

3.4 Peruutus (backtracking)

Tarkastellaan kahta esimerkkiongelmää:

Kahdeksan kuningattaren ongelma: sijoitettava 8×8 ruudun pelilaudalle 8 nappulaa siten, että millekään vaaka-, pysty- tai viistoriville ei tule kahta.

Kauppamatkustajan ongelma (TSP).

Yhteisiä piirteitä:

Osaratkaisut: alle 8 nappulan lailliset sijoitukset; syklittömät polut jotka eivät käy kaikissa kaupungeissa

Osaratkaisujen täydennökset: yhden nappulan lisääminen sallittuun ruutuun; polun jatkaminen yhdellä kaarella ilman että syntyy sykli

Lopullisen ratkaisun hyvyys: 1 tai 0 sen mukaan onko ratkaisu laillinen; polun kustannus

Kumpaankin ongelmaan voidaan etsiä ratkaisua ahneella heuristiikalla, mutta optimaalisuudesta ei ole takuuta. **Peruuttaminen** on systemaattinen tapa käydä koko ratkaisuavaruus läpi, jolloin paras ratkaisu varmasti löydetään.

Periaatealgoritmi:

```
peruuttava( $x$ : tapaus,  $y$ : osavastaus)
  if  $y$  on valmis ratkaisu then
    return  $y$ 
  else
    for kaikille  $y$ :n täydennöksille  $a_1, \dots, a_m$  do
       $y' :=$  peruuttava( $x, y \oplus a_i$ )
      if  $y'$  on valmis ratkaisu then
        return  $y'$ 
    end for
  return epäonnistui
end if
```

Tässä on ajateltu tilannetta, jossa riittää vain löytää yksi laillinen ratkaisu. Optimointitehtävässä pitää tietysti laskea **for**-silmukassa kaikki mahdolliset ratkaisut y' ja valita paras.

Käytännössä osaratkaisut pitää jotenkin järjestää, ettei samaan osaongelmaan päädytä useita teitä (esim.

$\{ \} \xrightarrow{a} \{ a \} \xrightarrow{b} \{ a, b \}$ ja $\{ \} \xrightarrow{b} \{ b \} \xrightarrow{a} \{ a, b \}$).

Kauppamatkustajan ongelman tarkemmin:

Esitetään osaratkaisu joukkona rajoituksia

- $+(a, b)$ tarkoittaa kaari (a, b) on reitillä
- $-(a, b)$ tarkoittaa kaari (a, b) ei ole reitillä

Osaratkaisu voidaan suoraan hylätä, jos sitä ei voida täydentää valmiiksi ratkaisuksi. Näin on ainakin jos osaratkaisu sisältää

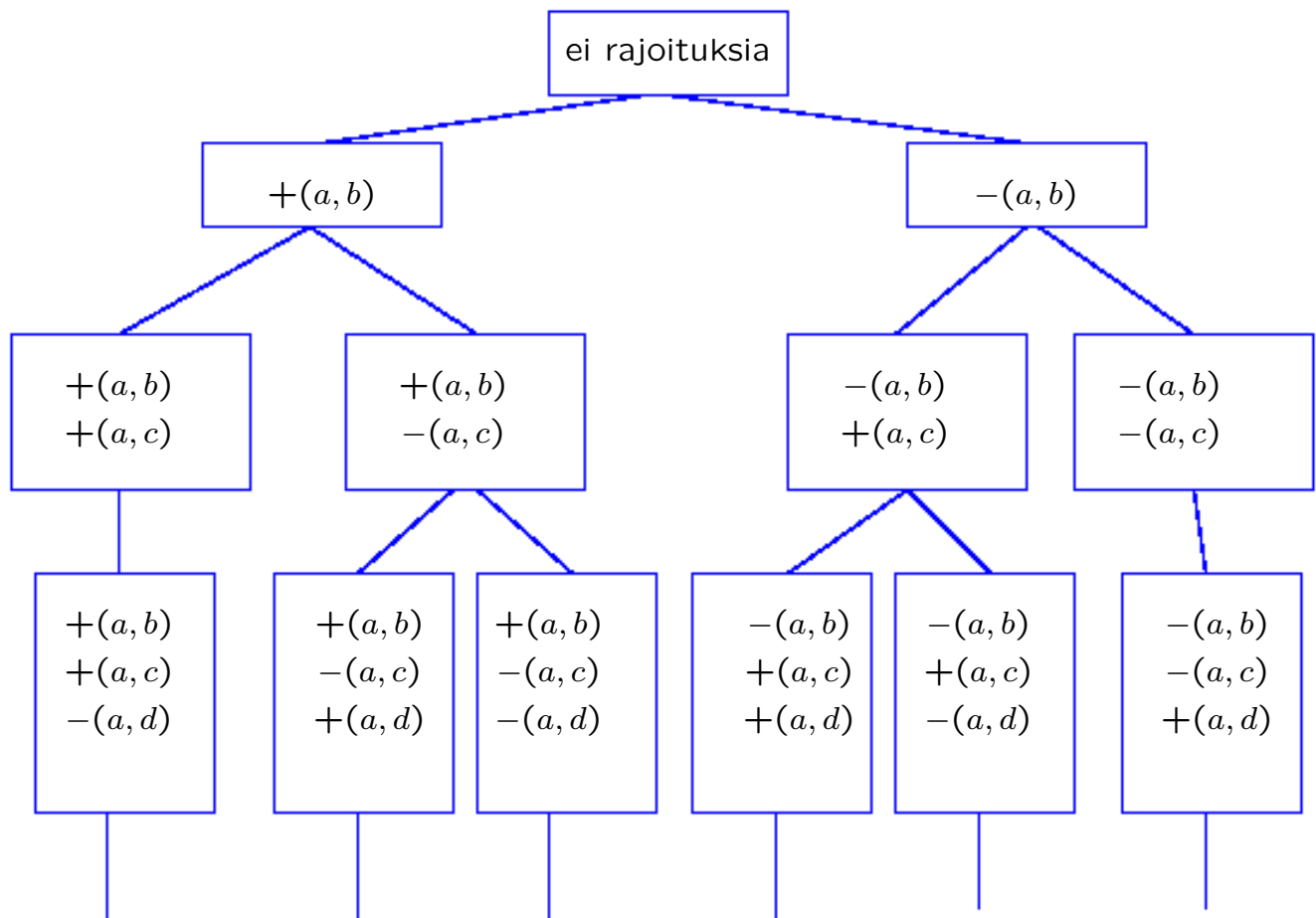
1. syklin jonka pituus on alle n tai
2. valinnan $-(a, x)$ ainakin $n - 2$ alkiolla x tai
3. valinna $+(a, x)$ ainakin 3 alkiolla x .

(Siis n on solmujen lukumäärä, ja (a, b) ja (b, a) esittävät samaa kaarta.)

Merkitään solmuja kirjaimilla a, b, c, \dots ja käsitellään kaaret leksikografisessa järjestyksessä.

Peruuttavan algoritmin etenemistä voidaan havainnollistaa **etsintäpuulla**:

<i>puu</i>	<i>etsintäongelma</i>
solmu	osaratkaisu
solmun jälkeläiset	osaratkaisun täydennökset
lehti	valmis ratkaisu



Etsintäpuun alkua kauppamatkustajan ongelmalle. Verkossa solmut a, b, c, d, e .

Etsintäpuun karsinta (branch-and-bound)

Perusidea: jos osaratkaisulle y' on aiemmin löydetty täydennös valmiiksi ratkaisuksi y jonka kustannus on c , niin ei enää tarvitse tutkia sellaisia osaratkaisuja $y' \oplus a_i$ joista tiedetään, että niiden täydennöksillä on aina vähintään kustannus c .

Rajoitusheuristiikka kauppamatkustajan ongelmaan:
Reitin (v_1, \dots, v_n) kustannus on

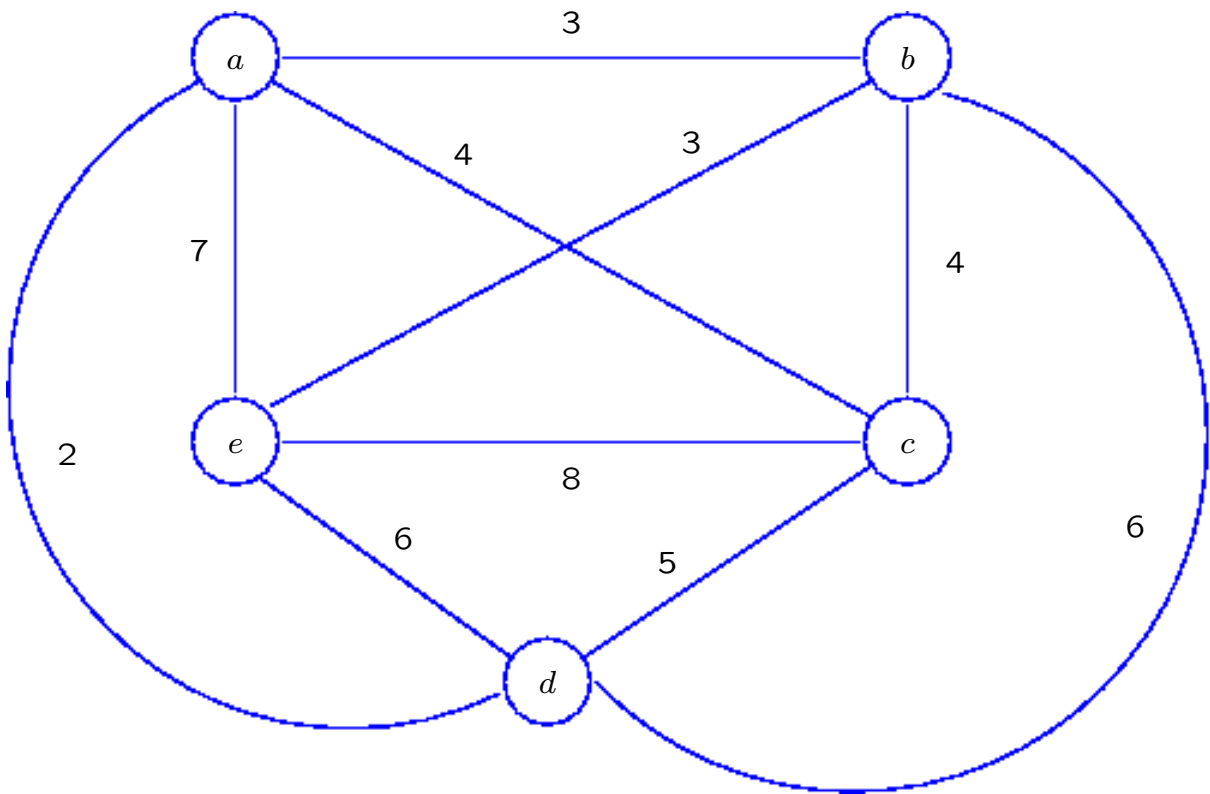
$$\begin{aligned} c &= \frac{1}{2} \sum_{i=1}^n (d(v_{i-1}, v_i) + d(v_i, v_{i+1})) \\ &\geq \frac{1}{2} \sum_{i=1}^n \min_{u \neq w} (d(u, v_i) + d(v_i, w)). \end{aligned}$$

(Merkitty $v_0 = v_n$ ja $v_{n+1} = v_1$.)

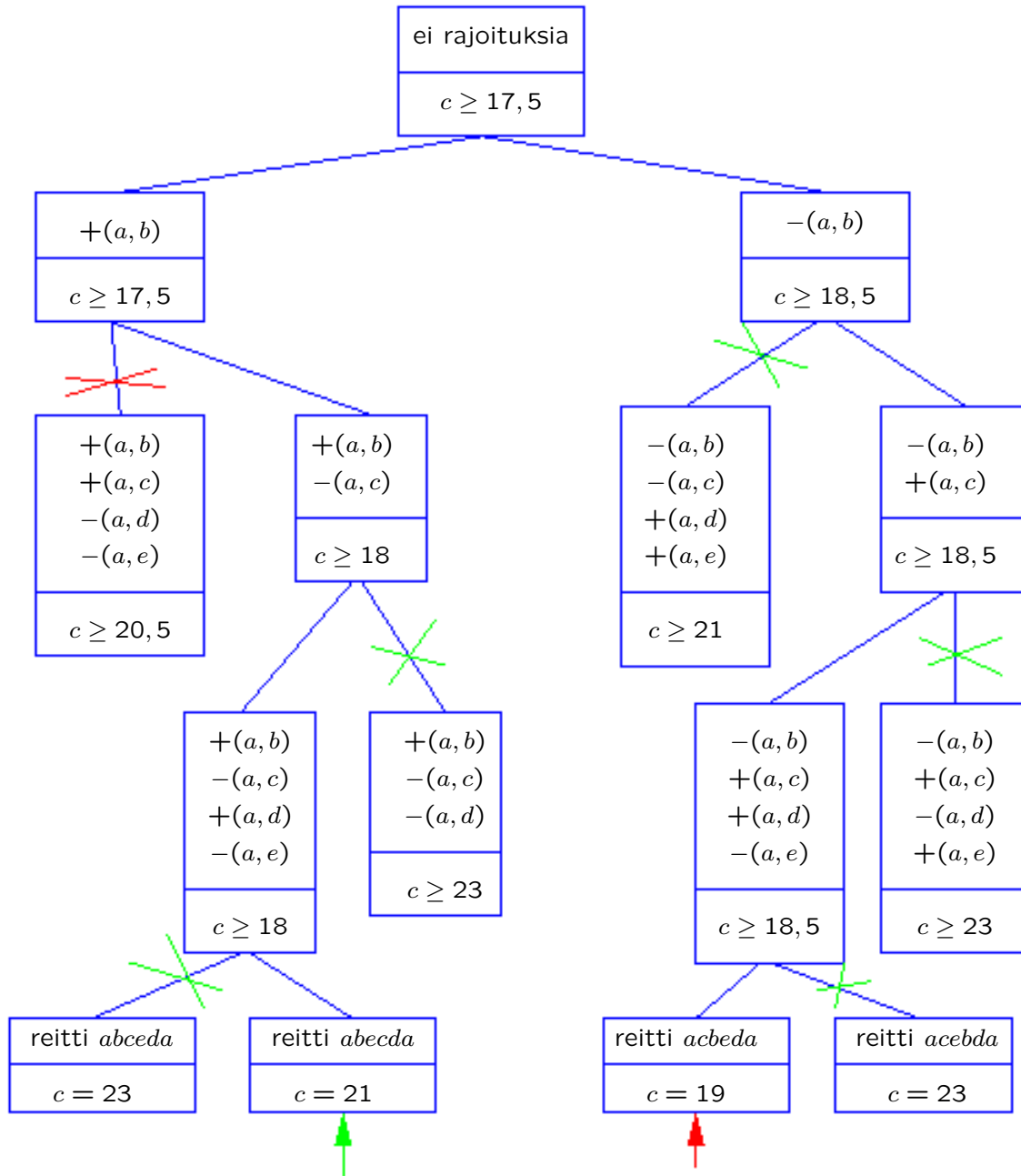
Yleisemmin jos on annettu joukko rajoituksia (eli osaratkaisu), niin rajoitukset toteuttaville reiteille (osaratkaisun täydennöksille) saadaan alaraja laskemalla ylläoleva summa rajoitusten vallitessa.

Esim. $V = \{a, b, c, d, e\}$, osaratkaisu $\{+(a, c), -(a, b)\}$:

$$\begin{aligned} c &\geq \frac{1}{2} (d(a, c) + \min \{d(a, d), d(a, e)\} \\ &\quad + \min \{d(x, b) + d(b, y) \mid x \neq y, x \neq a, y \neq a\} \\ &\quad + d(a, c) + \min \{d(b, c), d(c, d), d(c, e)\} \\ &\quad + \min \{d(x, d) + d(d, y) \mid x \neq y\} \\ &\quad + \min \{d(x, e) + d(e, y) \mid x \neq y\}). \end{aligned}$$



Esimerkki kauppamatkustajan verkosta



Branch-and-bound-ratkaisu (yksinkertaistettu)

Kommentteja branch-and-bound-menetelmästä

- usein paras (tai ainoa) vaihtoehto jos todella halutaan ratkaista tarkasti NP-kovia optimointiongelmia
- vie runsaasti suoritusaikaa
- mahdollista hyödyntää rinnakkaislaskentaa
- heuristiikkana voidaan käsitellä ensin ne alipuut, joissa laskettu alaraja on pienin (toivottavasti löytyy hyvä lehti jolla päästään karsimaan)
- karsinnassa voidaan myös käyttää jollain muulla algoritmilla ennakkoon laskettua alarajaa (approksimointialgoritmi, satunnaisalgoritmi tms.)

3.5 Paikallinen etsintä

Perusajatus: satunnaiseen alkuratkaisuun tehdään pieniä muutoksia, kunnes se ei enää parane.

Esimerkki Pienin virittävä puu painotetulle verkolle (V, E, ℓ)

1. Muodosta satunnaisesti jokin virittävä puu T .
2. Toista seuraavaa, kunnes joko löytyy puu T' jonka kustannus on pienempi kuin puun T kustannus, tai kunnes kaikki kaaret $e \in E - T$ on kokeiltu:
 - (a) Valitse $e \in E - T$.
 - (b) Aseta $T' := T \cup \{e\}$. Nyt T' sisältää yhden syklin.
 - (c) Poista T' :sta kustannukseltaan suurin kaari. Nyt T' on virittävä puu.
3. Jos löytyi kustannukseltaan pienempi T' , aseta $T := T'$ ja palaa kohtaan 2. Muuten palauta T .

Voidaan osoittaa, että tämä aina löytää pienimmän virittävän puun. Kruskalin algoritmi on kuitenkin tehokkaampi. (Tässä lopetustesti vie ajan $O(|V||E|)$.)

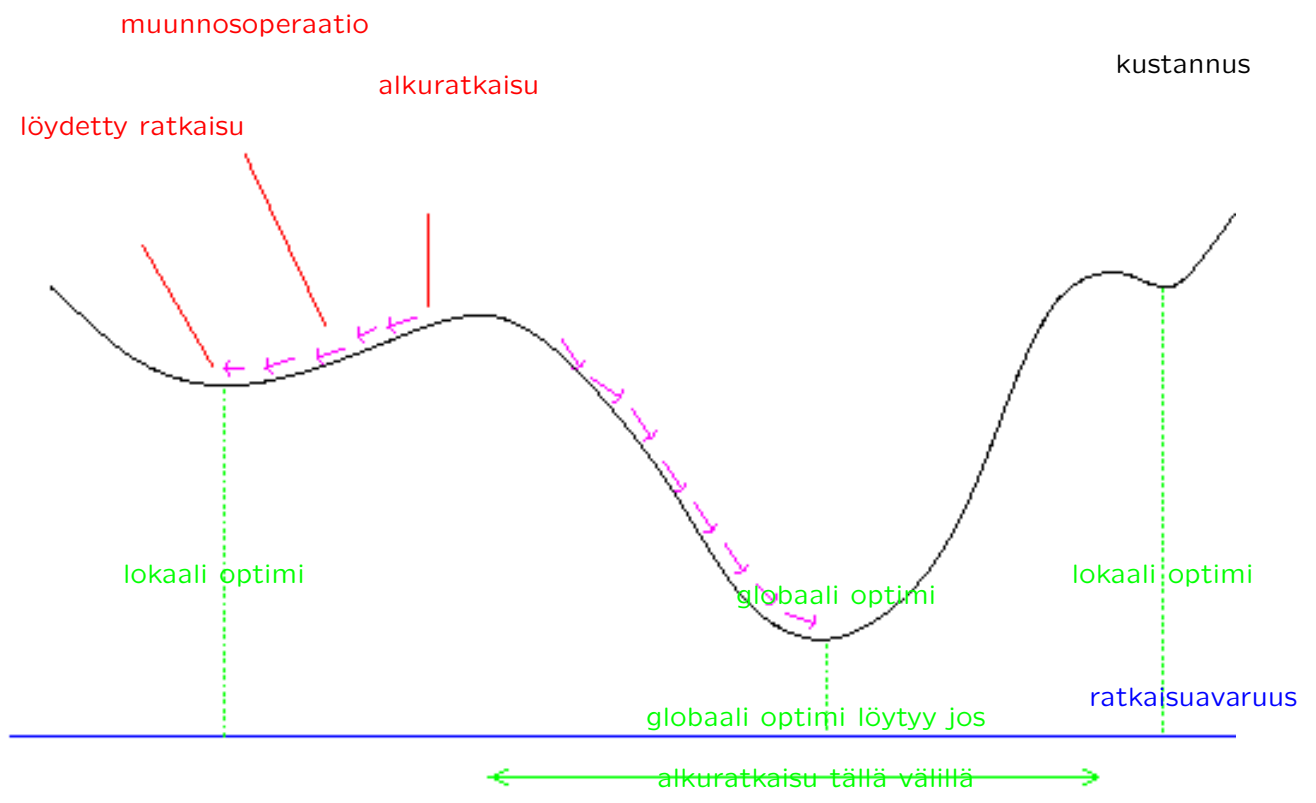
Yleisemmin: on määritelty

- sallittujen ratkaisujen joukko (esim. virittävät puut),
- kustannusfunktio (esim. puun paino) ja
- joukko muunnosoperaatioita (esim. kaariparin vaihtaminen).

Annetun sallitun ratkaisun S **naapureita** ovat ne sallitut ratkaisut S' , jotka saadaan ratkaisusta S yhdellä muunnosoperaatiolla.

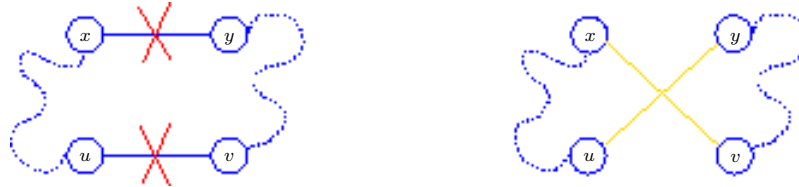
Paikallinen etsintä etenee seuraavasti:

1. $S :=$ satunnainen ratkaisu
2. Jos kustannus ratkaisulla S on ainakin yhtä pieni kuin parhaalla sen naapurilla, palauta S .
3. Muuten aseta $S := S'$ missä S' on jokin kustannukseltaan pienempi naapuri ja jatka kohdasta 2.



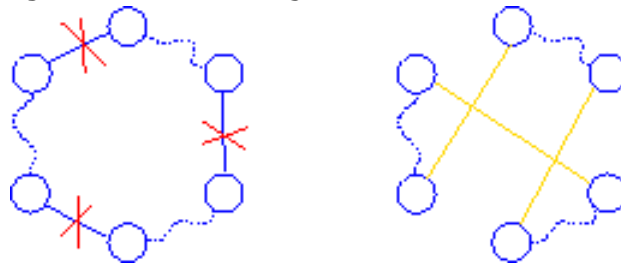
Paikallinen etsintä: periaatepiirros.

Esimerkki TSP, muunnosoperaationa 2-vaihto (2-opting, Lin ja Kernighan):



- valitse reitiltä kaksi kaarta (x, y) ja (u, v) ja poista ne
- oletetaan että jäljelle jää polut $x \rightsquigarrow u$ ja $y \rightsquigarrow v$
- lisää kaaret (x, v) ja (y, u)

Tällä on yleistys 3-vaihto jne.



Käytännössä 3-vaihto on parhaita TSP-algoritmeja. Se voi kuitenkin joutua lokaaliin minimiin ja vie pahimmassa tapauksessa eksponentiaalisen ajan

Kommentteja paikallisesta etsinnästä:

- toiminta riippuu vahvasti naapurustorakenteen (eli muunnosoperaatioiden) valinnasta
- usein vaihtokauppa: monimutkaiset operaatiot hidastavat suoritusta mutta antavat parempia lopputuloksia
- herkkä alkuratkaisun valinnalle, voi jäädä lokaaliin minimiin
- voidaan kokeilla useilla alkuratkaisuilla (rinnakkaistuu helposti)
- kehittyneempiä versioita: tabu-etsintä, simuloitu jäähdytys

Tabuetsintä: paikallisessa etsinnässä osa naapurustosta on "tabu" eikä sinne saa mennä.

1. Valitse satunnainen alkuratkaisu S . Aseta $\text{tabu} := \{S\}$ ja $\text{paras} := S$.
2. Olkoon N ratkaisun S naapurusto. Valitse $S' \in N - \text{tabu}$ joka on kustannukseltaan pienin.
3. Jos $\text{kustannus}(S') < \text{kustannus}(\text{paras})$, aseta $\text{paras} := S'$.
4. Jos *lopetusehto* toteutuu, palauta S . Muuten aseta $S := S'$, päivitä tabu ja jatka kohdasta 2.

Asettamalla aina $\text{tabu} = \{S\}$ ja ottamalla lopetusehdoksi $\text{kustannus}(S') \geq \text{kustannus}(S)$ saadaan paikallinen etsintä.

Mielenkiintoisempi mahdollisuus on pitää tabu-joukossa k viimeksi kokeiltua ratkaisua jollain k ; estää toistoja.

Toinen mahdollisuus on kieltää ratkaisun sellaisten osien muuttaminen, joita on äskettäin muutettu (juuri lisättyjä kaaria ei poisteta jne.).

Simuloitu jäähdytys (simulated annealing)

Lisätään paikalliseen etsintään satunnaisuutta paikallisten minimien välttämiseksi.

Lämpötilaparametri T säätelee, kuinka suurella todennäköisyydellä sallitaan siirtymä huonompaan ratkaisuun. (Vrt. termodynamiikka: ratkaisun kustannus \simeq tilan energia)

Seuraavassa i ja j indeksoivat ratkaisuja, S_i on ratkaisun i naapurien joukko ja c on kustannusfunktio, jota minimoidaan.

Lämpötilamuuttuja on T , ja T_{loppu} määrää lopetusehdon. Funktio *jäähdytä* säätelee iteraatioiden lukumäärää; palaamme pian sen yksityiskohtiin.

Simuloitu jäähdytys:

```
( $i, T, L$ ) := alkuarvot
while  $T > T_{\text{loppu}}$  do
  for  $k := 1$  to  $L$  do
    valitse satunnainen  $j \in S_i$ 
     $\Delta := c(j) - c(i)$ 
    if  $\Delta \leq 0$  then  $i := j$ 
    elsif  $\text{random}([0, 1]) \leq \exp(-\Delta/T)$ 
    then  $i := j$ 
    end if
  end for
  ( $T, L$ ) := jäähdytä( $T$ )
end while
return  $i$ 
```

Lämpötila T alustetaan johonkin korkeaan arvoon ja sitä lasketaan pikku hiljaa, kunnes ollaan halutussa loppulämpötilassa. Aina jäähtytysaskelen jälkeen suoritetaan sisempää silmukkaa L kertaa, jotta systeemi ehtii stabiloitua. (Huom. L riippuu lämpötilasta.)

Siirtyminen tilasta i kustannukseltaan suurempaan tilaan j sallitaan todennäköisyydellä, joka pienenee eksponentiaalisesti kun kustannusero Δ kasvaa tai lämpötila T pienenee.

Aluksi lämpötila on korkea ja algoritmi käyttäytyy hyvin satunnaisesti. Tämä antaa sille tilaisuuden vieraillla laajassa osassa ratkaisuavaruutta ja löytää globaalisti hyviä alueita.

Kun lämpötila pienenee, algoritmi muuttuu deterministisemmäksi ja "jähmettyy" kohti jotain lokaalia minimiä.

Voidaan osoittaa, että riittävän hitaalla jäähtytyksellä algoritmi päättyy globaaliin minimiin todennäköisyydellä 1.

Tarkastelemme ensin algoritmin toimintaa vakiolämpötilassa.

Oletamme jatkossa, että naapurustorakenne on **symmetrinen** eli $j \in S_i$ joss $i \in S_j$.

Naapurustorakenne on **yhtenäinen** jos sitä esittävä suuntaamaton verkko on, ts. mikä tahansa ratkaisu voidaan muuntaa miksi tahansa toiseksi ratkaisuksi soveltamalla jonoa muunto-operaatioita.

Naapurustorakenne on **tasa-asteinen** jos $|S_i| = |S_j|$ kaikilla i, j .

Oletetaan, että c ei ole vakio; muuten ongelma on triviaali.

Lause Olkoon $P_T(i, t)$ todennäköisyys, että ajanhetkellä t algoritmi tarkastelee ratkaisua i kun lämpötila on koko ajan T . Jos naapurusto on symmetrinen, yhtenäinen ja tasa-asteinen, niin

$$\lim_{t \rightarrow \infty} P_T(i, t) = \frac{1}{Z_T} \exp\left(-\frac{c(i)}{T}\right)$$

missä $Z_T = \sum_j \exp(-c(j)/T)$ on normituskerroin.

Todistus Osoitetaan, että algoritmia kuvaavalla Markovin ketjulla on tasapainojakauma, joka saadaan ylläolevasta lausekkeesta.

Markovian ketjun yhtenäisyys seuraa suoraan oletuksista.

Kun c ei ole vakio, ainakin joillain ratkaisuilla i siirtymällä $i \rightarrow i$ on nolasta poikkeava todennäköisyys. Siis Markovian ketju on sykliiton, ja sillä on tasapainojakauma.

Olkoon $P_T(i)$ ratkaisun i todennäköisyys tasapainojakaumassa. Selvästi $P_T(i) = \lim_t P_T(i, t)$.

Merkitään

$$Q(i) = \frac{1}{Z_T} \exp\left(-\frac{c(i)}{T}\right).$$

Pitää osoittaa, että Q on tasapainojakauma. Tunnetusti tähän riittää, että kaikilla i, j pätee

$$Q(j)p_{ji} = Q(i)p_{ij}$$

missä p_{ij} on todennäköisyys siirtyä $i \rightarrow j$. Koska ehto on symmetrinen, riittää tarkastella tapausta $c(i) \geq c(j)$. Tällöin

$$\begin{aligned} p_{ij} &= \frac{1}{k} \\ p_{ji} &= \frac{1}{k} \exp\left(-\frac{c(i) - c(j)}{T}\right) \end{aligned}$$

missä $k = |S_i| = |S_j|$. Väite seuraa suoraan laskemalla. \square

Edellinen lause ei kerro algoritmin dynaamisesta käyttäytymisestä kun lämpötilaa muutetaan.

Olkoon V kaikkien ratkaisujen joukko ja V^* kustannukseltaan minimaalisten ratkaisujen joukko. Olkoon $\ell(i, j)$ lyhimmän polun pituus solmusta i solmuun j naapurustoverkossa.

Merkitään

$$\Delta_{\max} = \max \{ c(i) - c(j) \mid j \in S_i \}$$

ja

$$L_0 = \max_{i \in V} \min_{j \in V^*} \ell(i, j).$$

Lause (Geman ja Geman 1984) Olkoon naapurustorakenne symmetrinen, yhtenäinen ja tasa-asteinen. Jos lämpötilan lasku tapahtuu siten, että vaiheessa m lämpötila on

$$T = \tau_m \geq \frac{(L_0 + 1)\Delta_{\max}}{\ln(m + 2)}$$

ja sisempää silmukkaa toistetaan aina $L = L_0$ kertaa, niin algoritmi konvergoi kohti jakaumaa P^* missä

$$P^*(i) = \begin{cases} 1/|V^*| & \text{jos } i \in V^* \\ 0 & \text{muuten.} \end{cases}$$

□

Kommentteja simuloidusta jäähdytyksestä

- yleiskäyttöinen menetelmä, joka tyypillisesti antaa hyviä tuloksia mutta vaatii paljon laskentaa.
- naapurustorakenteen valinta on ilmeisen tärkeää
- jäähdytysaikataululla on usein suuri vaikutus
- teorian antamat jäähdytysaikataulut ovat pessimistisiä ja käytännössä liian hitaita
- käytännössä tyypillisesti $\tau_{m+1} = \alpha\tau_m$ missä $0,8 \leq \alpha \leq 0,99$