Sormenjälkimenetelmät

Matti Risteli
mristeli@niksula.hut.fi

Seminaariesitelmä 23.4.2008
T-106.5800 Satunnaisalgoritmit
Tietotekniikan laitos
Teknillinen korkeakoulu

Tiivistelmä

Sormenjälkimenetelmät ovat satunnaisuutta hyväksikäyttäviä algoritmeja, joilla pyritään ratkaisemaan ns. ekvivalenssiongelmia. Nämä ovat ongelmia, joissa tietojoukkojen A ja B sisällön yhtäläisyyttä halutaan tutkia. Sormenjälkimenetelmiä hyödyntävät algoritmit voivat olla sekä Monte Carlo että Las Vegas -tyyppisiä. Satunnaisalgoritmeilla voidaan pienentää tietojoukkojen vertailussa tarvittavan tiedon määrää esim. tilanteissa joissa jokaisen tavun vertailu ei ole soveltuvaa(suuri tietomäärä hitaan yhteyden päässä toisistaan). Satunnaiset sormenjälkimenetelmät perustuvat todistajien runsaus -paradigmaan ja niiden virhetodennäköisyyttä voidaan parantaa helposti.

1 Johdanto

Tämä seminaarityö käsittelee satunnaisuuteen perustuvia sormenjälkimenetelmiä. Sormenjälkimenetelmillä pyritään ratkaisemaan ns. ekvivalenssiongelmia, joissa kahden tietojoukon yhtäläisyys halutaan selvittää tilanteissa joissa niiden suora vertailu tai kryptografisten tiivisteiden laskeminen ei ole käytännöllistä esimerkiksi suuren datamäärän vuoksi. Tällöin tietojoukkojen suoran vertailun sijaan verrataan niiden sormenjälkiä.

Sormenjäljet ovat epätarkkoja kuvauksia, joilla alkuperäinen tietojoukko esitetään pienemmässä tilassa. Tällä yhtäältä saavutetaan suorituskykyetua mutta toisaalta myös mahdollistetaan virheelliset päätelmät. Sormenjälkimenetelmät voidaan nähdä yleisesti todistajien runsaus -paradigman erityistapauksena, jonka virhetodennäköisyyttä voidaan pienentää amplifioimalla. Sormenjälkimenetelmiä voidaan käyttää hyväksi sekä Las Vegas että Monte Carlo -algoritmeissa.

Teksti perustuu Juraj Hromcovicin kirjaan “Design and Analysis of Randomized Algorithms”[1] ja mukailee pääpiirteissään kirjan Fingerprinting-luvun sisältöä. Aluksi esittelen mitä ekvivalenssiongelmalla tarkoitetaan ja miten sormenjälkimenetelmät periaatteellisella tasolla toimivat. Sen jälkeen käydään läpi sekä Las Vegas että Monte Carlo -algoritmejä käyttävät esimerkkiongelmat ja lopuksi niiden käytännön sovelluksia.

2 Ekvivalenssiongelmista

Ekvivalenssiongelma on ongelma, joka muodostuu kahdesta tietojoukosta, joiden yhtäläisyys halutaan selvittää. Yksinkertaisilla tietojoukoilla tämä voidaan tehdä helposti vertailemalla tietoja toisiinsa suoraan, mutta usein se ei ole soveltuvaa joko suuren tietomäärän tai vertailun vaikeuden aiheuttamien suorituskykyongelmien vuoksi.

Yksinkertainen esimerkki tällaisesta ongelmasta on polynomien ekvivalenssi ts. miten ratkaista onko esimerkiksi polynomit $(x-1)(x^2+4)$ ja x^3+4x-4 toistensa eri esitysmuotoja. Normaali lähestymistapa olisi tietysti johtaa molemmat polynomit jonkinlaiseen normaalimuotoon ja tarkistaa jokaisen termin yhtäläisyys erikseen. Tämä ei kuitenkaan monimutkaisemmissa tapauksissa ole järkevää, koska sekä vertailu että sen valmistelut ovat hitaita. Sormenjälkimenetelmällä tehty tarkistus on nopeampi ja virhetodennäköisyys on joko hyvin pieni tai virhettä ei ole ollenkaan. Polynomiesimerkin tapauksessa sormenjälkimenetelmiin perustuva tarkistus evaluoisi molemmat polynomit samalla syötteellä ja tutkisi vastausten yhtäsuuruutta. Virhetodennäköisyys olisi tässä tapauksessa yhtälön asteesta riippuva mutta tunnettu.

3 Sormenjälkimenetelmät

Sormenjälkimenetelmä voidaan ajatella todistajien joukkona, joiden avulla voidaan kuvata lähdetietojoukon monimutkaisen esityksen helpommin vertailtavaksi sormenjäljeksi. Polynomiesimerkissä todistajia ovat satunnaisesti valitut x:n arvot ja sormenjälkinä polynomeille valittujen x:n arvojen perusteella saadut arvot. Triviaalisti voidaan sanoa että x:n arvo 0 on huono todistaja. Yleisesti hyvä sormenjälkimenetelmä täyttää seuraavat vaatimukset:

- Sormenjälkien tulee olla tarpeeksi yksinkertaisia ja pieniä että niitä voidaan vertailla tehokkaasti. Tämän saavuttamiseksi sormenjälki saa olla jopa epätäydellinen esitys alkuperäisestä tietojoukosta.
- Sormenjäljen tulee sisältää mahdollisimman paljon olennaista informaatiota alkuperäisestä tietojoukosta.

Yleisemmin sormenjälkimenetelmää voidaan kuvata joukkona M, jonka jokainen alkio on funktio, joka tuottaa tietojoukon täydellisestä kuvauksesta sitä vastaavan sormenjäljen. Yksittäisessä tapauksessa ongelmana on löytää M, jonka tarpeeksi moni funktio f tuottaa kuvauksen, jolla erilaisille tietojoukoille O_1 ja O_2 pätee $f(O_1) \neq f(O_2)$. Tästä johtuen sormenjälkimenetelmä onkin todistajien runsaus -menetelmän erityistapaus. Joukko M voidaan ajatella joukoksi todistajakandidaatteja, joilla pyritään todistamaan että $O_1 \not\equiv O_2$ ja $h \in M$ yksittäiseksi todistajaksi sille että $O_1 \not\equiv O_2$, jos $h(O_1) \neq h(O_2)$.

3.1 Sormenjälkimenetelmän toiminta

Tarkoitus on selvittää kahden suuren tietojoukon O_1 ja O_2 ekvivalenssi. Sormenjälkiin perustuva ekvivalenssitarkistus etenee seuraavasti:

1. Tunnetaan joukko M, joka sisältää funktioita, joista jokainen kuvaa tietojoukon täydellisen sisällön pienempään tilaan. Valitaan tästä joukosta yksi funktio h satunnaisesti.

2. Lasketaan $h(O_1)$ ja $h(O_2)$, $h(O_i)$:tä kutsutaan O_i:n sormenjäljeksi, $i = 1, 2$

3. Verrataan $h(O_1)$:n ja $h(O_2)$:n arvoja.
   Jos $h(O_1) = h(O_2)$, ovat O_1 ja O_2 ekvivalentit.

Sormenjälkimenetelmän suunnittelussa olennaista on oikean joukon M löytäminen. Yhtäältä tavoiteltavaa on että käytetyt menetelmät tuottavat mahdollisimman uniikin sormenjäljen ja täten M sisältää paljon todistajiksi sopivia kandidaatteja, toisaalta sormenjäljen täytyy olla mahdollisimman pieni jotta sitä voidaan vertailla tehokkaasti. Nämä tavoitteet ovat selkeästi ristiriidassa, joten soveltuvan kompromissin löytäminen on tärkeää.

4 Esimerkkiongelmia

Tässä kappaleessa kuvataan aluksi kaksi läheisesti toisiinsa liittyvää ongelmaa ja niihin Monte Carlo -tyyppiseen algoritmiin perustuvat ratkaisut, jotka sallivat siten pienellä todennäköisyydellä myös virhepäätelmät. Toiseen ongelmatyyppiin esitän Las Vegas -algoritmin, jonka suorituksessa sormenjälkiä käytetään hyväksi ilman virheen mahdollisuutta. Molemmat algoritmit käyttävät hyväkseen samantyyppisiä sormenjälkiä, mutta ovat silti hyvin erityyppiset.

4.1 Tietoliikenne

Tutkitaan skenaarioita, joissa kahden tietokoneen, R_1 ja R_2, muistien sisältöä halutaan vertailla tehokkaasti. Koska tarkoitus on esitellä sormenjälkimenetelmiä oletetaan, että jokaisessa ongelmassa tietokoneet ovat niin kaukana, joko fyysisesti kaukana tai muuten vaan hitaan yhteyden päässä, toisistaan, ettei muistien sisällön suora vertailu ole tehokasta. Tästä syystä algoritmien tehokkuus mitataan niiden siirtämän tiedon perusteella.

Ensimmäisessä tapauksessa halutaan tutkia onko tietokoneen R_1 sisältämä bittijono $x \in \{0,1\}^*$ sama kuin tietokoneen R_2 sisältämä bittijono $y \in \{0,1\}^*$. Deterministinen algoritmi lähettäisi datan kononaisuudessaan toiselta koneelta toiselle, minkä jälkeen vertailu tehtäisiin sana(word) kerrallaan. Lisäksi siirretyn tiedon eheys aiheuttaa lisäkustannuksia.

Merkitään bittijonoja $x = x_1x_2x_3 \dots x_n$ ja $y = y_1y_2y_3 \dots y_n$, $x_i, y_i \in 0, 1$, $i = 1, \dots, n$. Number(x) tarkoittaa bittijonon x kymmenkantaista lukutulkintaa. Mahdollisten todistajien joukoksi otetaan kaikki alkuluvut $< n^2$, merkitään tätä joukkoa $Prim(n^2)$. Sormenjälkialgoritmi toimii seuraavasti

1. R_1 valitsee sormenjäljen generointia varten luvun $p \in Prim(n^2)$

2. R_1 laskee valitun luvun avulla
   $s = Number(x) \bmod p$
   ja lähettää sekä s:n että p:n R_2:lle

3. R_2 vastaanottaa s:n ja p:n ja laskee
   $q = Number(y) \bmod p$
   Jos $q = s$, todetaan että $x = y$, muuten $x \neq y$.

Algoritmin toiminnan kannalta olennaista ovat siirretyn tiedon määrä ja algoritmin virhetodennäköisyys. Tiedon määrästä tiedetään, että $s \leq p < n^2$. Tästä seuraa että s tai p vie maksimissaan $log_2\ n^2$ bittiä, joten koko viestin pituudeksi tulee $2 * log_2\ n^2 \leq 4 * log_2\ n$. Jos $n = 10^{16}$ saadaan siirrettävän tiedon määräksi $4 * 16 * log_2 10 = 256$ bittiä, joka on huomattava vähennys 10^{16} bittiin nähden.

Virhetodennäköisyyden arviointia varten jaetaan kaikki p:n mahdolliset arvot(alkuluvut jotka ovat pienempiä kuin n^2, merkitään Prim(n^2)) kahteen joukkoon, niihin jotka aiheuttavat virheen ja niihin jotka eivät). Koska jokaisen $p \in$ Prim(n^2) valintatodennäköisyys on sama on virhetodennäköisyys

$$\frac{huonotluvutPrim(n^2) : ssa}{Prim(n^2)}.$$

Alkulukuteoreeman perusteella tiedetään, että Prim(n^2) $> \frac{n^2}{2*\ln n}$. Bad(n^2):n maksimiarvoksi voidaan näyttää $n - 1$. Tästä saadaan, että kuvatun algoritmin virhetodennäköisyys on $\frac{\ln n^2}{n}$. Tapuksessa $n = 10^{16}$, virhetodennäköisyydeksi saadaan $0,36892 * 10^{-14}$, mikä ei ole todellinen riski.

Toinen tapaus on laajennettu ensimmäisestä. Tässä koneen R_1 muistissa on bittijono $x \in \{0,1\}^*$. Toisen koneen R_2 muistissa on bittijonojoukko $U = \{u_1, u_2, u_3, \ldots, u_k\}, u_i \in \{0,1\}^*, i = 1, \ldots, k$. Halutaan selvittää kuuluuko jono x joukkoon U. Deterministisellä algoritmillä ainoa tapa olisi lähettää jono x kokonaisuudessaan koneelle R_2, joka sen jälkeen vertaisi x:ää jokaiseen U:n alkioon. Satunnaisalgoritmi PSet etenee ensimmäisessä tapauksessa kuvatun mukaisesti mutta R_2 laskee p:n avulla

$q_i = Number(u_i) \bmod p, i = 1, \ldots, k,$

minkä jälkeen jos $s \in \{q_1, q_2, \ldots, q_k\}$ todetaan että $x \in U$, muuten $x \notin U$.

Kuvatun protokollan vaatima tiedonsiirto on triviaalisti sama kuin aiemmassa esimerkissä($4 * \log_2 n$), joten tutkitaan järjestelmän virhetodennäköisyyttä. Tilanne jaetaan kahteen tapaukseen

- Jos $x \in U$, tiedetään että on olemassa $j \in \{1, \ldots, k\}$ jolla $x = u_j$ ts. $Number(x) = Number(u_j) \bmod m$, joten algoritmi toimii varmasti kun $x \in U$.
- Jos $x \notin U$, merkitään A_i sitä tapausta kun

  $Number(x) \bmod p = Number(u_i) \bmod p, i = 1, \ldots, k$

  ja

  $A = \cup_{i=1}^{k} A_i,$

  joka on tapaus, jossa PSet tuottaa väärän vastauksen $x \in U$. Aiemman esimerkin perusteella tiedetään, että

  $P(A_i) \leq \frac{n-1}{Prim(n2)} \leq \frac{2*\ln n}{n}$, kaikille $i \in \{1, \ldots, k\}$.

  ja täten

$$
\begin{aligned}
Virhetn_{PSet}(x,U) &= P(A) \\
&= P(\cup_{i=1}^{k} A_i) \\
&\leq \Sigma_{i=1}^{k} P(A_i) \\
&\leq \Sigma_{i=1}^{k} \frac{2 * \ln n}{n} \\
&= k * \frac{2 * \ln n}{n},
\end{aligned}
$$

josta seuraa että

$Virhetn_{PSet}(x,U) \leq \frac{1}{2}$, kun $k \leq \frac{n}{4 * \ln n}$.

Täten PSet on yksipuolisen virheen(one-sided error) Monte Carlo -algoritmi, joka tarkistaa kuuluuko x joukkoon U, kun $k \leq \frac{n}{4 * \ln n}$.

Molemmissa kuvatuissa ongelmissa algoritmin virhetodennäköisyyttä voidaan pienentää käyttämällä useampaa alkulukua todistajana. Lisäksi voidaan näyttää, että virheen aiheuttavien alkulukujen määrä on aina maksimissaa $n - 1$, joten virheellisten ja hyvien alkulukujen suhdetta voidaan parantaa myös kasvattamalla joukkoa, josta alkulukuja valitaan. Algoritmia voidaan amplifioida jopa niin että hyötyjen katoamatta algoritmin virhetodennäköisyys saadaan niin pieneksi että muista syistä(rautavika, luonnon katastrofi tms.) johtuvien virheiden todennäköisyys on suurempi.

4.2 Alimerkkijono-ongelma

Hahmontunnistus(pattern matching) on yksi yleisimmistä tekstinkäsittelyyn ja algoritmeihin liittyvistä ongelmista. Sillä on myös käytännön sovellutuksia yleisesti molekyylibiologiassa. Hahmontunnistamisen perusteella voidaan jaotella asioita eri kategorioihin tai tehdä hakuja tietojoukosta.

Tässä kappaleessa kuvataan sormenjälkiä käyttävä tapa tunnistaa alimerkkijonoja tehokkaasti. Aluksi tunnetaan merkkijono $x = x_1x_2x_3 \ldots x_n$ ja teksti $y = y_1y_2 \ldots y_m$ aakkostossa Σ(ts. $x_i \in \Sigma$, $i = 1, \ldots, n$ ja $y_j \in \Sigma$, $j = 1, \ldots, m$). Halutaan selvittää sisältääkö y merkkijonon x. Lisäksi, mikäli x on y:n alimerkkijono, halutaan tietää myös pienin indeksi r, jolla

$$x_1x_2 \ldots x_n = y_r y_{r+1} \ldots y_{r+n-1}.$$

Selvyyden vuoksi tässä esimerkissä valitaan aakkostoksi $\Sigma = \{0, 1\}$. Jokaiselle $k \in \{1, \ldots, m-1\}$ ja $r \in 1, \ldots, m - k + 1$ merkintä

$$y(r,k) = y_r y_{r+1} \ldots y_{r+k-1}$$

tarkoittaa sitä k-pituista alimerkkijonoa, joka alkaa indeksistä r. Annetuille $x = x_1x_2 \ldots x_n$ ja $y = y_1y_2 \ldots y_m$ yksinkertainen deterministinen algoritmi vertaa x:ää jokaiseen $y(r,n)$:ään, $r = 1, 2, \ldots, m-n+1$. Yksinkertainen, jokaisen merkin vertailuun vasemmalta oikealle perustuva algoritmi suorittaa $O(n+m)$ operaatiota etsiessään x:ää y:stä. Alla esitellään sormenjälkiä hyväksikäyttävä algoritmi, joka suoriutuu tehtävästä nopeammin.

Algoritmin suorittamista varten määritellään funktio $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, jonka arvoa pienempiä alkulukuja algoritmissa käytetään. STRING(f)-algoritmi etenee seuraavasti

1. Valitaan satunnaisesti alkuluku p joukosta $Prim(f(n,m))$.
2. Lasketaan
   $$Finger_p(x) = Number(X) \bmod p$$
3. Lasketaan järjestyksessä
   $$Finger_p(y(r,n)) = Number(y(r,n)) \bmod p$$
   ja verrataan $Finger_p(y(r,n))$:ää $Finger_p(x)$:hen, kaikille $r \in \{1,2,\ldots,m-n+1\}$.. Jos $Finger_p(y(r,n)) = Finger_p(x)$, verrataan x:ää $y(r,n)$:ään. Jokaiselle $j \in \{1,2,\ldots,m-n+1\}$, jolle pätee $Finger_p(y(j,n)) = Finger_p(x)$, tarkistetaan vielä että x vastaa oikeasti $y(j,n)$:ää. Jos $y(j,n) = x$, algoritmi palauttaa j:n pienimmäksi indeksiksi josta merkkijono x löytyi. Muussa tapauksessa jatketaan vertailua $Finger_r(y(j+1,n))$:ästä.

Yllä esitelty algoritmi on Las Vegas -tyyppinen, koska kaikki osumat tarkistetaan vielä alkuperäisiä merkkijonoja käyttäen, mikä estää virheellisten osumien palauttamisen. Algoritmin aikakompleksisuutta tutkitaan seuraavaksi. Triviaalisti $Finger_p(x)$ tai $Finger_p(1,n)$ vie aikaa $O(n)$. Kaikki sormenjäljet

$$Finger_p(y(r,n)), kun\ r = 1,\ldots,m-n+1$$

voidaan laskea ajassa $O(m)$. Tämä pitää paikkansa koska valitusta aakkostosta johtuen

$$Number(y(r+1,n)) = 2 * [Number(y(r,n)) - 2^{n-1} * y_r] + y_{r+n}$$

ja tästä johtuen sormenjälki $Finger_p(y(r+1,n))$ voidaan laskea

$$Finger_p(y(r+1,n)) = (2 * [Finger_p(y(r,n)) - (2^{n-1} * y_r) \bmod p] + y_{r+n}) \bmod p.$$

Myös millä tahansa muulla aakkostolla voidaan alimerkkijonojen lukuarvot laskea edellisen alimerkkijonon arvon avulla.

Tutkitaan tapauksen, jossa y ei sisällä alimerkkijonona x:ää, aikavaatimuksen ylärajaa. Merkitään A_r sitä tapausta, jossa silti $Finger_p(x) = Finger_p(y(r,n))$. Edellisestä kappaleesta tiedetään että virheellisen sormenjäljen todennäköisyys on

$$P(A_r) \leq \frac{n-1}{Prim(f(n,m))} \leq \frac{n * \ln f(n,m)}{f(n,m)}$$

Täten STRING(f)-algoritmin aikavaatimus on

$O(m) + \Sigma_{r=1}^{m-n+1}(1 + Prob(A_r) * n)$, $O(m)$-ajassa saadaan laskettua $Finger_p(x)$ ja kaikki $Finger_p(y(r,n))$:n arvot.

$\leq O(m) + m * n^2 * \frac{\ln f(n,m)}{f(n,m)}$

Jos valitaan $f(n,m) = n^2 m * \ln(n^2 m)$, saadaan

$$Aikavaatimus_{STRING(n^2 m * \ln(n^2 m))}(x,y) \in O(m)$$

Tilanteen, jossa x löytyy y:stä aikavaatimus on pienempi, koska algoritmin suoritus loppuu tällöin aiemmin.

5 Käytännön sovelluksia

Yllä kuvatut esimerkit ovat luonteeltaan hyvin teoreettisia, joten tässä kuvaan muutamia käytännön sovelluksia, missä sormenjälkitekniikoista on hyötyä. Useat sormenjälkitekniikoiden käytännön sovellukset perustuvat Las Vegas -algoritmeihin, joissa jotain vaikeaa hakua nopeutetaan filtteröimällä tietoa alimerkkijonoesimerkin(ks. 4.2) tapaisesti ja hakemalla tästä suppeammasta joukosta osumia tehokkaasti. Tällaisia ongelmia on usein esimerkiksi tekstinkäsittelyssä.

Monte Carlo -tyyppinen esimerkki esitetään “Identifying redundancy in source code using fingerprints”-konfferenssipaperissa[2] tapa käyttää sormenjälkiä toisteisuuden tutkimiseen isossa lähdekoodimäärässä. Tämäkin menetelmä perustuu osittain kappaleessa 4.2 esitetyn alimerkkijono-ongelman ratkaisuun.

Tässä tapauksessa lähdekoodeista generoidaan sormenjäljet vastaavasti kuin alimerkkijonoesimerkin tekstistä y. Toisin kuin alimerkkijonoesimerkissä tässä käytettyjen alimerkkijonojen pituus on jokin rivimäärä n lähdetiedostoissa, mikä aiheuttaa sen että eri kohdissa syötettä sormenjälki generoidaan eri pituisista merkkijonoista. Koska tietoa on huomattava määrä sormenjälkien generoimisenkin jälkeen, poimitaan kaikista generoiduista sormenjäljistä ne, jotka kattavat pisimmät merkkijonot, kuvaamaan kaikkia merkkijonoja, jotka sisältyvät sormenjälkeen.

Jäljelle jääneistä sormenjäljistä poistetaan vielä ne jotka esiintyvät vain kerran, sekä yhdistetään peräkkäiset sormenjäljet, jotka esiintyvät joka paikassa samalla tavalla peräkkäin. Nyt jäljelle jääneiden sormenjälkien perusteella voidaan tutkia mahdollisia toisteisuus tarkemmin.

6 Yhteenveto

Sormenjälkimenetelmiä voidaan käyttää hyväksi tietyntyyppisissä päätösongelmissa. Sormenjälkien tarkoituksena on kuvata iso tietojoukko pienempään tilaan, siten että sormenjälkien perusteella tehty vertailu on tehokkaampaa.

Sormenjälkitekniikoiden käyttötapoja on sekä Monte Carlo -tyyppisissä että Las Vegas -tyyppisissä algoritmeissa. Monte Carlo -algoritmeissa ongelman ratkaisu perustuu suoraan tietojoukoista tuotettuun sormenjälkeen, mikä sormenjäljen ominaisuuksien vuoksi aiheuttaa virheitä tietyllä todennäköisyydellä. Usein kuitenkin virhetodennäköisyys on niin pieni(tai voidaan amplifioimalla saattaa hyvin pieneksi), että sitä ei tarvitse ottaa huomioon.

Toinen tapa käyttää sormenjälkiä on käyttää niitä tiedon filtteröintiin jolloin tarkempi vertailu voidaan tehdä sille tietojoukolle, jonka sormenjälki vastaa toista vertailtavaa. Tässä tapauksessa virheitä ei tule ja täten algoritmi on Las Vegas -tyyppinen, mutta filtteröinti vähentää silti vaadittavien vertailujen määrää ja siten tehostaa ongelman ratkaisua.

Viitteet

[1] Juraj Hromkovic. *Design And Analysis of Randomized Algorithms, Introduction to Design Paradigms*, chapter 4. Springer, 2005.

[2] J. Howard Johnson. Identifying redundancy in source code using fingerprints, 1993.