestaan. Mutta silloinkin peruutetaan yleensä samalla polulla taaksepäin ja kokeillaan jotakin melkein yhtä lupaavaa vielä tutkimatonta vaihtoehtoa.
- Kuvasta 29 nähdään, että ahneus *ei ole optimaalista*: löydetty reitti on 32 km pidempi kuin Rimnicu Vilcea – Pitesti -vaihtoehto. Tämä paras vaihtoehto jäi huomaamatta, koska Fagaras näytti *paikallisesti lupaavammalta*.
- Ahneus *ei ole myöskään täydellistä*: jos kuvan 15 ongelmana olisikin päästä Iasista Fagarasiin, niin ahne algoritmi (ilman luvun 3.1.7 toistojen poistoa) jää heilahtelemaan välille Iasi – Neamt.
- Algoritmien pahimman tapauksen aikavaatimukset ovat samat, joskin "hyvällä" h ahneesta algoritmista voi tulla hyvin nopea.
- Toisin kuin luvun 3.1.3 syvyyssuuntainen etsintä, ahne algoritmi joutuu pitämään kaikki etsintäpuun solmut muistissa.

  Etsintä saattaa näet palata myöhemmin kokeilemaan jotakin kesken jäänyttä solmua sitten kun kaikki sitä lupaavammat on käsitelty.

3.2.2. A*-etsintä

Luvun 3.2.1 ahne etsintä on nopea muttei täydellinen eikä optimaalinen. Luvun 3.1.2 algoritmi oli taas täydellinen ja optimaalinen muttei nopea. Yhdistetään ne (muista luku 3.2) valitsemalla Algoritmissa 4 $\textsc{Eval-Fn} = f$, missä

$$f(n) = g(n) + h(n) \tag{9}$$

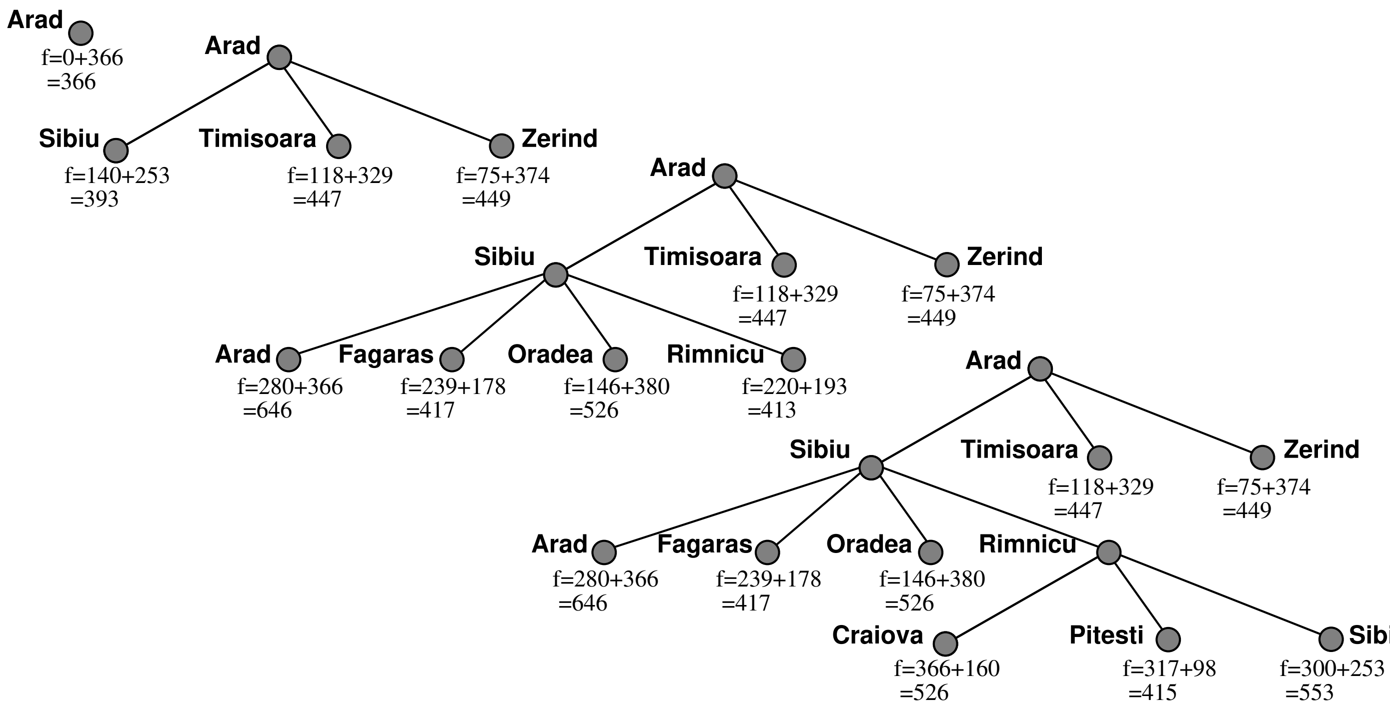
arvioi *halvimman sellaisen ratkaisun hintaa, joka kulkee solmun n kautta*.

Vaaditaan lisäksi että heuristiikka h on *luvallinen (admissible)*, eli että

$$h(n) \leq h^*(n)$$

jokaiselle solmuille n. Toisin sanoen,että h on *aina optimistinen*: vielä jäljellä oleva työmäärä näyttää aina pienemmältä kuin se oikeasti onkaan.

Silloin myöskään f ei koskaan yliarvioi pessimistisesti solmun n kautta kulkevien ratkaisujen hintoja. Kuvassa 15 luvallinen h on "etäisyys linnuntietä Bukarestiin":

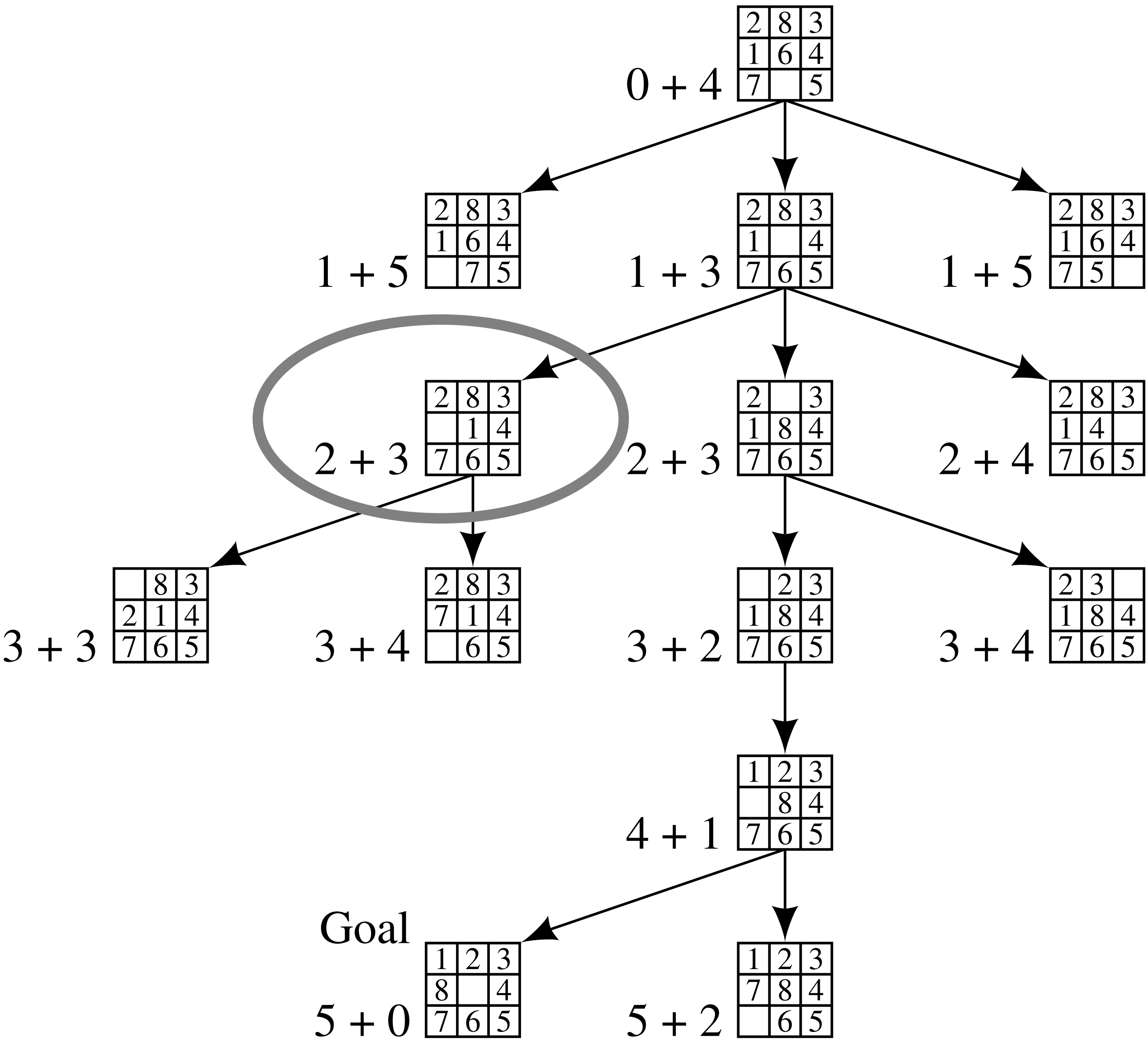


Kuva 30: Parhaiten Aradista Bukarestiin [RN95, kuva 4.4].

Tai jos 8-liukupelissä

$$\hat{h}(n) = \text{laudalla } n \text{ väärissä paikoissa olevien tiilten määrä}$$

niin etsintä onnistuu nopeasti (lukuunottamatta rengastettua haaraa):



Kuva 31: Heuristinen etsintä 8-liukupelissä [Nil98, kuva 9.2].

- g on *leveys*suuntainen osa joka pyrkii *huolellisuuteen*.
- h on *syvyys*suuntainen osa joka pyrkii *tehokkuuteen*.

A*-algoritmi on. . .

optimaalinen: Kuvan 30 juuresta alkavilla poluilla funktion f antamat arvot *eivät vähene*.

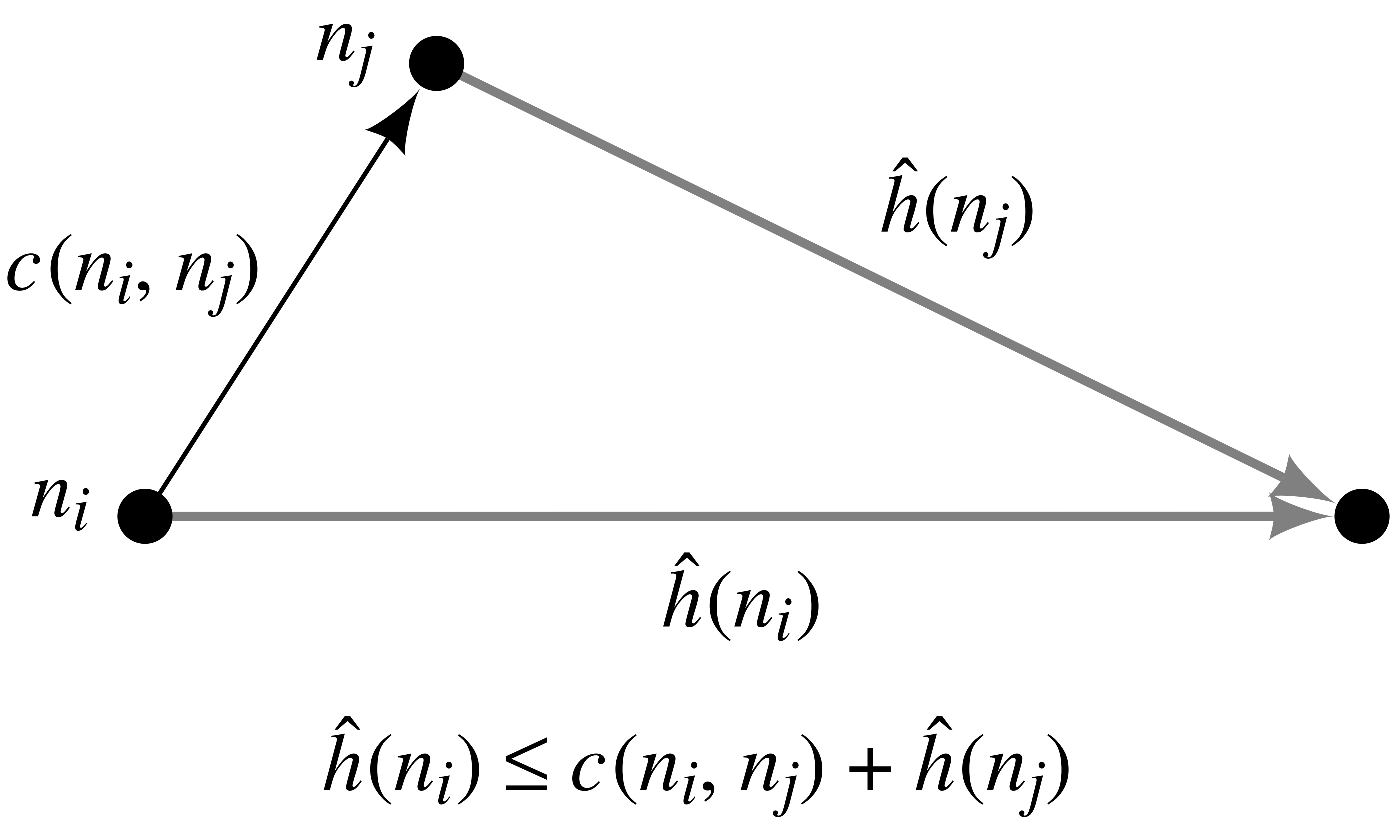
Tällaista f kutsutaan *monotoniseksi*, tarkemmin sanoen *kasvavaksi* [Myr74a, määritelmä 2.11.1].

Koska polkuhinta g oletettiin kasvavaksi ja '+' säilyttää kasvavuuden, riittää tarkastella heuristiikkaa h.

Funktio f on monotoninen jos ja vain jos h toteuttaa *kolmio-epäyhtälön*, joka saa nyt muodon

$$h(n) \leq k(n, n') + h(n')$$

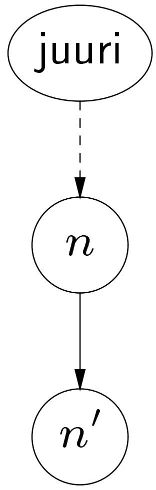
missä $k(n, n')$ on pienin polkuhinta solmusta n solmuun n' (todistus sivuutetaan).



Kuva 32: Kolmioepäyhtälö [Nil98, kuva 9.7].

Esimerkiksi kuvan 15 ”linnuntie-etäisyys” täyttää kolmioepäyhtälön, onhan se geometrinen etäisyysmitta.

Jos f ei olekaan monotoninen,niin etsintäpuusta löytyy haara



jossa $f(n) > f(n')$. Mutta $f(n)$ oli optimistinen arvio *kaikille* juuresta solmun n kautta maaliin kulkeville poluille — myös niille jotka kulkevat kaaren isäsolmusta n lapsisolmuun n' kautta, mutta näitähän $f(n')$ pitää vielä parempina!

Monotonisuus voidaan siis palauttaa yhtälöön (9) *pathmax-yhtälöllä*

$$f(n') = \max(f(n), g(n') + h(n')) \tag{10}$$

jonka jälkeen f on varmasti monotoninen, kunhan h on luvallinen.

Oletetaan siis f monotoniseksi. Silloin optimaalisuus voidaan näyttää seuraavasti:

Algoritmin voidaan nyt ajatella piirtävän sisäkkäisiä *korkeuskäyriä* etsintäavaruuteen (kuva 33).

Luvun 3.1.2 etsintäalgoritmissa (jossa $h = 0$) nämä käyrät ovat renkaita alkutilan ympärillä, kun taas paremmat heuristiikat ”venyttävät” niitä soikioiksi kohden maalitilaa.

Olkoon f^* parhaan ratkaisun polkuhinta. Silloin algoritmi laajentaa

- ensin kaikki solmut n käyrän f^* sisällä (eli $f(n) < f^*$) funktion f arvojen mukaan kasvavassa järjestyksessä, ja
- sitten osan solmuista itse käyrällä f^* (eli $f(n) = f^*$),

kunnes itse käyrältä f^* löytyy (jokin) maalisolmu.

Formaalisti: olkoon G optimaalinen maalisolmu, ja olkoon G_2 jokin toinen ei-optimaalinen maalisolmu, jolla siis

$$f(G_2) > f^*. \tag{11}$$

Voisiko algoritmi vastata G_2? Silloinhan se *ei* olisikaan optimaalinen! Olkoon n juuresta solmuun G vievän optimaalisen polun P tämänhetkinen lehti; jos sellaista ei ole, niin P on jo käsitelty ja G vastattu. (Kuva 34 näyttää tilanteen.)

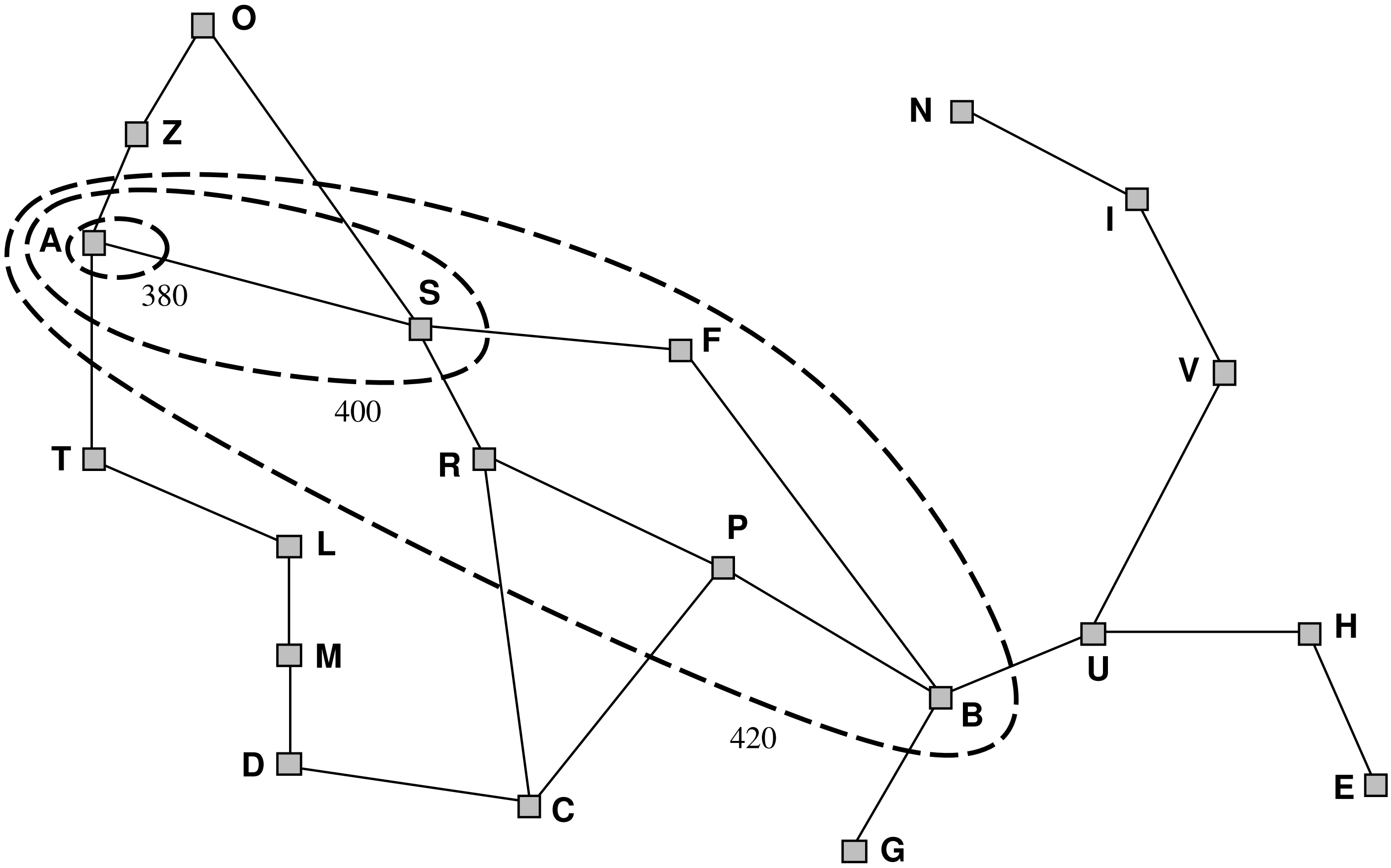
$$\begin{aligned} f^* &= f(G) \text{ koska } G \text{ on optimaalinen.} \\ &\geq f(n) \text{ koska } f \text{ on monotoninen.} \\ &\geq f(G_2) \text{ koska } G_2 \text{ valittiin ennen kuin } n. \\ &= g(G_2) + h(G_2) \text{ yhtälöstä (9).} \\ &= g(G_2) \text{ koska } h \text{ on luvallisena } 0 \text{ kaikille maaleille.} \end{aligned}$$

Tämä on kuitenkin ristiriidassa yhtälön (11) kanssa, joten algoritmi on optimaalinen.

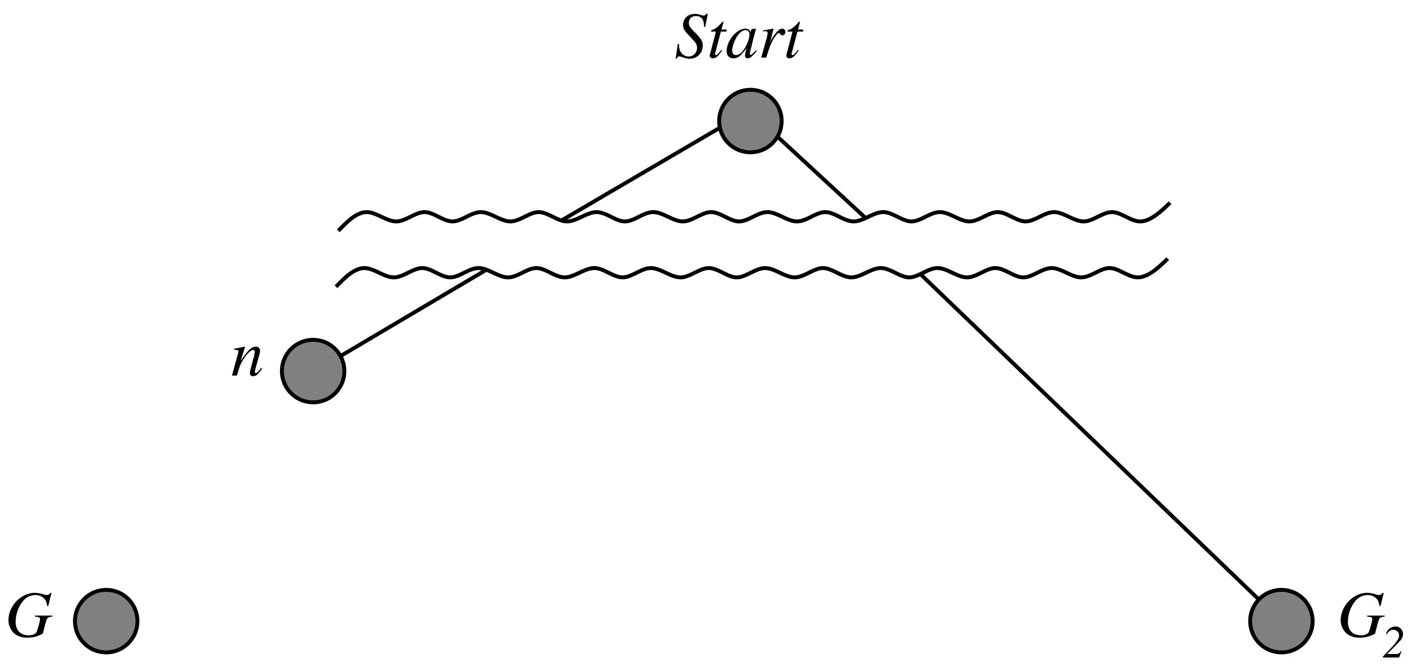
täydellinen: laajentamisjärjestyksensä vuoksi algoritmi löytää maalisolmun *ellei* käyrälle f^* tai sen sisälle jää äärettömän monta solmua, jotka algoritmi siis yrittäisi käsitellä ensin. Tämä on puolestaan mahdollista vain jos etsintäavaruudessa on. . .

- solmu, jolla on ääretön haarautumisaste.
  Tällöin etsintää tekevällä toimijalla olisi äärettömän monta eri vaihtoehtoa vastaavassa tilanteessa.
- ääretön polku $P = n_0 \xrightarrow{c_1} n_1 \xrightarrow{c_2} n_2 \xrightarrow{c_3} n_3 \cdots$ jonka hinta $g(P) = c_1 + c_2 + c_3 + \cdots \leq f^*$ pysyy äärellisenä.
  Tällöin etsintää tekevällä toimijalla olisi tarjolla mielivaltaisen "halpoja" tekoja.
  Tästä ehdosta mainittiin jo luvussa 3.1.4.

Algoritmi on siis täydellinen sellaisissa avaruuksissa joissa *haarautumisaste on äärellinen* ja *kaikkien tekojen kustannuksilla on jokin alaraja* $\delta > 0$.



Kuva 33: Korkeuskäyrät Aradista Bukarestiin [RN95, kuva 4.5].



Kuva 34: Algoritmin **A*** optimaalisuus [RN95, kuva 4.6].

tehokas: algoritmi on *optimaalisen tehokas (optimally efficient)* seuraavassa mielessä:

Olkoon **B*** minkä tahansa kilpaileva etsintäalgoritmi, joka. . .

- käyttää samaa heuristista ohjausinformaatiota h kuin **A***,
- on optimaalinen kuten **A***, ja
- etenee samoin kuten **A*** laajentamalla polkuja juuresta alkaen.

Silloin on väistämättä olemassa sellaisia etsintäongelmia, joissa **B*** *joutuu laajentamaan ainakin yhtä monta solmua kuin* **A***:

Muuten **B*** jättäisi laajentamatta solmuja käyrältä f^* tai sen sisältä, ja saattaisi näin jättää optimipolun laajentamatta. (Todistus sivuutetaan.)

Toisaalta käyrälle f^* ja sen sisälle jää yleensä eksponentiaalisen monta solmua verrattuna optimipolun solmujen määrään, joten **A*** on huolellisuutensa vuoksi yleensä *eksponentiaalinen*.

Tämä eksponentiaalisuus voidaan varmasti välttää vain jos

$$\left|h(n) - h^*(n)\right| \leq O(\log h^*(n))$$

eli jos heuristiikka h on niin tarkka, että sen tuottama *virhe on aina korkeintaan logaritminen*. (Todistus sivuutetaan.)

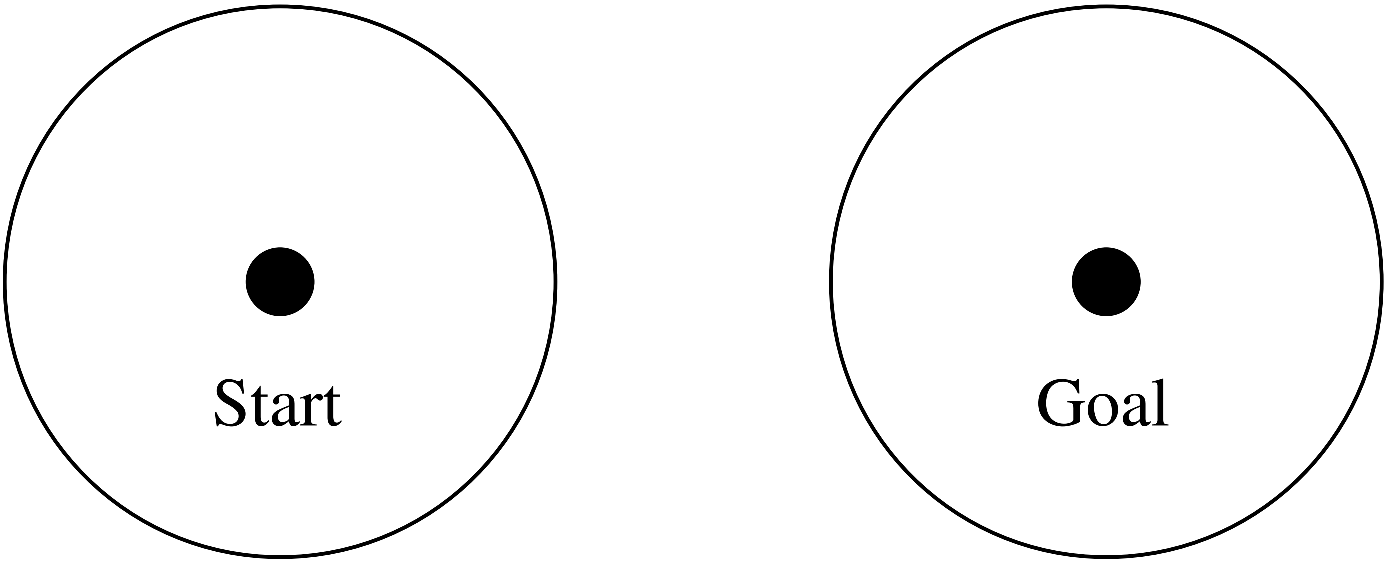
Käytännössä virhe tapaa olla ainakin luokkaa $g(n)$.

Algoritmin **A*** suurempi ongelma on kuitenkin (jälleen) *muistintarve*, koska se pitää kaikki (lehti)solmut muistissa. Tähän ongelmaan palataan luvussa 3.2.4.

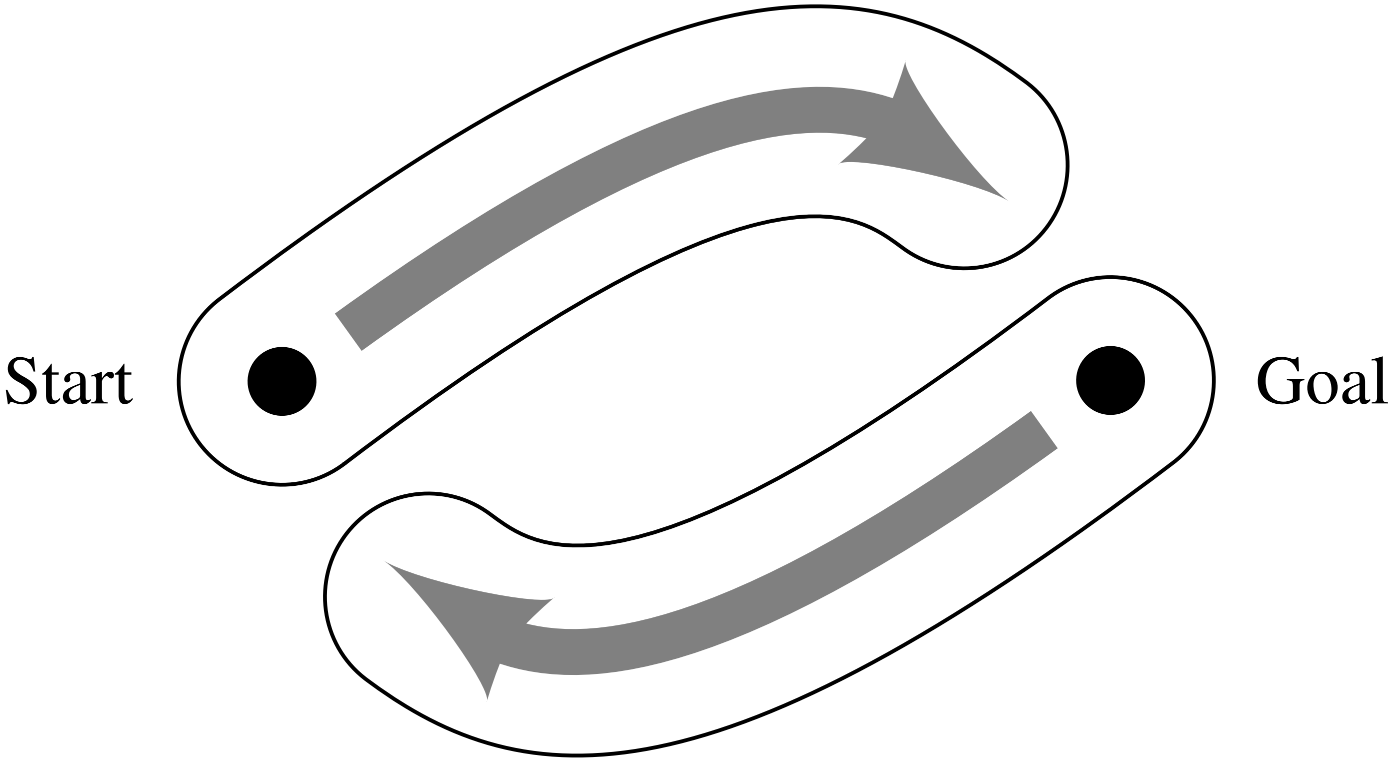
Jos heuristista etsintää (esimerkiksi **A***) käyttää luvun 3.1.6 kaksisuuntaisessa etsinnässä, voivat eri suuntiin etenevät, yksinään hyvät heuristiikat kumota toisensa [Nil98, kuva 9.10] (sivulla 86):

Kumpikin etenee omaan hyvään suuntaansa, ja ohittaa toisen. Etsintätyö ei putoakaan murto-osaan, vaan kaksinkertaistuu, koska kumpikin suunta päätyy tekemään työn kokonaan yksin, eikä puoliksi vastasuunnan kanssa.

Tästä syystä suositellaankin, että toinen kaksisuuntaisista etsintäalgoritmeista olisikin leveyshaku eikä heuristiikka.



(a) Breadth-first search



(b) Heuristic search

Entä jos heuristiikkamme h *ei* olekaan luvallinen?

- Silloin $h(n)$ voikin arvioida solmusta n lähimpään ratkaisuun vievää hintaa $h^*(n)$ *pessimistisesti* ylöspäin (eli $h(n) > h^*(n)$).
- Silloin saattaakin olla olemassa jokin toinen solmu n' jolla ylöspäin pyöristystä ei tapahdukaan (eli $h(n') \leq h^*(n)$).
- Silloin saattaakin olla

$$\underbrace{g(n') + h(n')}_{f(n')} < \underbrace{g(n) + h(n)}_{f(n)}$$

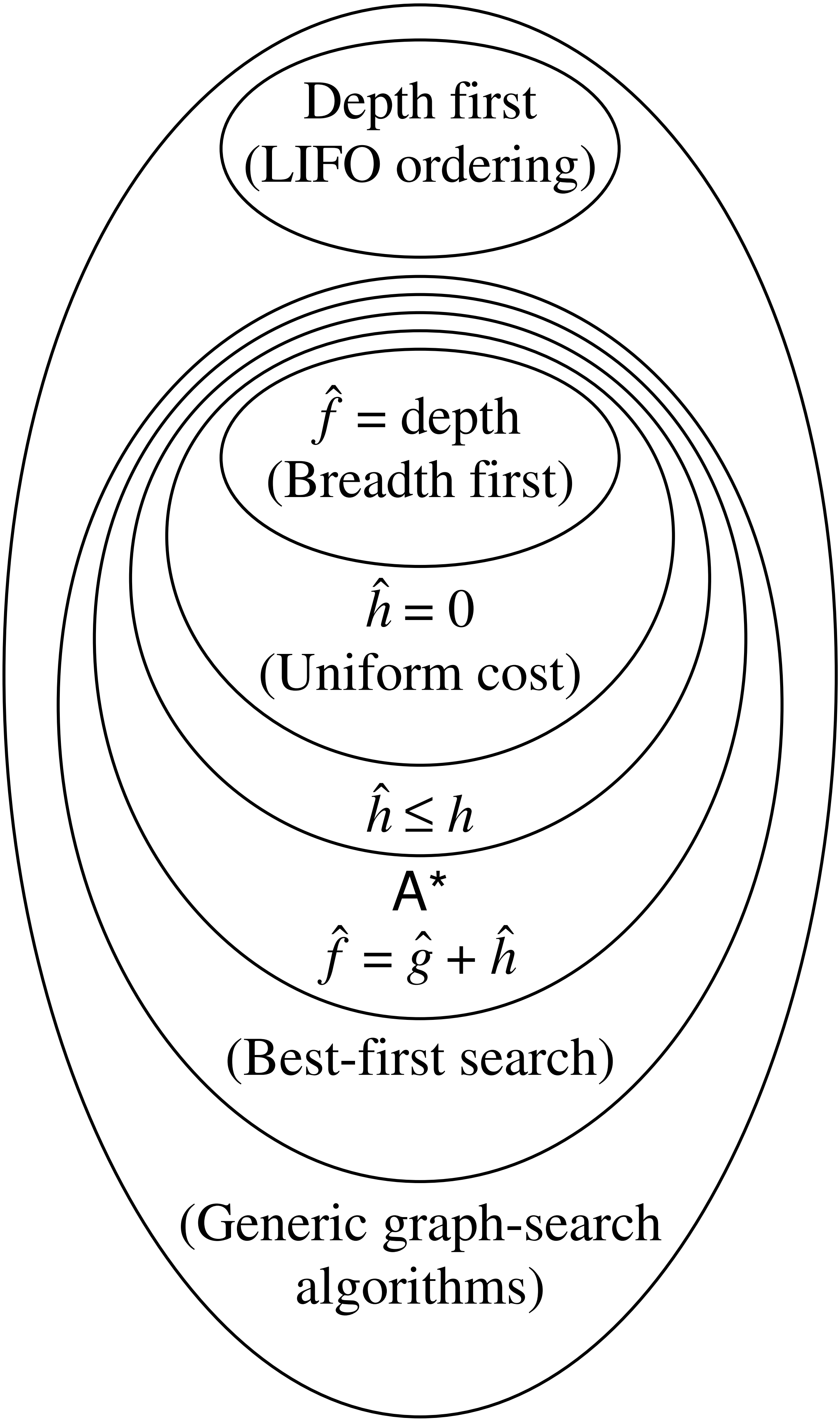
vaikka todellisille hinnoille

$$g(n') + h^*(n') > g(n) + h^*(n).$$

- Silloin etsintä valitseekin todellisuudessa huonomman solmun n' eikä parempaa solmua n.
- Siten etsintä voi päätyä tulokseen, joka ei olekaan halvin mahdollinen — enää etsintä *ei ole optimaalinen*.

Sivun 88 kuva [Nil98, kuva 9.6] kertaa eri etsintämenetelmien välisiä suhteita seuraavin merkinnöin:

[Nil98]	[RN95]
$\hat{h}$	h
h	h^*
$\hat{f}$	f
$\hat{g}$	g


Depth first
(LIFO ordering)
$\hat{f}$ = depth
(Breadth first)
$\hat{h} = 0$
(Uniform cost)
$\hat{h} \leq h$
A*
$\hat{f} = \hat{g} + \hat{h}$
(Best-first search)
(Generic graph-search
algorithms)

3.2.3. Miten tutkia heuristisia funktioita?

Tarkastellaan kuvan 20 8-liukupeliä. Luvallisia heuristiikkoja ovat esimerkiksi

$h_1(n) = \hat{h}(n)$ luvusta 3.2.2

$$h_2(n) = \sum_{1 \leq i \leq 8} \text{tiilen } i \textit{ Manhattan-etäisyys} \text{ oikeasta paikastaan}$$
$$= 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 \text{ kuvan 20 lähtötilassa.}$$

Eräs heuristiikan h hyvyyden mitta on sen aikaansaama *todellinen haarautumisaste (effective branching factor)* b^*: Lasketaan ensin

- montako solmua N algoritmi **A*** laajentaa heuristiikalla h ja
- näin löydetyn ratkaisun syvyys d.

Ratkaistaan sitten b^* yhtälöstä (7)

$$N = \frac{(b^*)^{d+1} - 1}{b^* - 1}.$$

Tulos kertoo

- millainen haarautumisaste (reaalilukuna) olisi saman kokoisella *täydellisellä* puulla, eli
- kuinka leveää luvun 3.1.1 *leveyssuuntaista* etsintää **A*** vastaa heuristiikalla h.

Tulos mittaa heuristiikan h laatua koska

- leveyshaku oli hidas mutta huolellinen menetelmä joka ei huomioinut mitään lisäinformaatiota, joten

- mittausarvo kertoo kuinka kapeaksi heuristiikka pystyi kaventamaan leveyshaun, eli
- kuinka monta sellaista vaihtoehtoa yhdessä etsintäaskeleessa on, joista pitäisi valita "umpimähkään".

  Pienempi mittausarvo on siis parempi heuristiikka!

Jos esimerkiksi

$$N = 52 \text{ ja}$$
$$d = 5 \text{ niin}$$
$$b^* \approx 1.91$$

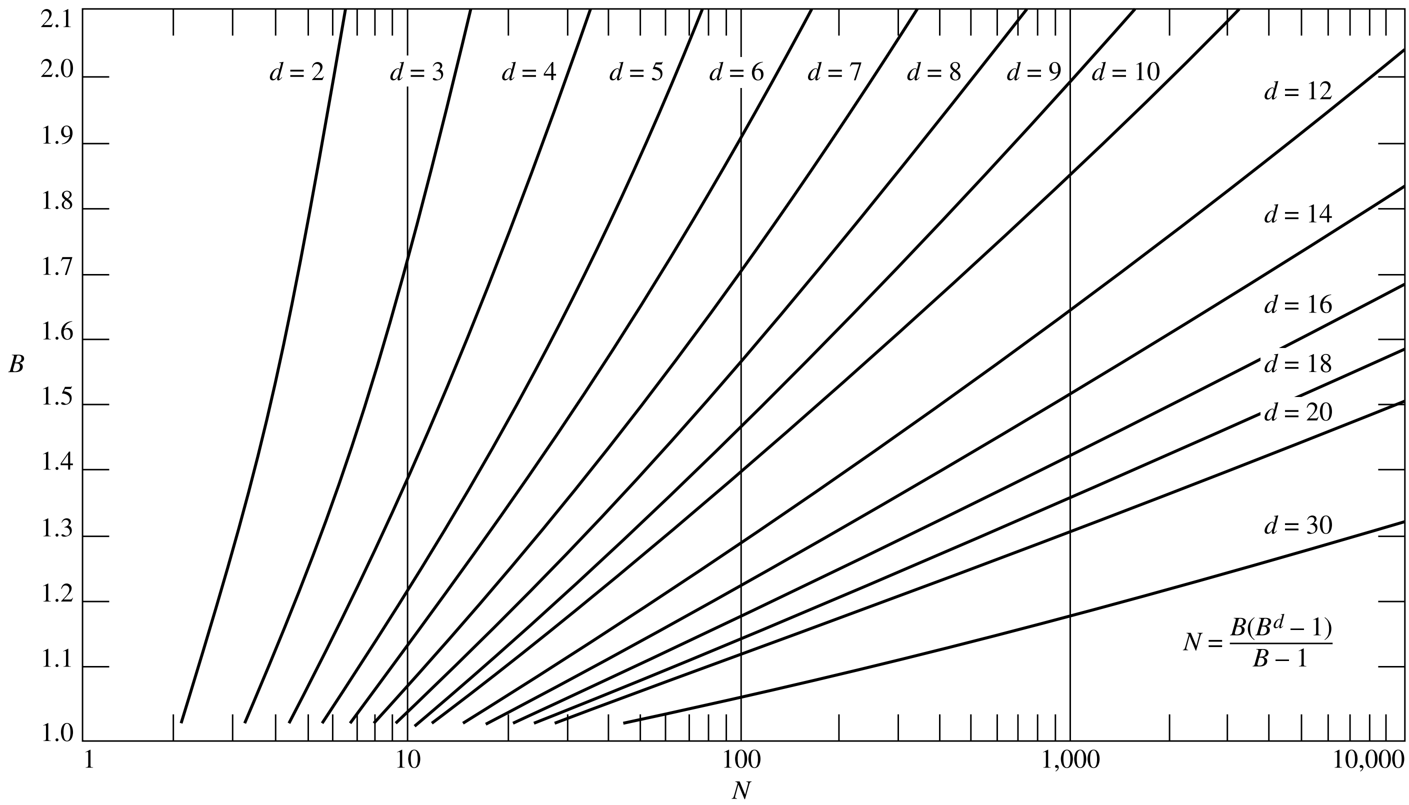
eli

- hakupuussa on $d = 5$ sukupolvea isäsolmuja, joista kullakin on (keskimäärin) $b^* = 1.91$ lapsisolmua, joten
- 9 etsintäaskeleen jälkeen on noin 9 tutkimatonta "yhtä hyvää" vaihtoehtoa, binääripuussa taas 10.

Taulukossa 2 verrataan mitalla b^* heuristiikkoja h_1 ja h_2 toisiinsa ja luvun 3.1.5 iteratiiviseen syventämiseen.

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Taulukko 2: Heuristiikkojen vertailua [RN95, kuva 4.8].



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.

$$\vdots$$

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.

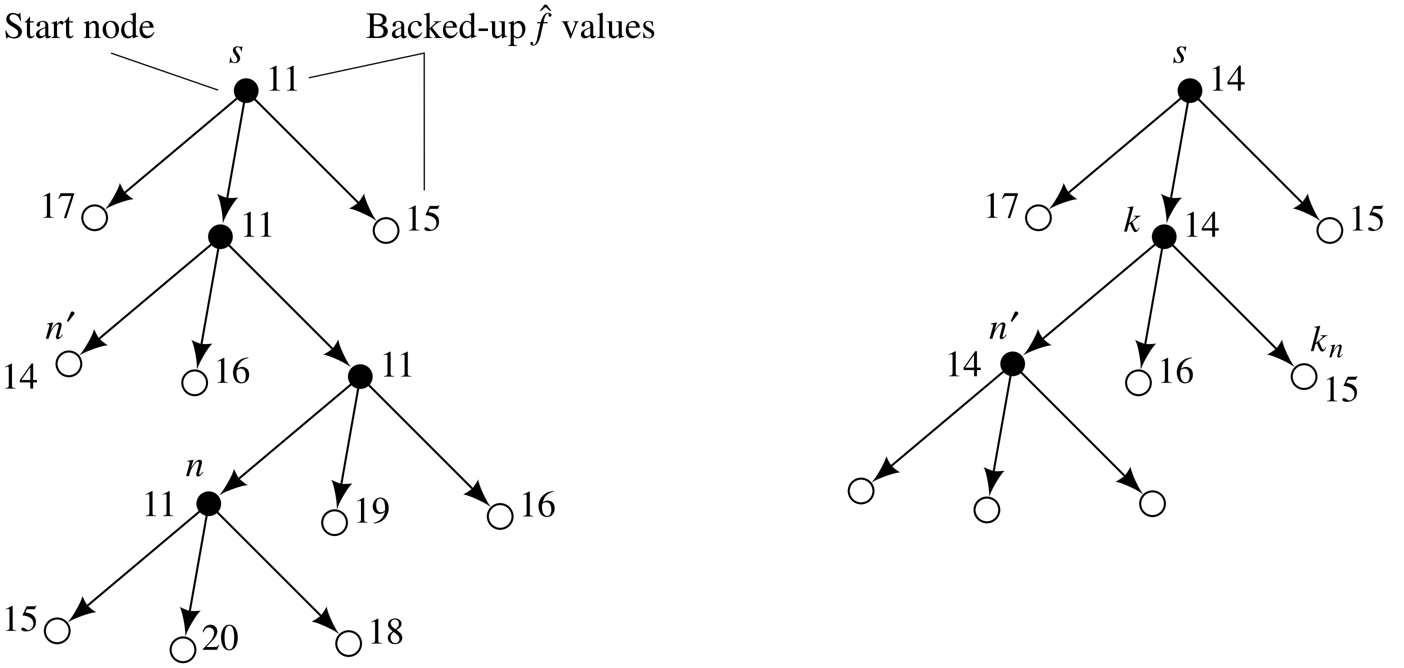
Algoritmin **RBFS** muistinsäästöstrategia on *pitää muistissa sisäsolmuista vain ne, joiden kautta on saatu parhaat uutiset*.

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 $f\text{-}limit$, 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].

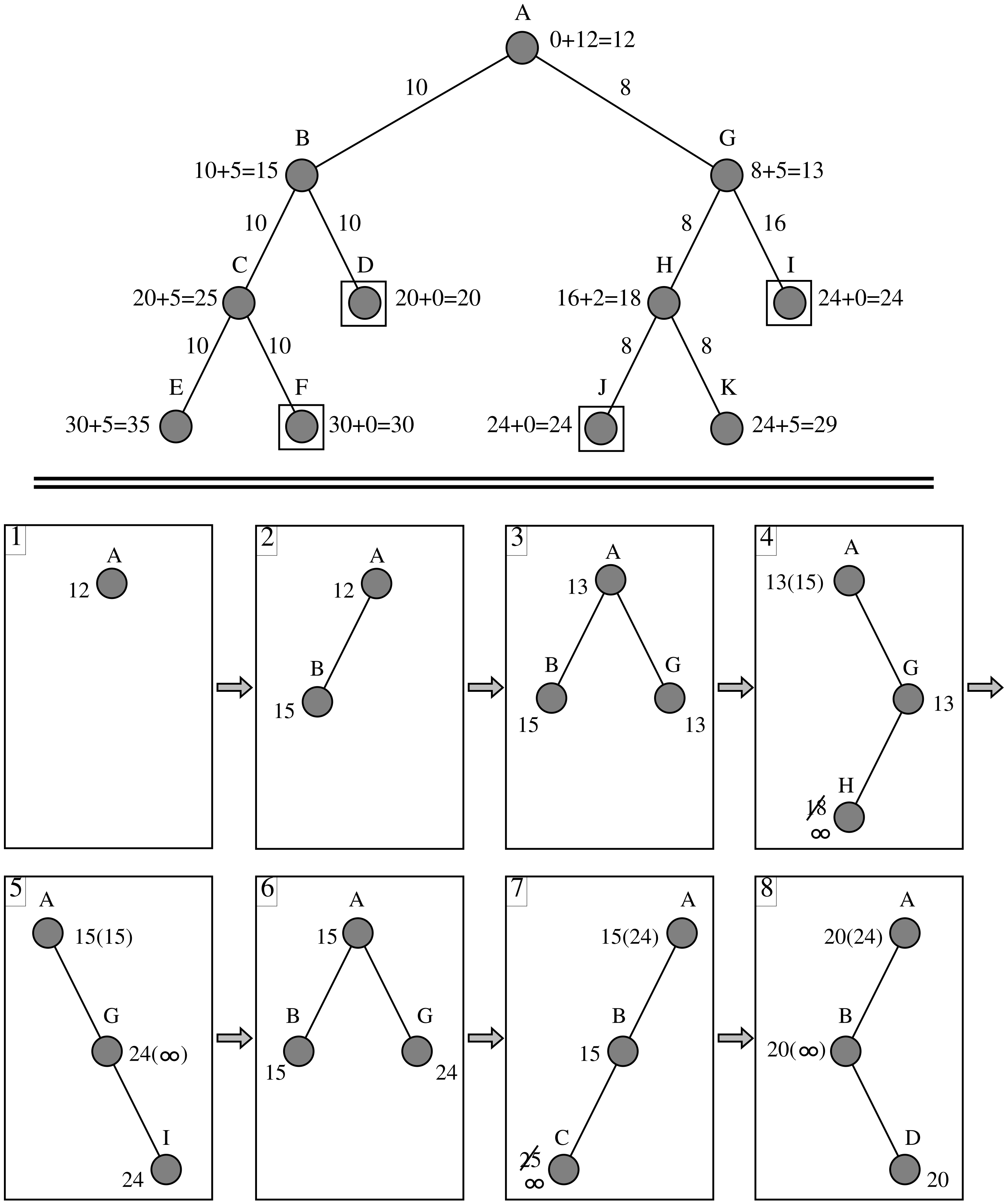
- **SMA*** päivittää solmujensa peruutusarvoja kuten **RBFS**.
  Merkitään solmun n nykyistä peruutusarvoa $f(n)$.
- **SMA*** pitää solmussa n kirjaa myös parhaasta luovutetun lapsen peruutusarvosta.

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 $f(H) = \not{1}8 = \infty$ 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. . .

4. Lisää ongelmanratkaisumenetelmiä

Jatketaan luvussa 3 esiteltyä etsintäteemaa sellaisissa ongelmissa joissa

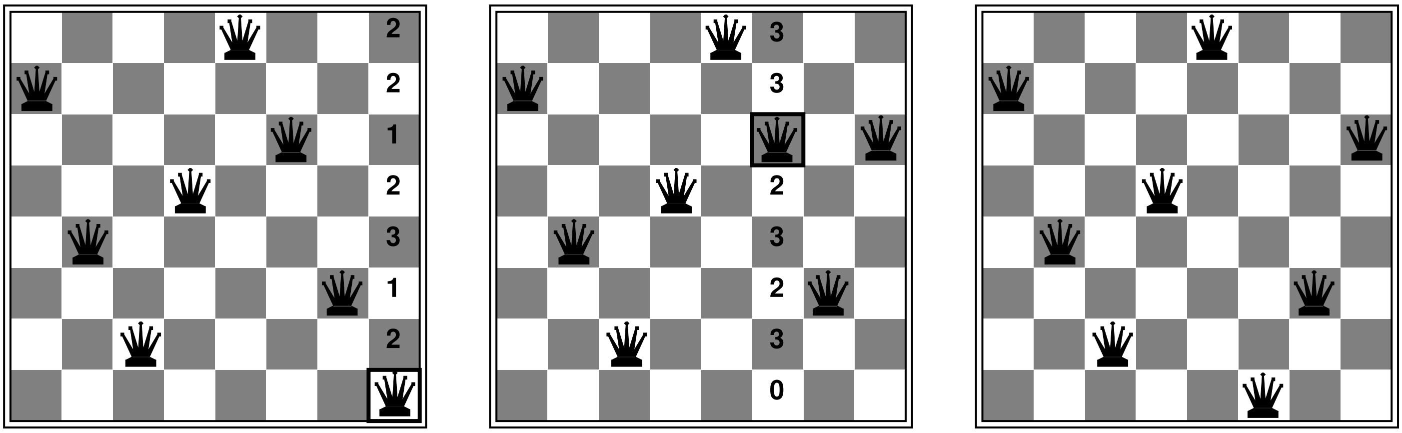
1. vain itse ratkaisu kiinnostaa, ei siihen johtanut tekojono
2. maaliehto ei enää olekaan "musta laatikko", vaan jakautuu erillisiin osaehtoihin, jotka voidaan hoitaa yksi kerrallaan
3. on mukana etsijän lisäksi pahantahtoinen vastustaja, kuten peleissä
4. on mukana etsijän vastustajien lisäksi välinpitämätön "luonto", kuten noppa onnenpeleissä.

Tällöin käytetään erilaisia etsintämenetelmiä kuin aiemmin:

1. Iteratiivisia parannusmenetelmiä.
2. Rajoitteiden levitystä.
3. Etsintää pelipuissa.
4. Etsintää todennäköisyyksin varustetuissa pelipuissa.

4.1. Iteratiiviset parannusmenetelmät [RN95, luku 4.4] [Nil98, luvut 11.1-2]

Esimerkiksi 8 kuningattaren (shakki)ongelmassa ei ratkaisun saavuttanutta tekopolkua kysytä; vain lopputuloksen laatu ratkaisee.



Kuva 38: 8 kuningattaren ratkaiseminen [RN95, kuva 4.16]. (Ruudun numero kertoo moniko sitä uhkaa.)

Tällöin on usein käytännöllisintä *tehdä alkutilaan toistuvasti pienia paikallisia korjauksia* ja toivoa, että näin päästäisiin ratkaisuksi kelpaavaan tilanteeseen.

Esimerkiksi VLSI-piirien optimoinnissa jo alkutila on kelvollinen piiri, jonka komponenttien paikkoja aletaan vaihdella, ja näin yritetään minimoida tarvittavien johtojen pituuksia.

Nämä menetelmät yrittävät *optimoida (minimoida/maksimoida) kohdefunktiota (evaluation function)*, joka mallintaa etäisyyttä ratkaisuun tai nykyisen ratkaisun hyvyyttä.

Menetelmien periaate selviää kuvasta 39: piirretään kohdefunktion $\textsc{Value}\colon \mathbb{R}^2 \mapsto \mathbb{R}$ kuvaaja, ja sijoitetaan nykyinen tila sinne.

Saadaan alppimaisemassa vaeltaja, joka yrittää nousta vuoren huipulle (tai minimointitehtävässä laskeutua laakson pohjalle), vaikka hän. . .

- on niin *likinäköinen*, että näkee vain yhden askeleen verran ympärilleen
- kärsii Korsakovin syndroomasta eli *ei pysty painamaan mitään asioita muistiinsa*.

Yksinkertainen strategia on katsella ympärilleen tuon yhden askeleen verran, ja ottaa niistä *se askel, joka vie korkeimmalle*.

Näin saadaan *vuoristokiipeilyalgoritmi 7 (Hill-Climbing Algorithm).* Se pysähtyy, kun ylöspäin ei enää päästä, ja ilmoittaa lötäneensä huipun.

Minimointitehtävissä tätä (jyrkimmin alaspäin menevää) menetelmää kutsutaan *gradienttilaskeutumiseksi (Gradient Descent)*.

Koulumatematiikassa funktion $f\colon \mathbb{R} \mapsto \mathbb{R}$ *derivaatta* f' (alias $\mathbf{D}f$, $f\frac{x}{dx}, \ldots$) pisteessä $x_0 \in \mathbb{R}$ oli geometrisesti:

- Piirrä funktion $y = f(x)$ kuvaaja κ 2-ulotteisessa (suorakulmaisessa) koordinaatistossa.
- Piirrä kuvaajalle κ se *tangenttisuora* τ joka kohtaa kuvaajan pisteessä x_0.
- Derivaatan arvo $f'(x_0)$ on *tangenttisuoran* τ *kulmakerroin*. . .
- eli *funktion* $f(x)$ *hetkellinen muutosnopeus kohdassa* x_0.
  Esimerkiksi fysiikassa (hetkellinen)
  - nopeus on paikan
  - kiihtyvyys on nopeuden

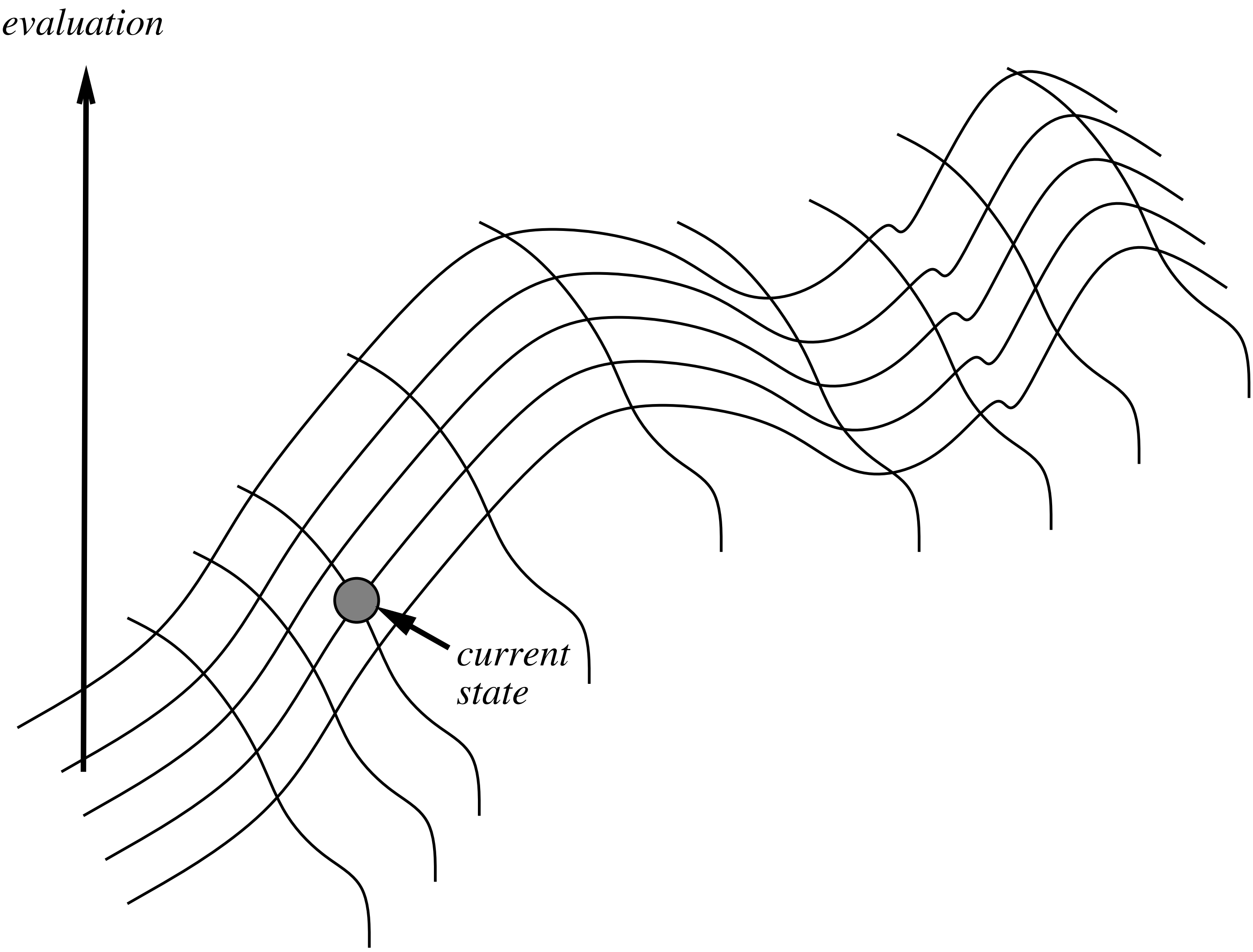
  derivaatta ajan suhteen.

Matematiikan kurssin "Differentiaali- ja integraalilaskenta II" gradientti on sama idea m-ulotteisten avaruuksien funktioille $g\colon \mathbb{R}^m \mapsto \mathbb{R}$.

3-ulotteisessa kuvassa 39 on maiseman $z = g(x, y)$ kuvaaja K.

Funktion arvon z hetkellinen muutosnopeus ja -suunta saadaan nyt siitä *tangenttitasosta* T, joka kuvaajalla K on nykyisessä pisteessä $\mathit{current\ state} = \langle x_0, y_0 \rangle$.

Maisemafunktion g gradientti on *tangenttitason* T *normaalivektori*.



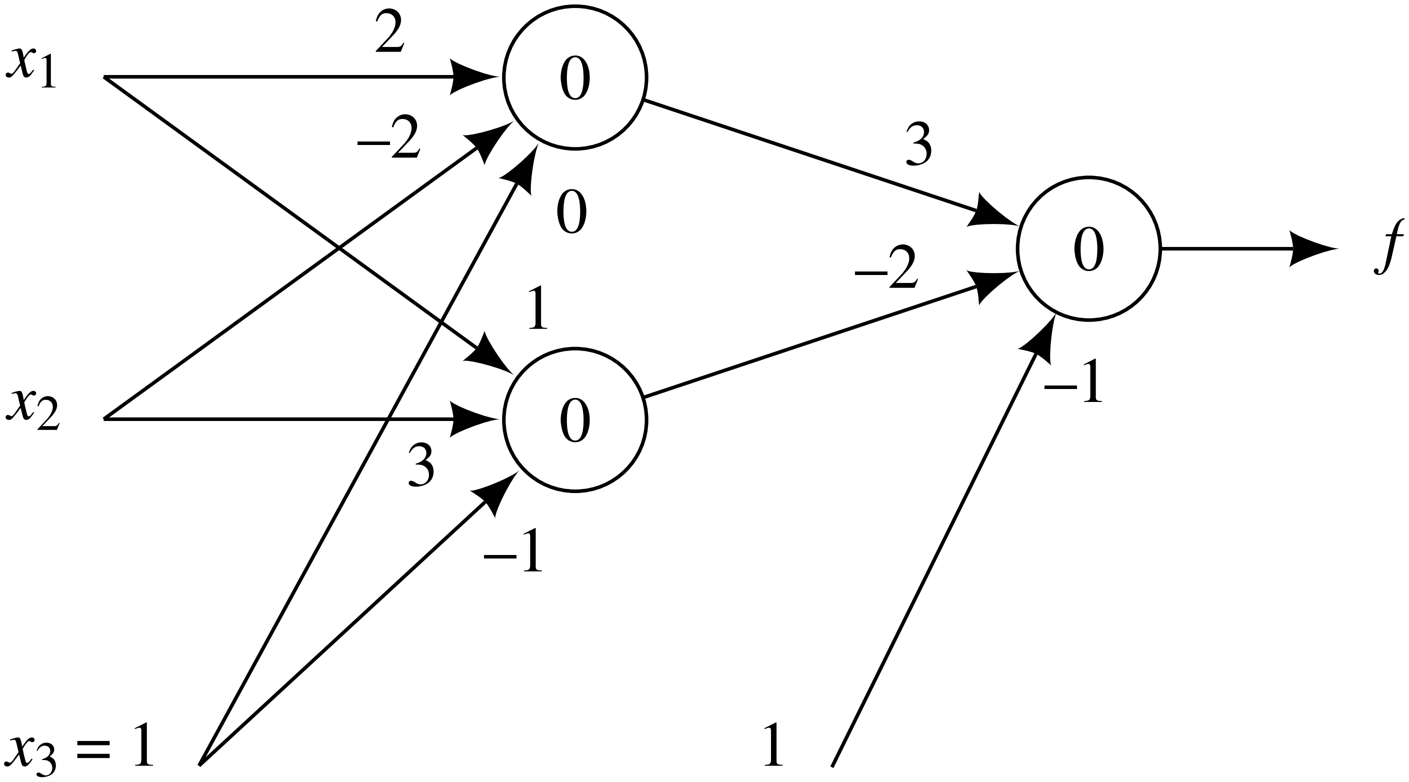
Kuva 39: Optimoitavan funktion maisemaa [RN95, kuva 4.13].

```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
       next ← a highest-valued successor of current
       if VALUE[next] < VALUE[current] then return current
       current ← next
   end
```

Algoritmi 7: Vuoristokiipeily [RN95, kuva 4.14].

Gradienttilaskeutumisella voi siis suorittaa etsintää *jatkuvissa* avaruuksissa kuten reaalilukuja sisältävissä ongelmissa. (Luvun 3 mentelmät olivat diskreettejä.)



Kuva 40: Pieni monitasoinen hermoverkko [Nil98, kuva 3.6].

Luvussa 1.5 mainittu monisoluisten hermoverkkojen oppimisalgoritmi on gradienttilaskeutumisen eräs sovellus:

- Perustehtävässä annetaan
  - hermoverkko H jonka kaarien (reaali)painoja voi muutella
  - esimerkkiaineisto A joka verkon H pitää oppia.
    Aineisto on valmis lista pareja $\langle$annettu syöte, haluttu tulos$\rangle$.
- Minimoitava funktio g mittaa sitä *virhettä*

$$|H(\text{annettu syöte}) - \text{haluttu tulos}|$$

jonka nykyinen H aineistosta poimitulla esimerkillä tekee.
Yleensä valitaan mitaksi virheen neliö.

- Maisema $\mathcal{M}$ koostuu kaikista mahdollisista H:
  Jokainen muuttuva kaaripaino on oma ulottuvuutensa.
- Johdetaan gradientin lauseke funktio(i)lle $f_a \colon \mathcal{M} \mapsto \mathbb{R}$ joka kertoo, miten kaaripainoja pitää päivittää, kun halutaan muuttaa nykyinen verkko sellaiseksi, joka muistaa (myös) seuraavan esimerkin a.
  Esimerkki a on funktiossa ikään kuin tuntematon vakio.
- Opettamisessa
  1. valitaan jokin esimerkki
  2. muistetaan se päivittämällä kaaripainoja gradienttisäännöllä

  kunnes jäljelle jää enää pieni virhe.

Vuoristokiipeilyalgoritmilla 7 on kolme tunnettua ongelmamaastoa:

Paikallinen huippu (Local Maximum) on pieni ympäristöään korkeampi mäennyppylä.
Vaeltaja kiipeää sille luullen päässeensä koko vuoren huipulle, koska sen ympärillä on pelkkää alamäkeä niin pitkälle kuin vaeltajan lähes sokea silmä kantaa.

Tasanko (Plateau) on alue, jolla korkeus ei vaihtele niin pitkälle kuin silmä kantaa.
Tasangolla vaeltajamme hoipertelee *satunnaiskävelyn (random walk)*.

Harjanne (Ridge) viettää sekä "oikealle" että "vasemmalle" jyrkästi alaspäin.
Jos vaeltajan askelvalikoimasta puuttuu juuri jyrkänteen harjaa "eteenpäin" vievä suunta, niin vaeltaja kulkee epätaloudellisesti hyppien rinteen puolelta toiselle.

Kaikissa tapauksissa vaellus saavuttaa sellaisen maaston, jossa etenemistä ei enää tapahdu tai se on tuskallisen hidasta.

Yksi tapa yrittää välttää näitä ongelmia on *aloittaa satunnaisesti (Random-Restart)* uusi vaellus jostakin toisesta etsintäavaruuden satunnaisesta kohdasta sen jälkeen kun edellinen on pysähtynyt, ja kokeilla löytääkö se aiempaa korkeamman huipun.

Milloin satunnaisen aloittamisen voi lopettaa?

- Kun vaelluksia on tehty ennalta kiinnitetyt L kappaletta.
- Kun uusia entistä korkeampia huippuja ei enää ole löytynyt viimeisten L vaelluksen aikana.

Optimiratkaisun löytyminen on

- todennäköistä kun L on "riittävän suuri"
- nopeaa jos maaston huippuja on vähän ja ne erottuvat selvästi ympäristöstään.

Vaikeiden etsintäongelmien maasto

- sisältää eksponentiaalisen monta paikallista huippua mihin takertua ja mitä väistellä
- eli on piikikäs kuin siilin selkä.

Simuloitu jäähdytys/mellotus (Simulated Annealing) tarjoaa systemaattisemman tavan väistellä paikallisia huippuja.

Simuloidussa jäähdytyksessä tarkastellaan joka hetki *satunnaista* mahdollista askelta $next$:

- Jos $next$ veisi ylöspäin, niin otetaan se.
- Jos $next$ veisikin *alaspäin*, niin *arvotaan* otetaanko se vai jäädäänkö paikoilleen.

  Tässä arvonnassa askeleen $next$ todennäköisyys on sitä pienempi mitä
  - suurempi on vastaava pudotus ΔE, ja
  - mitä kauemman aikaa t vaellus on jo kestänyt.

Silloin vaelluksen

alussa pyritään väistelemään paikallisia huippuja laskeutumalla usein alaspäin

lopussa on todennäköisesti päädytty jonkin korkean huipun lähistölle, joten aletaan yrittämään useammin sitä ylöspäin.

Siirtyminen alkuvaiheesta loppuvaiheeseen on *asteittainen*.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  static: current, a node
          next, a node
          T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
     T ← schedule[t]
     if T=0 then return current
     next ← a randomly selected successor of current
     ΔE ← VALUE[next] – VALUE[current]
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

Algoritmi 8: Simuloitu jäähdytys [RN95, kuva 4.15].

Asteittaiseen menetelmään päädyttiin *fysiikan* kautta:

1. Lämmitetään kiinteää rautamöykkyä R sellaiseen korkeaan lämpötilaan T, jossa se sulaa hyvin juoksevaksi nesteeksi.

   Tehdään tämä upottamalla R sitä itseään paljon suurempaan lämpöhauteeseen, joka on lämmitetty lämpötilaan T.
2. Silloin möykyn R jokainen atomi a pääsee liikkumaan (melko) vapaasti.
3. "Luonnonlakien mukaan" kukin atomi a on satunnaisessa lämpöliikkeessä, jossa atomin a oma energiatila voi

   nousta ottamalla lämpöenergiaa hauteesta (joka ei moisesta kylmene)

   laskea luovuttamalla lämpöenergiaa hauteeseen (joka ei moisesta lämpene).

Energiatilan laskeminen on nousemista todennäköisempää.

4. Näin koko R todennäköisesti "rauhoittuu" aikanaan tilaan, jossa valtaosa sen atomeista a on energiaminimissään.
   Tämä energiaminimi riippuu nykyisestä lämpötilasta T.
5. Jäähdytetään rautaa R laskemalla hauteen lämpötilaa T hieman.
   Odotetaan taas että valtaosa sen atomeista a ehtii jälleen etsiytyä uuteen energiaminimiinsä.
6. Jäähdytystä jatketaan varovasti, kunnes R on taas kiinteää.
   Nyt sen atomit a ovat hyvin todennäköisesti hyvin lähellä kiinteän olomuodon energiaminimiään, eli raudan luontaista hilarakennetta.

Näin saadaan *meltorautaa*, jonka pitäisi olla rakenteeltaan hyvin homogeenistä sekä vapaata sisäisistä jännityksistä ja (hius)halkeamista.

Simuloitu jäähdytys saadaan tästä prosessista siten, että kohdefunktion vastaluku $-\textsc{Value}(R)$ kertoo möykyn R *kokonaisenergian* joka pyrkii siis minimoitumaan.

Jos. . .

- jäähdytys aloitetaan "tarpeeksi kuumasta" alkulämpötilasta, ja
- lämpötilaa lasketaan "tarpeeksi hitaasti"

niin. . .

- simuloitu jäähdytys päätyy lopuksi globaaliin maksimikohtaan todennäköisyydellä 1

mutta. . .

- "todennäköisyydellä 1" ei ole sama asia kuin "varmasti", ja
- vastaava *jäähdytysstrategia* (jonka määrää algoritmin 8 funktio $schedule$) on eksponentiaalisen hidas.

Simuloitu jäähdytys on osoittautunut *käytännössä tehokkaaksi optimointimenetelmäksi* varsinkin silloin kun ongelma on hyvin suuri, esimerkiksi mutkikkaiden VLSI-piirien optimoinnissa.

Eräänlainen iteratiivinen parannusmenetelmä ovat myös *geneettiset algoritmit (Genetic Algorithms, GA)*:

- Niissä simuloidaan jäähtymisen sijasta
  - geenien sekoittumista suvullisessa lisääntymisessä
  - mutatoitumista yksilöissä.
- Simuloitu luonnonvalinta ohjaa populaatiota sukupolvi sukupolvelta soveltumaan paremmin ympäristöönsä eli ratkaistavaan ongelmaan.

Näitä menetelmiä ei kurssilla kuitenkaan käsitellä.

4.2. Rajoiteongelmat [RN95, luvut 3.7, 4.2 ja 4.4] [Nil98, luvut 11.3-4]

Rajoiteongelmissa luvun 3 maaliehdon tiedetään jakautuvan konjunktioksi paikallisia ehtoja ("ehto 1 **ja** ehto 2 **ja** . . . ") joita voidaan tarkastella osin erikseen.

Kuvan 38 mukainen 8 kuningattaren ongelma voidaan muotoilla *rajoiteongelmana (Constraint Satisfaction Problem, CSP)* seuraavasti:

1. Tavoitteena on etsiä sopivat arvot *muuttujille* $X_1, \ldots, X_8$.

   Muuttujan X_i arvo kertoo silloin sarakkeelle i asetetun kuningattaren i rivin.

   Muuttujia on aina äärellinen määrä.

2. Kukin muuttuja saa arvokseen rivinumeron

   $$X_i \in D_i = \{1, \ldots, 8\}\,.$$

   Joukkoa D_i kutsutaan muuttujan X_i *arvoalueeksi (domain)*.

   Arvoalue voi yleisessä tapauksessa olla myös ääretön kuten $\mathbb{Q}$ tai jopa jatkuva kuten $\mathbb{R}$.

   Jokaista tällaista *yhtä* muuttujaa koskevaa ehtoa "$X_i \in D_i$" kutsutaan *unaariseksi rajoitteeksi (unary constraint)*.

3. Kaikille $i < j$ pitää päteä, että kuningattaret ruuduissa X_i ja X_j eivät uhkaa toisiaan.

   Jokaista tällaista *kahta* muuttujaa koskevaa ehtoa kutsutaan *binääriseksi* rajoitteeksi.

   Sitä voidaan ajatella karteesisen tulon $D_i \times D_j$ sellaisena osajoukkona $R_{i,j}$, joka luettelee kaikki sallitut parit.

Rajoitteet voivat yleisessä tapauksessa käsittää myös 3 tai useampia muuttujia kerrallaa.

Jos kuitenkin kaikki arvoalueet ovat äärellisiä, niin useampimuuttujaiset rajoitteet voidaan korvata sopivilla uusilla unaari- ja binäärirajoitteilla.

Rajoite voi olla

ehdoton ***(absolute)*** jolloin ratkaisun on pakko toteuttaa se (kuten 8 kuningattaren ongelmassa)

suositus ***(preference)*** jolloin voidaan hyväksyä myös sellainen ratkaisu, joka ei sitä täytä, jos parempaan ei pystytä.

Esimerkiksi laitoksen opetusohjelmaa laadittaessa

ehdoton rajoite on, että samaan saliin ei voi sijoittaa samaan aikaan kahta eri luentoa

suositus on, että cum laude -tason pakollisten kurssien luentoja ei sijoitettaisi samaan aikaan.

Keskitytään ratkomaan *ehdottomista äärellisistä unaari- ja binäärirajoitteista koostuvia ongelmia* oppimillamme menetelmillä.

Luvun 3.1 systemaattisia etsintämenetelmiä voidaan soveltaa seuraavasti:

- Tavoitteena on löytää arvot äärelliselle määrälle muuttujia $X_1, \ldots, X_N$.

  Tilannetta voi ajatella etsintäpuuna, jossa syvyydellä d etsitään arvoa muuttujalle X_{d+1}, ja mahdolliset teot ovat D_{d+1}.

  Voidaan käyttää luvun 3.1.3 *syvyyssuuntaista etsintää*:

  nyt se pysähtyy itsestään viimeistään syvyydellä N (eli on täydellinen).

  Kuningataresimerkissämme $N = 8$.
- Olkoon syvyysetsintä jo antanut arvot muuttujille X_i, missä $1 \leq i \leq j$.

  Valitaan seuraavalle muuttujalle X_{j+1} sopivaa arvoa $V \in D_{j+1}$.

Ensin voidaan tutkia kaikki binäärirajoitteet muuttujien X_i ja X_{j+1} välillä, ja *karsia* joukosta D_{j+1} pois ne ehdokkaat V jotka rikkoisivat jonkin niistä.

Jos yhtään ehdokasta ei jää jäljelle, voidaan *peruuttaa (backtrack)* etsinnässä jo nyt: koko solmusta X_{j+1} alkava alipuu voidaan jättää tutkimatta, koska jo siihen vievä polku on havaittu mahdottomaksi.

Näin rajoiteongelmissa voimme kysyä väliaikatietoja "Onko tämä keskeneräinen ratkaisu oikein tähän saakka?". Se tehostaa etsintää yleisiin ongelmiin verrattuna.

Esimerkissämme tämä strategia tarkoittaa, että kuningatar $j+1$ laitetaan aina sellaiseen ruutuun, jota mikään jo laitetuista kuningattarista $1, \ldots, j$ ei uhkaa.

- Kääntäen voimme kuningatarta $j+1$ asettaessa merkitä samalla ne ruudut, joita se uhkaa.

  Silloin voimme katsoa, jääkö laudalle tarpeeksi uhattomia ruutuja lopuille kuningattarille $j+2, \ldots, 8$, kun kuningatarten $1, \ldots, j+1$ uhkaamat ruudut on poistettu.

  Tätä strategiaa kutustaan *eteenpäin tarkistamiseksi (forward checking)*.

  Sen avulla voidaan käytännössä ratkoa esimerkkiongelmamme laajennuksia aina noin tapaukseen $N = 30$ saakka (eli sovittaa N kuningatarta $N \times N$ ruudun laajennetulle shakkilaudalle).

  Kun siis valitaan jokin $X_{j+1} = V$, niin samalla poistetaan kaikista binäärirajoitteista $R_{j+1,j+k}$, missä $k > 1$, kaikki parit jotka eivät ole muotoa $\langle V, W \rangle$ (missä siis $W \in D_{j+k}$). Jos jokin niistä tyhjenee, niin voidaan taas peruuttaa.

- Tämä poistostrategia on *kaarikonsistentiksi (arc consistency) pakottamista*:

  Etsinnän tila on kaarikonsistentti jos jokaisen muuttujan X_i arvoalueessa D_i on edelleen jäljellä jokin sellainen arvo V_i, joka kelpaa kaikille muuttujaan X_i liittyville rajoitteille.

  Tai kääntäen, $V_i \notin D_i$ jos yksikin muuttujaan X_i liittyvä rajoite kieltää muuttujalta X_i arvon V_i.

  Toisin sanoen, D_i sisältää joka hetki vain ne muuttujan X_i

arvot V_i jotka ovat yhä *mahdollisia, kun ongelmaa katsellaan vain muuttujan X_i lähistöltä*, ei kokonaisuutena.

  – Monimutkaisemmissa ongelmissa lähistöksi voidaan ottaa yhden sijasta $k = 2, 3, 4, \ldots$ muuttujaa.
  – Eräs tapa ratkoa rajoiteongelma(luokka)a onkin analysoida tarvittavan lähistön kokoa k, ja pakottaa ongelmat k-konsistenteiksi. Tarpeeksi suurella k voidaan jopa *välttää peruutus*.
  – Toisaalta k on pakottamisen aikavaatimuksen eksponentissa. Lisäksi "tarpeeksi suuri" k voi lähestyä muuttujien lukumäärää N.
  – Yleensä tyydytäänkin 2- eli *polku*konsistenssiin *(path c.)*.

- Kaarikonsistentiksi pakottaminen tapahtuu poistamalla mahdottomia arvoja alueista D_i ja mahdottomia pareja rajoitteista $R_{i,j}$ niin pitkään kuin mahdollista.

  Kukin poisto voi mahdollistaa uusia poistoja.

  Jos jokin joukko tyhjenee, ei kyseisellä rajoiteongelmalla ole ratkaisuja.

  Tämä pakottamismenetelmä on eräs *rajoitteiden levittämisen (constraint propagation)* erikoistapaus.

- Yleisesti voidaan rajoiteongelmaa ratkoa rajoitteita levittämällä vaikkapa seuraavasti:

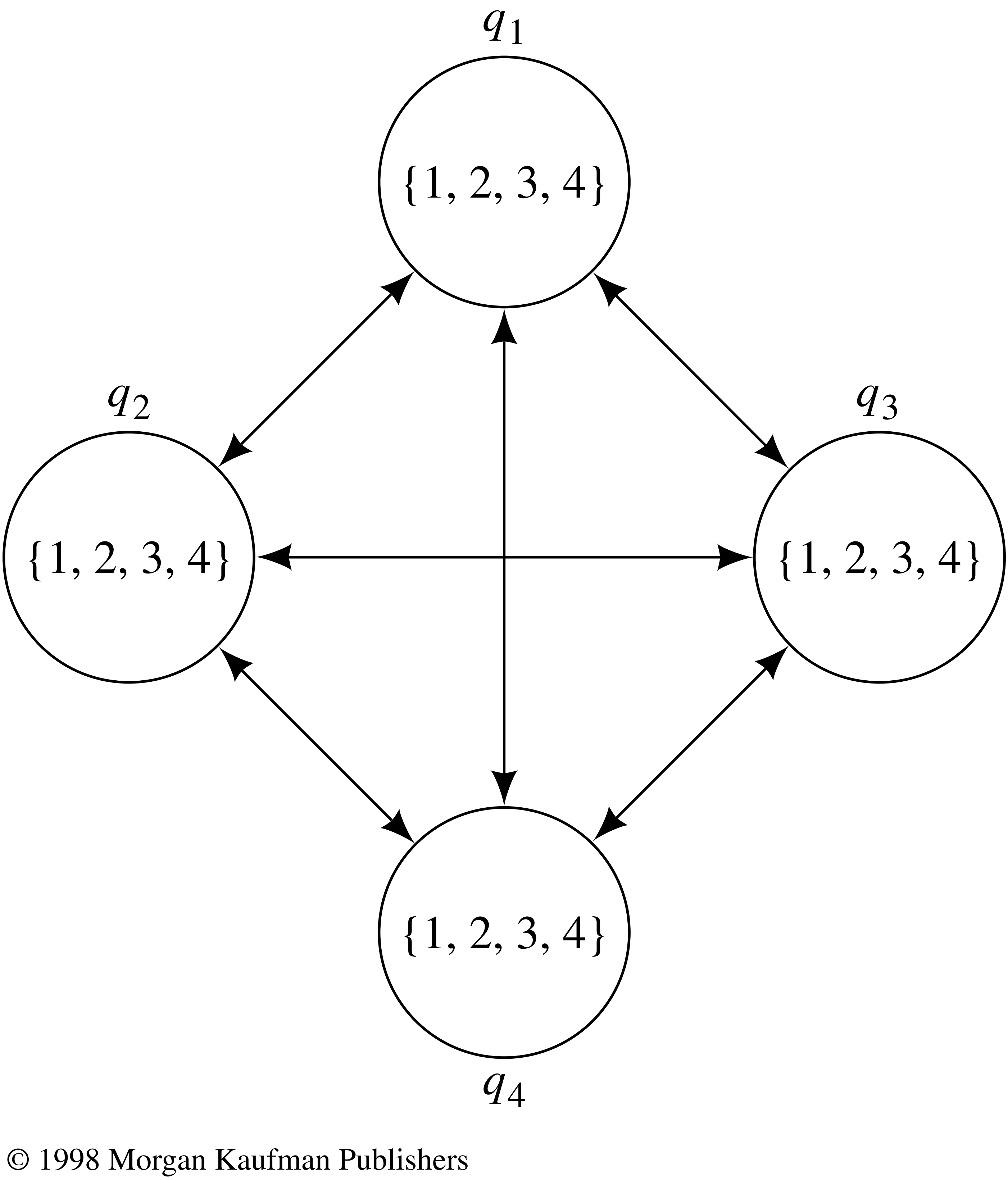
  1. Aloitetaan ongelmasta, jossa tiedetään vain unaarirajoitteet, eli kukin $R_{i,j} = D_i \times D_j$.
  2. Lisätään muuttujien X_i ja X_j välinen binäärirajoite $R'_{i,j} \subseteq D_i \times D_j$ poistamalla pareista $R_{i,j}$ kaikki ne jotka eivät kuulu rajoitteeseen $R'_{i,j}$.
  3. Poistoja jatketaan valituksi k-konsistentiksi saakka.
  4. Sama toistetaan kaikille binäärirajoitteille.
  5. Lopuksi suoritetaan syvyyssuuntainen etsintä jäljelle jääneessä rajoiteongelmassa.

- Rajoitteiden levittämistä käytetään myös muiden etsintämenetelmien
  – esivaiheena
  – osana

  karsimaan etsintäavaruutta ennen raskasta hakua.

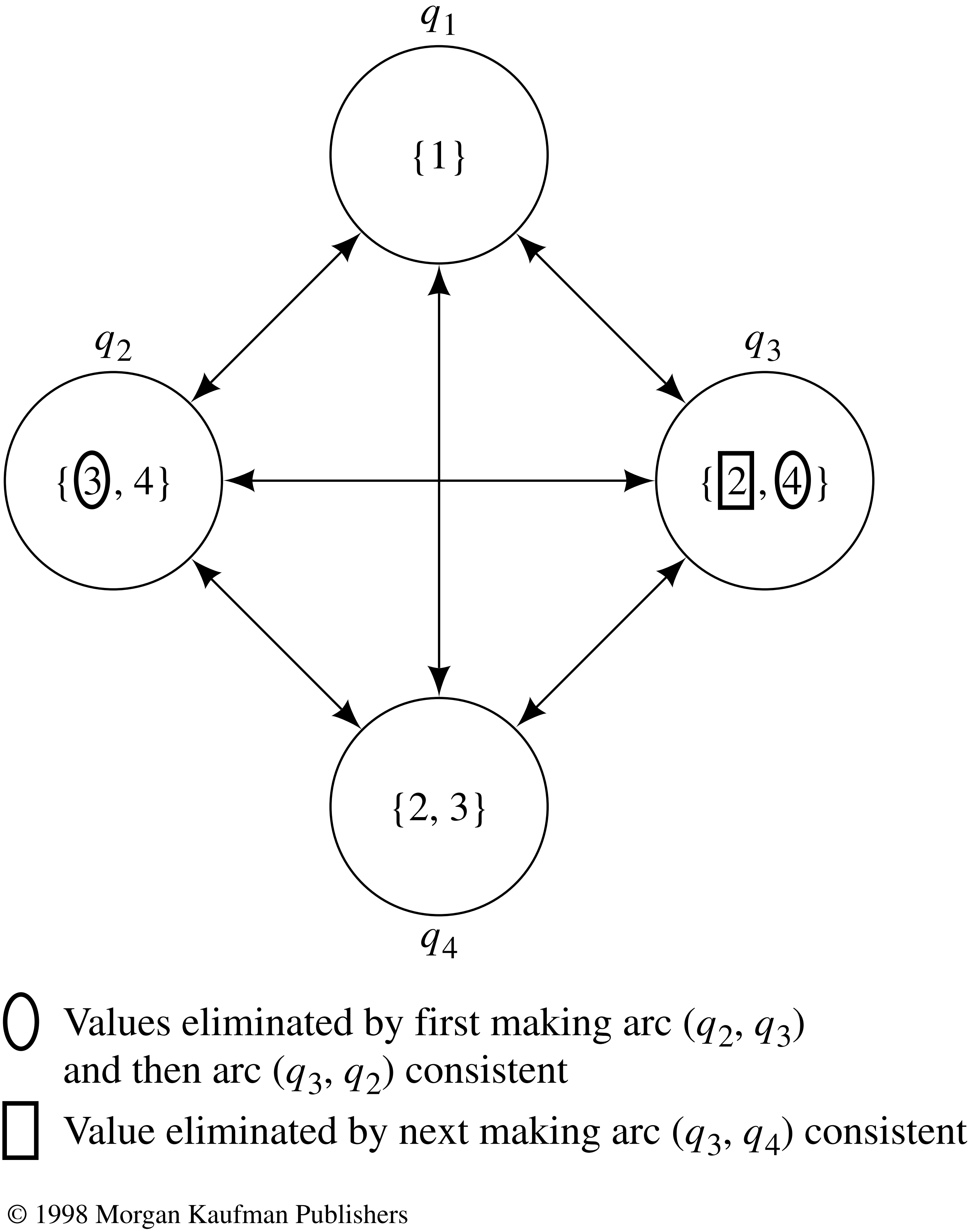
- Tarkastellaan esimerkkinä 4 kuningattaren sijoittelua 4×4-shakkilaudalle.

Alkutilanne on kuvassa 41: kuningattaret ovat muuttujina $q_1, \ldots, q_4$ joille sallitaan rivit $1, \ldots, 4$.



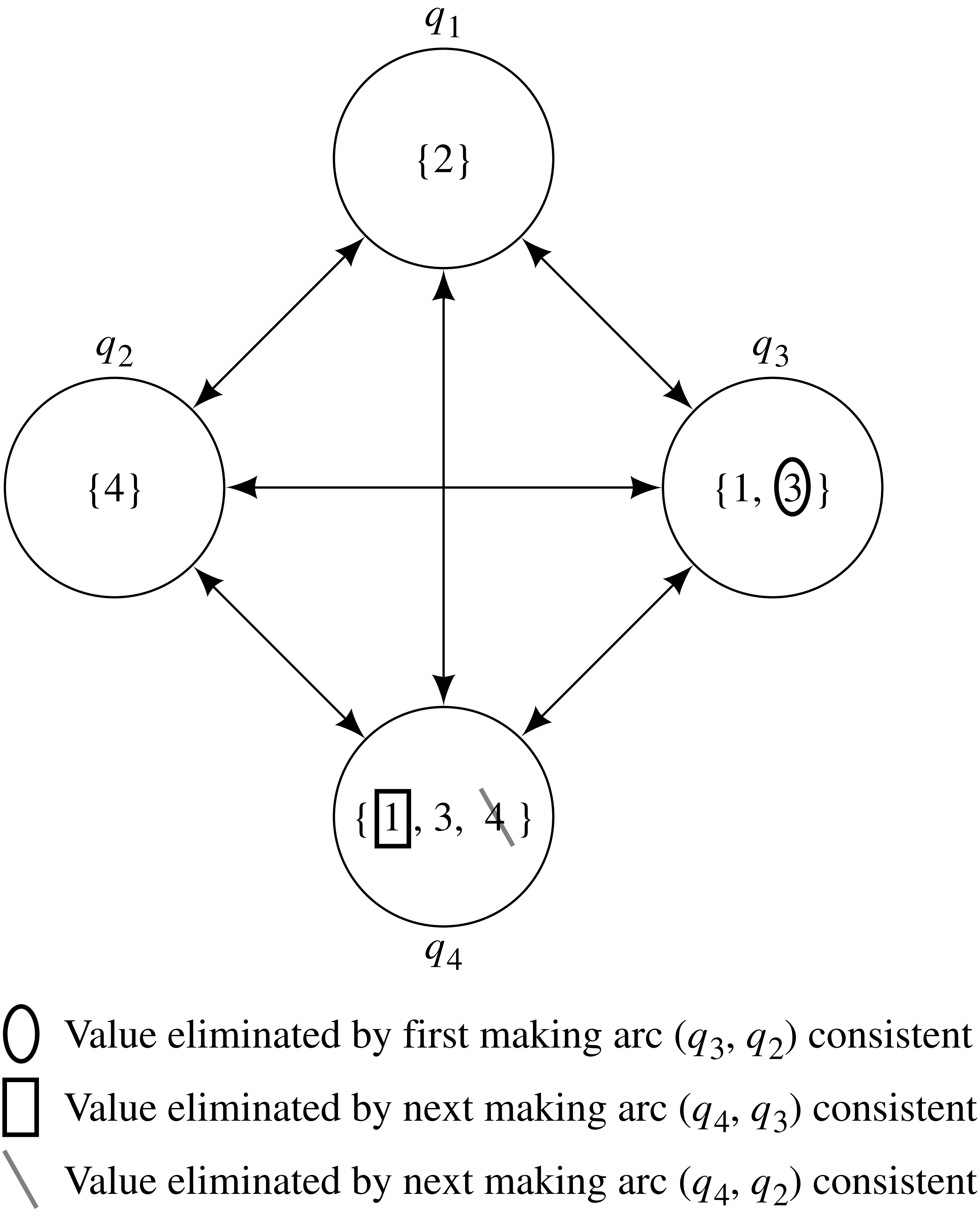
Kuva 41: 4 kuningattaren rajoiteongelman alkutila [Nil98, kuva 11.4].

Jos valitaan ensin $q_1 = 1$, niin päädytään kuvan 42 kautta umpikujaan.



Kuva 42: 4 kuningattaren rajoiteongelma kun $q_1 = 1$ [Nil98, kuva 11.5].

Peruutetaan valintaan $q_2 = 2$, josta kuvan 43 kautta ratkaisuun.

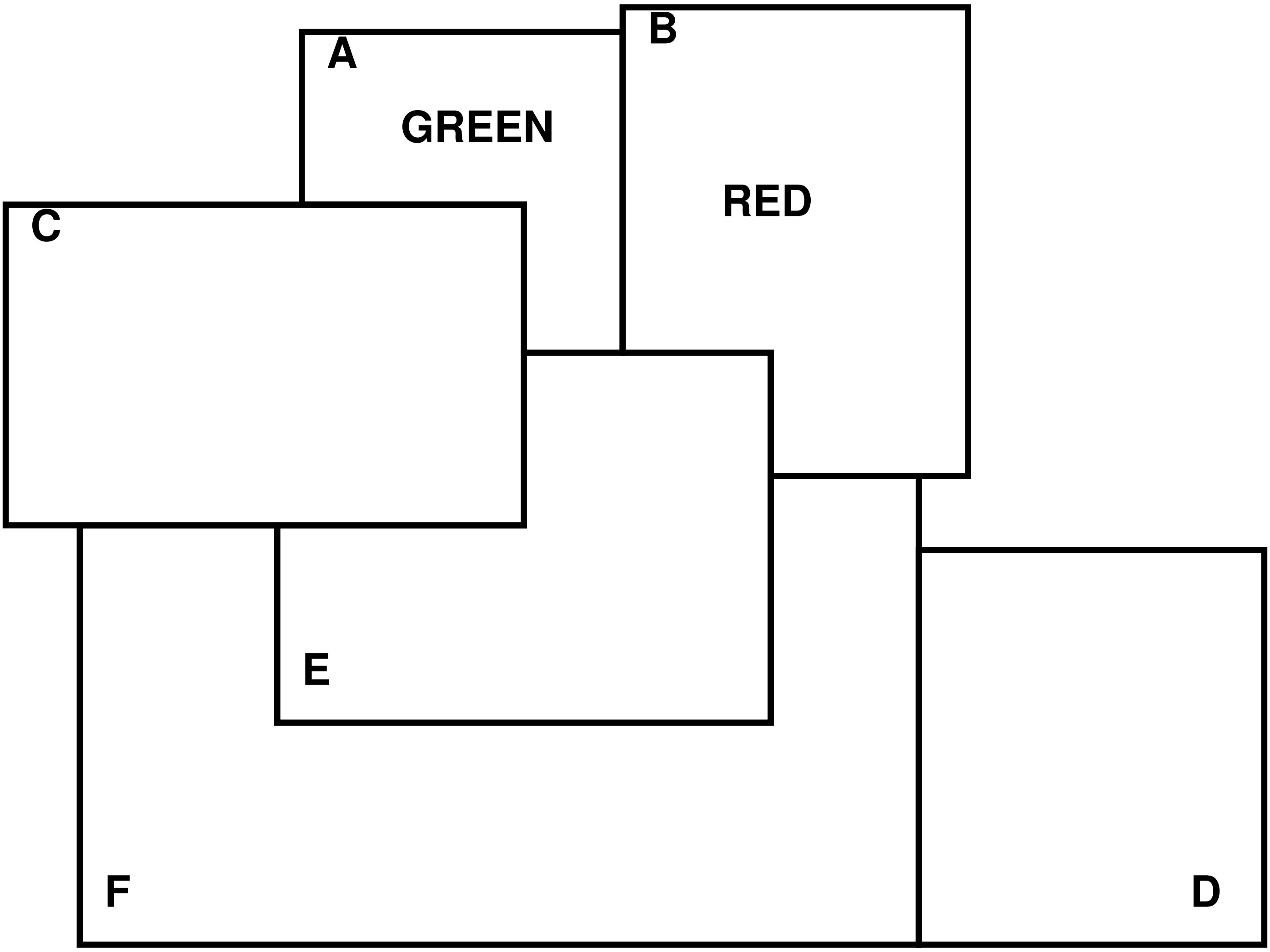


Kuva 43: 4 kuningattaren rajoiteongelma kun $q_1 = 2$ [Nil98, kuva 11.6].

- Rajoiteongelmissa käytetään heuristiikkoja *valitsemaan seuraava muuttuja jolle määrätään arvo*. (Muuttujia ei siis enää käsitellä ennalta kiinnitetyssä järjestyksessä.)

  Tämä on erilaista heuristista etsinnän ohjausta kuin luvussa 3.2.

- Tarkastellaan esimerkiksi ongelmaa, jossa seuraava kuvio pitää värittää 3 värillä siten, että naapuruksilla on aina eri värit. Kaksi aluetta A ja B on jo väritetty vihreäksi (green) ja punaiseksi (red).



Kuva 44: Kolmiväriongelma [RN95, kuva 4.9].

- Seuraavaksi kannattaa värittää alue E siniseksi (blue), koska sillä ei ole enää muita vaihtoehtoja. Sitten:

  1. C voi olla vain punainen.
  2. F voi olla vain vihreä.
  3. D voi olla joko sininen tai punainen muttei vihreäksi.

  Tämä on ääriesimerkki *eniten rajoitettu muuttuja (most-constrained-variable) -heuristiikasta*:

Valitse käsittelyyn sellainen muuttuja X_j, jonka arvoalueessa D_j on *vähiten mahdollisia vaihtoehtoja* jäljellä.

Tämä valinta voidaan tehdä luontevasti eteenpäin tarkistettaessa.

Intuitiivisesti tämä heuristiikka pyrkii minimoimaan etsinnän *välitöntä* haarautumista valitsemalla aina vähiten haaroja synnyttävän vaihtoehdon.

Esimerkiksi kuvan 38 kuningatarongelmassa päästään aina tapauksiin $N = 100$ saakka, kun valitaan aina sellainen sarake j, jolla on vähiten vielä mahdollisia ruutuja.

- Toinen haarautumista ehkäisevä heuristiikka on *eniten rajoittava (most-constraining)* muuttuja -heuristiikka: valitse käsittelyyn sellainen muuttuja, joka on mukana mahdollisimman monessa binäärirajoittessa vielä käsittelemättömien muuttujien kanssa.

  Intuitiivisesti tämä heuristiikka pyrkii minimoimaan etsinnän *tulevaa* haarautumista valitsemalla aina sellaisen vaihtoehdon, jonka jälkeen myöhemmillä vaihtoehdoilla on vähän haaroja jäljellä.

- Kun muuttuja X_j on valittu jommalla kummalla heuristiikalla, on vielä valittava sille jokin mahdollinen arvo $V \in D_j$.

  Vähiten rajoittava arvo (least constraining value) -heuristiikka valitsee muuttujalle X_j sellaisen arvon $V \in D_j$, joka poistaa vähiten mahdollisia arvoja vielä käsittelemättömiltä muuttujilta.

  Valitaankin kuvassa 44 ensin käsittelyyn alue C (eikä E). Jos se väritetään

  siniseksi niin alueelle E ei jää yhtään mahdollista väriä.
  Alueelle F jää vain vihreä.

  punaiseksi niin alueelle E jää sininen.
  Alueelle F jää vihreä ja sininen (jonka E kieltää vasta myöhemmin).

  Valitaan siis punainen väri.

  Kuvan 38 kuningatarongelmassa päästään aina tapauksiin $N = 1\ 000$ saakka, jos valitaan sarakkeen j ruuduista se, joka synnyttää vähiten uusia uhkauksia vielä käsittelemättömille riveille.

Luvun 4.1 iteratiiviset parannusmenetelmät antavat *heuristisia korjausmenetelmiä (heuristic repair methods)*: ne ratkovat rajoiteongelmia tekemällä pieniä muutoksia olemassa olevaan puutteelliseen ratkaisuun.

Eräs heuristiikka on *konfliktien minimointi (min-conflicts)*: (jollakin muulla heuristiikalla) valitulle muuttujalle X_j kokeillaan jotakin sellaista uutta arvoa $V \in D_j$, joka toteuttaa mahdollisimman monta tähän muuttujaan liittyvää binäärirajoitetta.

Kuvasarja 38 antaa esimerkin 8 kuningattaren ongelman ratkaisusta tällä periaatteella. Sen avulla jopa satunnainen tapaus $N = 1\ 000\ 000$ ratkeaa keskimäärin alle 50 muutoksella!

Esitetään vielä vuorottain rekursiivinen kaarikonsistenssialgoritmi, joka

- kieltää mahdottomana arvon $X_i = v_i$ poistamalla alkion v_i mahdollisten arvojen joukosta D_i
  (aliohjelma `poista-yksi`)
- kieltää mahdottomana arvoyhdistelmän $X_i = v_i, X_j = v_j$ poistamalla parin $\langle v_i, v_j \rangle$ mahdollisten arvoparien joukosta $R_{i,j}$
  (aliohjelma `poista-pari`)
- jatkaa näitä poistoja rekursiivisesti kunnes
  joko konsistenssi on palautettu jälleen voimaan poistamalla kaikki mitä voidaan
  tai mahdollisten joukot tyhjenevät, mikä osoittaa ettei koko rajoiteongelmalla ole yhtään ratkaisua.

Esityksen yksinkertaistamiseksi oletetaan, että

$$R_{j,i} = \{ \langle v_j, v_i \rangle :\ \langle v_j, v_i \rangle \in R_{i,j} \}$$

kun $i > j$, eli että indeksien järjestyksellä ei ole väliä.

Algoritmia voi optimoida eri tavoin.

Polkukonsistenssi saataisiin muuttamalla algoritmia sellaiseksi, joka saattaa voimaan seuraavan "kolmioehdon":

Kun jokin joukko D_j kutistuu, niin

- Tutki jokainen erillinen (muuttuja)indeksipari i, k (missä siis $i \neq j \neq k \neq i$).
- Poista indeksiparin joukosta $R_{i,k}$ jokainen sellainen arvopari $\langle v_i, v_k \rangle$ joka. . .
- . . . menetti viimeisenkin välittävän alkion $v_j \in D_j$, eli sellaisen jolla $\langle v_i, v_j \rangle \in R_{i,j}$ ja $\langle v_j, v_k \rangle \in R_{j,k}$.

procedure `poista-yksi`(i, v_i)

if $v_i \in D_i$ **then**
    $D_i \leftarrow D_i \setminus \{v_i\}$;
    for all $\langle v_i, v_j \rangle \in R_{i,j}$ missä $j \neq i$ **do**
        `poista-pari`$(i, j, \langle v_i, v_j \rangle)$
    end for
end if

procedure `poista-pari`$(i, j, \langle v_i, v_j \rangle)$

if $\langle v_i, v_j \rangle \in R_{i,j}$ **then**
    $R_{i,j} \leftarrow R_{i,j} \setminus \{\langle v_i, v_j \rangle\}$;
    if $R_{i,j}$ ei sisällä yhtään paria $\left\langle v_i, v'_j \right\rangle$ **then**
        `poista-yksi`(i, v_i)
    end if
    if $R_{i,j}$ ei sisällä yhtään paria $\left\langle v'_i, v_j \right\rangle$ **then**
        `poista-yksi`(j, v_j)
    end if
end if

Algoritmi 9: Kaarikonsistenssialgoritmi.

- 3-konsistenssissa kolmiosta i, j, k tulee "tetra" i, j, k, l
- 4-konsistenssissa "pentagrammi" i, j, k, l, m
- ja k-konsistenssissa täydellinen $k + 1$ solmun aliverkko.

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].