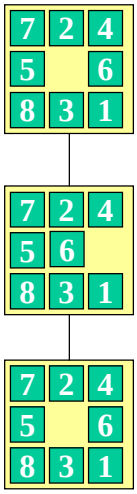
- Haun merkittävä ongelma on mahdollisuus laajentaa uudelleen jo kertaalleen tutkittu tila
- Näin äärellinen tila-avaruus voi johtaa äärettömään hakupuuhun
- Ratkeava ongelma voi muuttua käytännössä ratkeamattomaksi
- Jo laajennettujen tilojen tunnistaminen edellyttää käsiteltyjen tilojen tallentamista muistiin
- Näin on tehtävä myös syvyyssuuntaisessa haussa
- Toisaalta karsinta voi johtaa optimaalisen polun karsimiseen



Haku osittaisen informaation vallitessa

- Edellä teimme (epärealistisen) oletuksen, että maailma on täysin havainnoitavissa ja deterministinen
- Tällöin agentti aina tietää täsmälleen missä tilassa se on ja voi laskea toimintojonon suorituksesta seuraavan jonon
- Toisaalta agentin havainnot eivät anna mitään uutta tietoa

- Realistisempi tilanne on, että agentin tieto tiloista ja toiminnoista on epätäydellinen
- Jos agentilla ei ole sensoreita lainkaan, niin sen näkökulmasta alkutila voi olla jokin monista ja täten myös toiminnot voivat johtaa moniin seuraajatiloihin

- Iman sensoreita toimiva agentti joutuu tekemään päättelyä tilajoukoissa tilojen sijaan
- Kullakin hetkellä agentilla on uskomus siitä, missä tiloissa se voi olla
- Toimitaan siis tila-avaruuden Σ potenssijoukossa $\mathcal{P}(\Sigma)$, jossa on 2^{Σ} uskomustilaa (belief state)
- Nyt ratkaisu on polku, joka johtaa uskomustilaan, jonka kaikki jäsenet ovat maalitiloja
- Jos toimintaympäristö on osittain havainnoitavissa tai jos toiminnot ovat epävarmoja, niin jokainen uusi toiminto antaa uutta informaatiota
- Jokainen mahdollinen havainto määrää tilanteen jota varten on tehtävä suunnittelua

- Epävarmuuden aiheuttaja voi olla toinen agentti, *vastustaja* (adversary)
- Maailman tilojen ja toimintojen seurausten ollessa epävarmoja, on agentin *tutkiskeltava* (explore) ympäristöä tietojen saamiseksi
- Osittain havainnoitavassa maailmassa toimintojonoja ei voi kiinnittää ennakkoon suunniteltaessa, vaan niiden tulee olla ehdollisia uusille havainnoille
- Kun agentti voi kerätä uutta tietoa toimintansa kautta, niin yleensä ei kannata tehdä suunnitelmia kaikkia mahdollisia tilanteita varten, vaan pikemminkin lomittaa toiminta ja suunnittelu

3.1 HEURISTINEN HAKU

- Sokean haun sijaan meillä on nyt ongelmakohtaista tietoa käytössämme etsinnän tehostamiseksi
- Tila-avaruuden säännöllisestä rakenteesta on ylimääräistä tietoa
- Hakupuun solmun n lupaavuus maalitilan löytymisen kannalta arvotetaan *evaluointifunktiolla* $f(n)$
- Paras ensin -haku laajentaa mahdollisista solmuista sen, joka näyttää evaluointifunktion arvon perusteella lupaavimmalta
- Yleensä pyrkimyksenä on funktion f arvon minimoiminen

Heuristiset funktiot

- Evaluointifunktion keskeinen komponentti on heuristinen funktio $h(s)$, joka antaa arvion tilasta s maalitilaan johtavan lyhimmän polun kustannukselle
- Etsintäongelman yhteydessä heuristiikalla tarkoitetaan varmaa (mutta löysää) ylä- tai alarajaa parhaan ratkaisun kustannukselle
- Maalitilat kuitenkin tunnistetaan: vastaavassa solmussa n vaaditaan, että $h(n) = 0$
- Esim. mitään informaatiota tuomaton varma alaraja olisi asettaa $h(s) \equiv 0$
- Heuristiset funktiot ovat yleisin tapa antaa ylimääräistä tietoa etsintäongelmaan

Ahne haku

- Pyritään ratkaisuun mahdollisimman nopeasti laajentamalla se solmu, joka on lähinnä maalia
- Evaluointifunktio siis on $f(n) = h(n)$
- Esim. maantie-etäisyyksien minimoinnissa heuristinen alaraja kaupunkien etäisyyksille voisi olla niiden etäisyys linnuntietä

- Ahne haku jättää huomiotta solmuun n jo kuljetun polun kustannuksen
- Täten sen tuottama ratkaisu ei ole välttämättä optimaalinen
- Jos toistuvia tiloja ei havaita, voi ahne haku oskilloida ikuisesti kahden lupaavan tilan välillä

- Äärettömien polkujen mahdollisuus merkitsee, ettei ahne haku ole myöskään täydellinen
- Ahneutensa vuoksi haku tekee valintoja, jotka voivat johtaa umpikujaan; tällöin hakupuussa peruutetaan takaisin syvimmällä olevaan laajentamattomaan solmuun
- Ahne haku muistuttaa syvyyssuuntaista hakua tavassaan tutkia hakupuuta oksa kerrallaan ja peruuttaa umpikujaan joutuessaan
- Pahimman tapauksen aika- ja tilavaativuus ahneella haulla on $O(b^m)$
- Heuristisen funktion hyvyys määrää ahneen haun käytännön toimivuuden

Hakualgoritmi A*

- A* laskee yhteen heuristisen funktion $h(n)$ arvon ja solmun n saavuttamisen vaatiman kustannuksen $g(n)$
- Evaluointifunktio

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

siis arvioi halvimman solmun n kautta kulkevan ratkaisun kustannusta

- Jos $h(n)$ ei koskaan yliarvioi maalin saavuttamisen kustannusta, niin puun läpikäynnissä A* palauttaa optimaalisen ratkaisun
  - Olkoon g_2 puuhun generoitu, ei-optimaalinen maalisolmu
  - Olkoon C^* optimiratkaisun kustannus
  - Koska g_2 on maalisolmu, niin $h(g_2) = 0$, joten $f(g_2) = g(g_2) > C^*$

- Toisaalta, jos ongelmaan on ratkaisu, niin puuhun on generoitu solmu n, joka kuuluu optimaaliseen polkuun
- Koska $h(n)$ on aliarvio, niin $f(n) = g(n) + h(n) \leq C^*$
- Näin ollen solmua g_2 ei valita tutkittavaksi

- Esimerkiksi linnuntie on arvio, joka ei koskaan yliarvioi maantietä kuljettavaa matkaa
- Verkkoetsinnässä optimaalisen ratkaisun löytäminen vaatii huolehtimaan, ettei optimaalista ratkaisua karsita toistuvissa tiloissa
- Erityisen tärkeä erikoistapaus ovat *monotoniset* heuristiikat

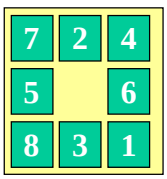
- Näille pätee *kolmioepäyhtälö* (triangle inequality) muodossa
  $$h(n) \leq c([n,a], n') + h(n'),$$
  missä $n' \in S(n)$ (valittu toiminto on a)
  ja $c([n,a], n')$ on askelkustannus
- Linnuntie on myös monotoninen heuristiikka
- Monotonisella heuristiikalla A* on optimaalinen myös verkkoetsinnässä
- Jos $h(n)$ on monotoninen, niin arvot $f(n)$ millä tahansa polulla ovat ei-vähéneviä, joten ensinnä tutkittu maali on optimaalinen ratkaisu
  $$f(n') = g(n') + h(n')$$
  $$= c([n,a], n') + g(n) + h(n')$$
  $$\geq g(n) + h(n) = f(n)$$

- Ratkaisua etsiessään A* laajentaa kaikki solmut n, joilla $f(n) < C^*$, ja osan niistä, joilla $f(n) = C^*$
- Sen sijaan kaikki solmut solmut n, joilla $f(n) > C^*$, *karsitaan*
- On selvää, että A* on täydellinen hakualgoritmi
- Se on myös *optimaalisen tehokas* menetelmä, sillä minkä tahansa hakualgoritmin on tutkittava kaikki solmut, joilla pätee $f(n) < C^*$
- Täydellisyydestä, optimaalisesta tehokkuudesta ja optimaalisen ratkaisun palauttamisesta huolimatta A*:llä on myös heikkoutensa
- Solmujen lukumäärä, joille pätee $f(n) < C^*$, voi olla eksponentiaalinen ratkaisun pituuden suhteen

- Jälleen kerran hakualgoritmin tilavaativuus on kuitenkin pahempi ongelma
- Sen tähden monia vain rajoitetun muistin käyttäviä versioita A*-algoritmista on jouduttu kehittämään
- IDA* (Iterative Deepening A*) soveltaa iteratiivisen syventämisen ideaa
- Solmujen syvyyden sijaan katkaisuehtona on tällä kertaa f-kustannus
- Kullakin kierroksella uudeksi katkaisuehdoksi valitaan edelliseltä kierrokselta karsittujen solmujen pienin f-arvo
- Uudemmat algoritmit tekevät monimutkaisempaa karsintaa

Heuristisista funktioista

- 8-puzzlessa voimme määritellä seuraavat heuristiset funktiot, jotka eivät koskaan yliarvioi
  - h_1: väärässä positiossa olevien laattojen lukumäärä: kutakin on siirrettävä ainakin kerran ennen kuin laatat ovat halutussa järjestyksessä
  - h_2: väärässä positiossa olevien laattojen Manhattan etäisyys oikeasta paikastaan: laatat on kuljetettava oikeille paikoilleen ennen kuin ne ovat halutussa järjestyksessä
- Alkutilanteessa kaikki laatat ovat väärässä positiossa: $h_1(s_1) = 8$
- Toisen heuristiikan arvo esimerkkikuvalle on:
  $3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$



- Heuristiikka h_2 antaa aina tiukemman arvion kuin h_1
- Sen sanotaan *dominoivan* jälkimmäistä
- Koska A* laajentaa kaikki solmut joilla $f(n) < C^* \Leftrightarrow h(n) < C^*-g(n)$, niin tiukempi heuristinen arvio johtaa suoraan tehokkaampaa etsintään

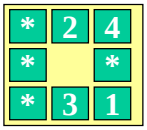
- 8-puzzlen haarautumisaste on noin 3
- Efektiivisellä haarautumiskertoimella mitataan ongelman ratkaisussa generoitavien solmujen keskimääräistä määrää
- Esim. kun A* käyttää heuristiikkaa h_1, niin 8-puzzlen efektiivinen aste on keskimäärin n. 1.4, ja heuristiikalla h_2 n. 1.25

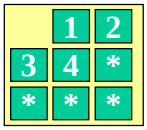
- Heurististen funktioiden kehittämiseksi voidaan tarkastella *relaksoituja* ongelmia, joista on poistettu joitain alkuperäisen ongelman rajoitteita
- Relaksoidun ongelman optimiratkaisun kustannus on aina aliarvio alkuperäisen ongelman ratkaisun kustannuksesta
- Alkuperäisen ongelman optimiratkaisu on myös helpotetun ongelman ratkaisu
- Esim. 8-puzzlen heuristiikka h_1 on antaa optimaalisen kustannuksen helpotettuun peliin, jossa laatat voivat pompata mihin paikkaan tahansa
- Samoin h_2 antaa optimaalisen ratkaisun relaksoituun 8-puzzle peliin, jossa laatat voivat liikkua myös miehitettyihin ruutuihin

- Jos tarjolla on joukko heuristisia funktioita, joista yksikään ei dominoi toisia, niin voimme käyttää koostettua heuristiikkaa

$$h(n) = \max\{ h_1(n), \ldots, h_m(n) \}$$

- Koostefunktio dominoi kaikkia komponenttifunktioitaan ja on monotoninen, jos komponentit eivät yliarvioi
- Yksi relaksoimisen muoto on aliongelmien tarkastelu
- Esim. 8-puzzlessa voisimme tarkastella vain neljää laattaa kerrallaan ja antaa loppujen ajautua mihin tahansa positioon
- Koostamalla eri laattoja koskevat heuristiikat yhteen funktioon saadaan koko ongelman heuristinen funktio, joka on paljon Manhattan etäisyyttä tehokkaampi



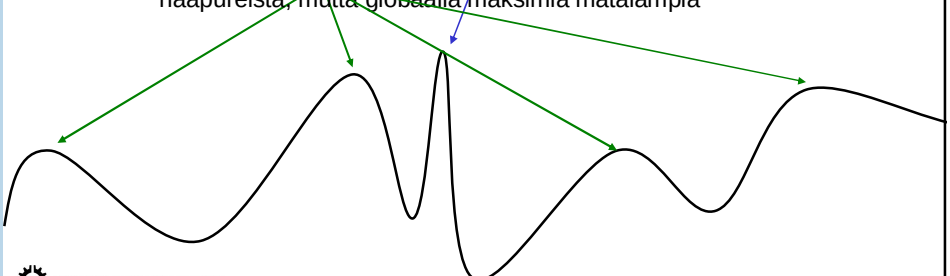


Lokaali haku

- Kaikissa ongelmissa maaliin johtavalla polulla ei ole merkitystä
- Vain ongelman ratkaisu on tarpeen tietää
- Lokaalit hakualgoritmit pitävät yllä vain tietoa nykyisestä tilasta ja etsintä keskittyy yleensä vain sen naapureihin
- Etsinnän kulkemia polkuja ei yleensä talleteta
- Muistitilan tarve on vähäinen, vain vakiotila
- Toimivat suurissa, jopa äärettömissä (jatkuvissa), maailmoissa löytäen hyviä ratkaisuja kun systemaattiset hakumenetelmät eivät ole laisinkaan sovellettavissa

- Lokaali haku on myös optimointiongelmien ratkaisumenetelmä
- Optimoinnissa tavoitteena on *kohdefunktion* (objective function) suhteen parhaan tilan löytäminen
- Optimointiongelmat eivät aina ole hakuongelmia samassa mielessä kuin edellä olemme tarkastelleet
- Esimerkiksi Darwinilainen evoluutio pyrkii optimoimaan lisääntymiskykyisyyttä
- Mitään lopullista maalitilaa ei ole olemassa
- Myöskään polun kustannuksella ei ole merkitystä tässä tehtävässä

- Kohdefunktion arvon optimointia voidaan kuvata tilamaisemana, jossa funktion arvoa vastaa huippujen korkeus ja laaksojen mataluus
- Maksimointiongelmaan optimaalisen ratkaisun tuottava hakualgoritmi löytää globaalin maksimin
- Lokaalit maksimit ovat korkeampia huippuja kuin yksikään niiden naapureista, mutta globaalia maksimia matalampia



Mäenkapuaminen

- Hill-climbing
- Mäenkapuamisessa toistuvasti valitaan nykytilan s seuraajista se, jolla on korkein kohdefunktion f arvo

$$\max_{s' \in S(s)} f(s')$$

- Haku pysähtyy kun tilan kaikki naapurit ovat matalammalla
- Yleensä haku siis päätyy lokaaliin maksimiin, sattumalta joskus globaaliin maksimiin
- Myös tasangot aiheuttavat ongelmia tälle ahneelle lokaalille hakumenetelmälle
- Toisaalta alkutilanteen paraneminen on usein nopeaa

- Sivuttaissiirtymät voidaan sallia kun etsintä saa edetä myös nykytilan kanssa yhtä hyviin tiloihin
- *Stokastinen* mäenkapuaminen valitsee tilannetta parantavista naapureista satunnaisesti yhden
- Naapureita voidaan tutkia satunnaisessa järjestyksessä ja valita ensimmäinen, joka on nykytilaa parempi
- Myös nämä versiot mäenkapuamisesta ovat epätäydellisiä, sillä ne voivat edelleen juuttua lokaaliin maksimiin
- *Satunnaisilla uudelleenkäynnistyksillä* voidaan varmistaa menetelmän täydellisyys
  - Mäenkapuaminen käynnistetään satunnaisesta alkutilasta, kunnes ratkaisu löytyy

Simuloitu jäähdytys

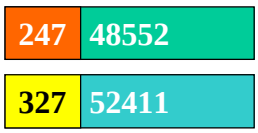
- Simulated annealing
- *Satunnaiskulku* (random walk) — siirtyminen satunnaisesti valittuun naapuriin on se parempi kuin nykytila tai ei — on täydellinen etsintämenetelmä, mutta ohjaamattomana tehoton
- Sallitaan "huonoja" siirtymiä jollain todennäköisyydellä p
- Tilannetta huonontavien siirtymien todennäköisyys pienenee eksponentiaalisesti ajan myötä ("lämpötila")
- Siirtymäkandidaatti valitaan satunnaisesti ja se hyväksytään, jos
  1. kohdefunktion arvo paranee;
  2. muuten todennäköisyydellä p
- Jos lämpötilan pieneneminen on riittävän hidasta, niin menetelmä konvergoi globaaliin optimiin tn.:llä → 1

Lokaali keilahaku

- Local beam search
- Lähdetään liikkeelle k kappaleesta satunnaisesti valittuja tiloja
- Jokaisessa askelessa kaikkien muistissa olevien tilojen seuraajat generoidaan
- Jos jokin seuraajista on maalitila, algoritmi päättyy
- Muutoin k parasta seuraajaa pidetään muistissa ja etsintä jatkuu
- Keilahaun rinnakkainen etsintä johtaa nopeasti huonojen polkujen hylkäämiseen
- Stokastisessa keilahaussa jatkoon pääsevät seuraajasolmut valitaan niiden hyvyyden mukaisella todennäköisyydellä

Geneettiset algoritmit

- Geneettiset algoritmit (GA) soveltavat perinnöllisyydestä tuttuja operaatioita hakuun
- GAn toiminta lähtee liikkeelle k satunnaisesti valitun tilan *populaatiosta* kuten keilahaku
- Nyt merkkijonoin esitettyjä tiloja kutsutaan *yksilöiksi*
- Seuraavan sukupolven populaation muodostamiseksi kaikki yksilöt evaluoidaan *kelpoisuusfunktiolla* (fitness function)
- Yksilöiden tn. päästä lisääntymään riippuu niiden kelpoisuudesta (*valinta*)
- Valittujen yksilöiden satunnaisia pareja *risteytetään* (crossover) keskenään s.e. molemmista merkkijonoista valitaan sopiva katkaisukohta ja niiden loppuosat vaihdetaan keskenään



- Muodostettujen jälkeläisten kaikki merkkipositiot ovat lopulta *mutaation* mahdollisena kohteena
- Mutaatiossa merkkejä vaihdetaan satunnaisesti toisiksi (hyvin) pienellä todennäköisyydellä
- GA:n operaatioiden toimivuus edellyttää huolellista yksilöiden koodauksen valintaa ja usein operaatioiden rajoittamista s.e. jälkeläiset ovat mielekkäitä
- Hyvin suosittu heuristinen optimointimenetelmä nykyisin, tehoton

Jatkuvat avaruudet

- Olkoon kohdeftio $f(x_1, y_1, x_2, y_2, x_3, y_3)$ kuuden jatkuva-arvoisen muuttujan funktio
- Funktion gradientti ∇f on vektori, joka kertoo jyrkimmän nousun suunnan ja suuruuden

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

- Usein gradienttia ei voi soveltaa globaalisti, mutta lokaalisti kyllä
- Jyrkimmän nousun mäenkapuamista voidaan tehdä päivittämällä nykytilaa

$$\bar{u} \leftarrow \bar{u} + \alpha \nabla f(\bar{u}),$$

missä α on "pieni vakio"

- Jos kohdefunktio ei ole derivoituva, niin gradientin suuntaa voidaan hakea empiirisesti tekemällä pieniä muutoksia kuhunkin koordinaattiarvoon
- Vakion α arvon määrääminen on keskeistä: liian pieni arvo johtaa liian moneen askeleen ja liian suuri arvo puolestaan voi johtaa maksimin hukkaamiseen
- Usein käytetty menetelmä on vakion α tuplaaminen aina siihen asti kunnes f:n arvo pienenee
- Muotoa $g(x) = 0$ olevien yhtälöiden ratkaisemiseen voidaan käyttää Newton-Raphson -menetelmää
- Ratkaisulle x lasketaan arvio Newtonin kaavalla

$$x \leftarrow x - g(x)/g'(x)$$

- Funktion f maksimoimiseksi tai minimoimiseksi tulee löytää $\bar{u}$ s.e. $\nabla f(\bar{u}) = 0$
- Soveltamalla Newtonin kaavaa
  $\bar{u} \leftarrow \bar{u} - \mathbf{H}_f^{-1}(\bar{u})\, \nabla f(\bar{u})$,
  missä $\mathbf{H}_f(\bar{u})$ on toisten derivaattojen Hess-matriisi,
  $H_{ij} = \partial^2 f / \partial x_i\, \partial x_j$
- Hess-matriisi on neliöllinen ja suuriulotteisissa avaruuksissa sen käyttäminen on kallista
- Myös jatkuva-arvoista optimointia koskevat mm. lokaalien optimien ja tasankojen ongelmat
- *Optimointi rajoitteiden vallitessa* edellyttää ratkaisulta joidenkin ehtojen täyttymistä
- Esimerkiksi *lineaarisessa ohjelmoinnissa* rajoitusehdot ovat lineaarisia epäyhtälöitä, jotka muodostavat konveksin alueen ja kohdefunktio on myös lineaarinen