

## 7. Satunnaisalgoritmit (randomized algorithms)

Satunnaisuudella on laskentaongelmien ratkaisemisessa moninaisia käyttötapoja. Tässä tarkastellaan lähinnä perinteisten algoritmien nopeuttamista, ja sitäkin suhteellisen pinnallisesti.

Tämän luvun jälkeen opiskelija

- omaa yleiskuvan satunnaisuuden käyttötapoista,
- osaa luokitella satunnaisalgoritmeja niiden laatukriteerien mukaan (esim. Monte Carlo vs. Las Vegas),
- tuntee edustavan joukon esimerkkejä satunnaisalgoritmien suunnittelutekniikoista (Sherwood-algoritmit, otanta, sormenjälkitekniikka)
- osaa soveltaa odotusarvojen ja virherajojen arvioimisen perustekniikoita (odotusarvon lineaarisuus, binomijakauma),

Aiheesta on luennoitu erikoiskurssi viimeksi keväällä 2003. Hyvä oppikirja on Motwani ja Raghavan: Randomized algorithms.

## 7.1 Peruskäsitteitä

Satunnaisuutta voidaan käyttää useisiin eri tarkoituksiin, kuten

- symmetrian rikkominen (hajautetut järjestelmät)
- vastustajan hämääminen (pelit, salaus)
- otanta
- tavallisten laskentaongelmien nopea ratkaiseminen "arvaamalla"
- monimutkaisten algoritmien yksinkertaistaminen (esim. aiemmin esitetty mediaanialgoritmi)
- yleiskäyttöiset optimointimenetelmät (simuloitu jäähtytys, geneettiset algoritmit)
- uudet laskentaparadigmat (kvanttilaskenta jne.)

Mielenkiintoinen periaatteellinen kysymys: mitä satunnaisuudella todella voidaan voittaa?

- Toisinaan satunnaisalgoritmeja saadaan "derandomisoiduksi".
- Ehkä kuitenkin joissain tilanteissa satunnaisuudesta on aidosti hyötyä?
- Jos näin on, mitä on ajateltava pseudosatunnaislukugeneraattoreista, joihin käytännön sovellukset yleensä perustuvat?

Yleisesti uskotaan, että perinteinen satunnaisuus ei riitä NP-kovien ongelmien ratkaisemiseen polynomisessa ajassa.

Kvanttilaskennan asema on epäselvempi.

Ensimmäisenä johdattellevana esimerkkinä tarkastellaan seuraavaa satunnaistettua pikajärjestämisalgoritmia:

**RandQS( $A$ ):**

**if**  $|A| > 1$  **then**

$p := \text{random}(1..|A|)$

$\text{pivot} := A[p]$

Vertaa alkiota  $\text{pivot}$  kaikkiin taulukon  $A$  alkioihin.

Muodosta taulukko  $A_1$

alkiota  $\text{pivot}$  pienemmistä ja

$A_2$  alkiota  $\text{pivot}$  suuremmista alkioista.

RandQS( $A_1$ )

RandQS( $A_2$ )

**return**  $A_1 \parallel [\text{pivot}] \parallel A_2$

**else return**  $A$

Kutsu RandQS( $A$ ) järjestää taulukon  $A$ . (Oletamme yksinkertaisuuden vuoksi, että taulukon luvut ovat erisuuria.)

Funktio random( $i..j$ ) palauttaa satunnaisen luvun joukosta  $\{i, i + 1, \dots, j\}$  siten, että jokaisella luvulla on sama todennäköisyys ja eri kutsukertojen tulokset ovat toisistaan riippumattomia. Merkintä  $A \parallel B$  tarkoittaa taulukoiden  $A$  ja  $B$  yhdistämistä peräkkäin.

Olkoon  $C$  taulukko, jossa on  $n$  lukua, pienimmästä suurimpaan  $x_1, \dots, x_n$ . Analysoidaan kutsua RandQS( $C$ ).

Olkoon  $X_{rs}$  satunnaismuuttuja, joka kertoo kuinka monta kertaa alkioita  $x_r$  ja  $x_t$  verrattiin toisiinsa kutsun  $\text{RandQS}(C)$  suorituksen aikana.

Tietyn suorituskerran kokonaisaikavaativuus on selvästi verrannollinen vertailujen lukumäärään

$$\sum_{r=1}^n \sum_{s=1}^{r-1} X_{rs}.$$

Tarkastellaan tämän odotusarvoa

$$E \left[ \sum_{r=1}^n \sum_{s=1}^{r-1} X_{rs} \right] = \sum_{r=1}^n \sum_{s=1}^{r-1} E[X_{rs}]$$

missä on käytetty hyväksi odotusarvon lineaarisuutta.

Koska  $X_{rs}$  on joko 1 tai 0, saadaan

$$E[X_{rs}] = 0 \cdot (1 - p_{rs}) + 1 \cdot p_{rs} = p_{rs}$$

missä  $p_{rs}$  on todennäköisyys, että alkioita  $x_r$  ja  $x_s$  verrataan. Haluamme siis arvioida summaa

$$\sum_{r=1}^n \sum_{s=1}^{r-1} p_{rs}.$$

Määritellään binääripuu  $T(A)$  kaikille niille  $A$ , joilla kutsun  $\text{RandQS}(C)$  suorituksen aikana tehtiin rekursiivinen kutsu  $\text{RandQS}(A)$ . Puussa on  $n$  solmua, joihin sijoitellaan luvut  $x_1, \dots, x_n$ .

Määritelmä on rekursiivinen.

Jos  $|A| = 0$ , niin  $T(A)$  on tyhjä puu.

Jos  $|A| = 1$ , niin  $T(A)$  on lehti, jossa on taulukon  $A$  ainoa alkio.

Muuten olkoon  $pivot$ ,  $A_1$  ja  $A_2$  kuten algoritmissa. Nyt  $T(A)$  on puu, jonka juuressa on luku  $pivot$ , vasempana alipuuna  $T(A_1)$  ja oikeana  $T(A_2)$ .

Siis puun  $T(A)$  vasemmassa alipuussa ovat juurta pienemmät ja oikeassa juurta suuremmat taulukon alkio. Järjestys  $x_1, \dots, x_n$  vastaa puun  $T(C)$  läpikäyntiä sisäjärjestyksessä.

Algoritmin ja puun  $T(A)$  määritelmistä seuraa, että kutsun  $\text{RandQS}(A)$  suorituksen aikana puun juurta verrataan tasan kerran kuhunkin muuhun puun alkioon. Siis  $X_{rs} = 1$  jos ja vain jos joko  $x_r$  on alkion  $x_s$  jälkeläinen puussa  $T(C)$  tai kääntäen.

Olkoon  $\pi$  taulukon  $C$  järjestys, joka saadaan luettelemalla ensin puun  $T(C)$  juuri, sitten juuren lapset vasemmalta oikealle, sitten lapsenlapset vasemmalta oikealle jne.

Olkoon  $s < r$ . Siis  $x_q$  on solmujen  $x_r$  ja  $x_s$  lähin yhteinen esivanhempi, jos se on järjestyksessä  $\pi$  ensimmäinen jolla  $x_s \leq x_q \leq x_r$ . Tämä on yhtäpitävää sen kanssa, että  $x_q$  päättyy *pivot*-alkioksi aiemmalla rekursiotasolla kuin mikään muu alkio joukosta  $\{x_s, x_{s+1}, \dots, x_r\}$ .

Siis  $X_{rs} = 1$  jos ensimmäinen joukosta  $\{x_s, x_{s+1}, \dots, x_r\}$  valittava *pivot* on joko  $x_r$  tai  $x_s$ . Koska *pivot* valitaan satunnaisesti, tämän todennäköisyys on  $2/(r - s + 1)$ .

Saadaan

$$\begin{aligned} \sum_{r=1}^n \sum_{s=1}^{r-1} p_{rs} &= \sum_{r=1}^n \sum_{s=1}^{r-1} \frac{2}{r-s+1} \\ &= \sum_{r=1}^n \sum_{k=1}^{r-1} \frac{2}{k+1} \\ &\leq 2 \sum_{r=1}^n \sum_{k=1}^n \frac{1}{k} \\ &= 2nH_n. \end{aligned}$$

Koska  $H_n \sim \ln n$ , saadaan

**Lause** Algoritmi RandQS tekee odotusarvoisesti  $O(n \log n)$  vertailua  $n$ -alkioisen taulukon järjestämisessä.  $\square$

## Huomioita:

- Deterministisessä pikajärjestämisessä aikavaativuus on  $O(n \log n)$  keskimäärin, kun oletetaan syötteen eri järjestyksen yhtä todennäköisiksi. Tässä odotusarvo on algoritmin sisäisten "rahanheittojen" yli; ei tarvita oletuksia syötteestä.
- Jos käytettävissä tosiaan on "rahanheittoja" eli (pseudo)satunnaisia bittejä, satunnaislukujen generoiminen mielivaltaisesta joukosta  $\{1, \dots, n\}$  on hieman epätriviaalia (mutta ei tässä mikään varsinainen ongelma).
- Tarkemmalla analyysillä voidaan osoittaa, että **suurella todennäköisyydellä** suoritus on  $O(n \log n)$ . Toisin sanoen suuren poikkeamat ovat epätodennäköisiä, mikä usein on tärkeää.

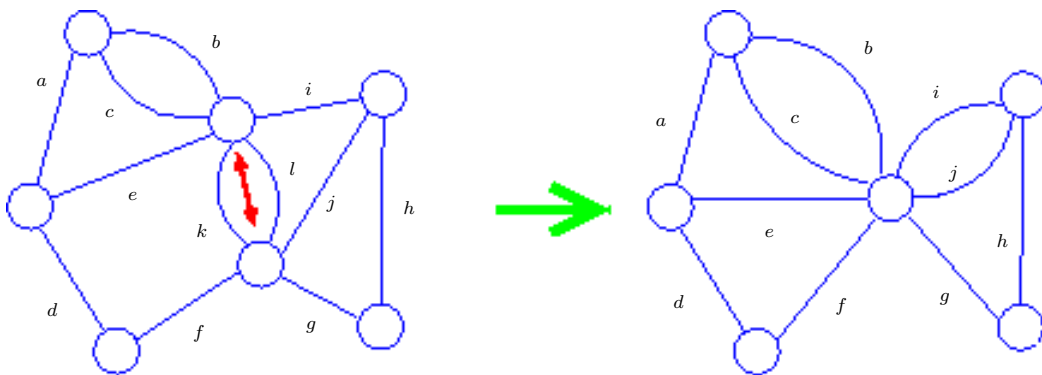
Toisena johdantoexamplesimerkkinä tarkastellaan verkon **minimileikkauksen** määrittämistä. On järkevää ratkaista ongelma yleisemmin moniverkoille  $G = (V, E)$  missä siis  $E \subseteq V \times V$  on monijoukko (multiset, bag; sama kaari saa esiintyä monta kertaa). Emme jatkossa toista etuliitettä "moni". Jatkossa  $n = |V|$  ja  $m = |E|$ .

Kaarijoukko  $C \subseteq E$  on **leikkaus**, jos verkko  $(V, E - C)$  on epäyhtenäinen. Leikkaus  $C$  on **minimileikkaus**, jos kaikilla leikkauksilla  $C'$  pätee  $|C'| \geq |C|$ .

Aiemmin todetun perusteella (s. 290–291) minimileikkausongelma palautuu maksimivuo-ongelmaan, jos on lisäksi annettu solmut  $s$  ja  $t$  joiden tiedetään olevan minimileikkauksen eri puolilla. Jos tällaisia solmuja ei tunneta, ongelma voidaan silti ratkaista etsimällä  $|V|$  maksimivuota (harjoitus 12, tehtävä 2). Itse asiassa ongelma voidaan ratkaista verkkovuota käyttämällä ajassa  $O(nm \log(n^2/m))$ .

Esitämme seuraavaksi periaatetasolla yksinkertaisen satunnaisalgoritmin ongelmalle. Idea tarkentamalla saadaan satunnaisalgoritmi, joka *suurella todennäköisyydellä* löytää minimileikkauksen ajassa  $O(n^2(\log n)^c)$  jollain vakiolla  $c$ . (Palaamme pian siihen, mitä "suurella todennäköisyydellä" tarkalleen tarkoittaa.)

Kaaren  $e = (u, v) \in E$  **kutistaminen** (contraction) tarkoittaa solmujen  $u$  ja  $v$  liittämistä yhdeksi uudeksi solmuksi. Kaari  $(u, v)$  häviää ja muuten solmuihin  $u$  ja  $v$  tulevat kaaret korvautuvat uuteen solmuun liittyvillä kaarilla.



Esimerkki kutistuksesta; selvyiden vuoksi kaaret nimetty. Kutistettavana jompi kumpi kaarista  $k$  tai  $l$ .

Jos verkko  $G'$  on saatu kutistuksella verkosta  $G$ , ja  $C$  on leikkaus verkossa  $G'$ , niin  $C$  on selvästi leikkaus myös verkossa  $G$  (kun ajatellaan kaarten "identiteetit" säilytetyksi ylläolevan esimerkin tapaan).

Siis erityisesti **kutistuksessa minimileikkauksen koko ei pienene**. Minimileikkauksen koko voi kasvaa, jos kutistettavana on minimileikkaukseen kuuluva kaari.

Saadaan seuraava periaatealgoritmi, joka löytää verkossa  $G = (V, E)$  jonkin leikkauksen:

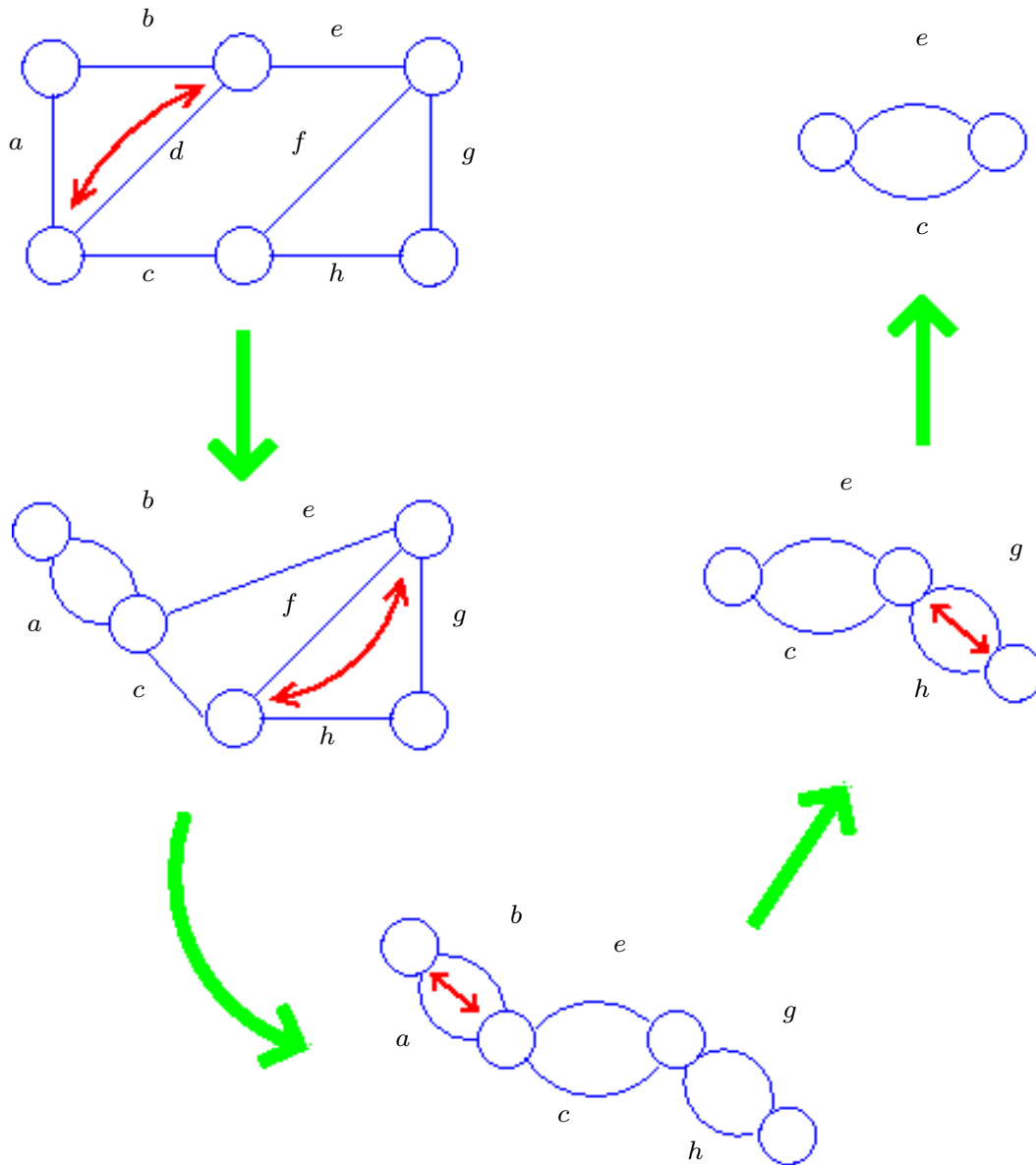
1. Valitse satunnainen kaari  $e \in E$  ja kutista se.
2. Jos  $|V| > 2$ , palaa kohtaan 1.
3. Nyt  $E$  on leikkaus.

Tässä siis taas oletetaan, että kaarilla on "identiteetti" josta pidetään kirjaa kutistuksissa. Lopuksi  $V = \{a, b\}$  joillain  $a, b$ , ja kaikki kaaret ovat solmujen  $a$  ja  $b$  välillä. Jos näiden kaarten vastinkaaret poistetaan alkuperäisestä verkosta, syntyy kaksi erillistä komponenttia, jotka koostuvat solmuun  $a$  päätyvistä ja solmuun  $b$  päätyvistä solmuista. Siis lopuksi  $E$  todella on leikkaus.

Olkoon  $C^*$  jokin alkuperäisen verkon  $G$  minimileikkaus, ja  $k = |C^*|$ .

Jos kutistettavaksi ei koskaan valita joukon  $C^*$  kaarta, niin lopuksi  $E = C^*$ . Tässä vaiheessa verkossa voi olla jäljellä vain kaksi solmua, sillä muuten jokin leikkauksen  $C^*$  aito osajoukko olisi leikkaus. Siis algoritmi löysi minimileikkauksen.

Analysoidaan siis todennäköisyyttä, että kaikki satunnaiset valinnat osuvat joukon  $C^*$  ulkopuolelle.



Eräs minimileikkausalgorithmien suoritus. Kutistettavat kaaret valittu sopivasti ( $d, f, a, h$ ) niin, että löytyy minimileikkaus  $\{e, c\}$ .

Kiinnitetään jokin minimileikkaus  $C^*$ ,  $|C^*| = k$ . Tällöin verkon jokaisen solmun  $v$  asteluku on ainakin  $k$ , koska muuten solmuun  $v$  päättyvät kaaret muodostaisivat pienemmän leikkauksen.

Lisäksi missä tahansa algoritmin suoritusvaiheessa, **jos** ei vielä ole kutistettu mitään joukkoon  $C^*$  kuuluvaa kaarta, niin  $C^*$  on edelleen minimileikkaus ja jokaisen solmun asteluku on edelleen ainakin  $k$ .

Oletetaan, että vaiheissa  $1, \dots, i$  on kutistettu vain joukkoon  $C^*$  kuulumattomia kaaria. Koska solmuja on jäljellä  $n - i$ , niin kaaria on jäljellä ainakin  $k(n - i)/2$ . Todennäköisyys, että seuraava valinta osuu joukkoon  $C^*$ , on korkeintaan  $k/(k(n - i)/2) = 2/(n - i)$ .

Siis todennäköisyys osua joukon  $C^*$  ulkopuolelle vaiheessa  $i + 1$ , jos aina ennenkin on osuttu, on ainakin  $1 - 2/(n - i)$ . Todennäköisyys pysyä joukon  $C^*$  ulkopuolella kaikissa vaiheissa  $1, \dots, n - 2$  on ainakin

$$\begin{aligned} \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) &= \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} \\ &= \frac{(n-2)(n-3)\dots\cdot 2\cdot 1}{n(n-1)\dots\cdot 4\cdot 3} \\ &= \frac{2}{n(n-1)}. \end{aligned}$$

Olemme siis osoittaneet, että ainakin todennäköisyydellä  $2/(n(n-1)) > 2/n^2$  mitään joukon  $C^*$  kaarta ei valita, jolloin algoritmi palauttaa minimileikkauksen  $C^*$ .

Tämä onnistumistodennäköisyys on tietysti häviävän pieni vähänkään suuremmilla  $n$ .

Menetellään nyt niin, että toistetaan algoritmia  $n^2/2$  kertaa, ja valitaan löydetyistä leikkauksista pienin. Jos ei löydetty minimileikkausta, niin *kaikki* suorituskerrat epäonnistuivat, minkä todennäköisyys on korkeintaan

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < e^{-1}.$$

Tämä todennäköisyys saadaan mielivaltaisen pieneksi toistamalla algoritmia riittävän monta kertaa. Palaamme pian tämän tyyppisen tilanteen tarkempaan analyysiin.

## 7.2 Satunnaisalgoritmien luokittelua

Hyvällä onnella tietysti kaikki satunnaisalgoritmit antavat hyvän ratkaisun. Algoritmeja voidaan luokitella sen mukaan, mitä yleisiä toimivuustakuita niillä on:

**Las Vegas -algoritmit:** vastaus on aina oikein, mutta pienellä todennäköisyydellä suoritus voi kestää kauan

**Monte Carlo -algoritmit:** vastaus saa olla väärä pienellä todennäköisyydellä

**Satunnaiset approksimointialgoritmit:** suurella todennäköisyydellä vastaus on ainakin suunnilleen oikein

Edellä esitetyistä RandQS oli tyyppiä Las Vegas ja minimileikkaus tyyppiä Monte Carlo.

Nimitys "Las Vegas" on melko tuore [Babai 1979]. Perinteisesti kaikenlaisia satunnaisalgoritmeja, kuten numeerisia approksimointialgoritmeja, on nimitetty Monte Carlo -algoritmeiksi.

Las Vegas -nimikettä käytetään toisinaan sellaisista algoritmeista, jotka toimivat aina nopeasti eivätkä koskaan anna väärää vastausta, mutta toisinaan vastaavat "en tiedä". Tämä on oleellisesti sama käsite kuin edellä esitetty.