Läheisyyshaku

Leena Salmela
leena.salmela(at)hut.fi

Seminaariesitelmä 15.3.2006
T-106.850 Tiedonhaku
Tietotekniikan osasto
Teknillinen korkeakoulu

Tiivistelmä

Kun tehdään hakuja suuriin dokumenttikokoelmiin, hakutuloksia tulee erittäin paljon. Käyttäjälle on tällöin hyvin tärkeää, että tulokset pystytään järjestämään siten, että tärkeimmät dokumentit ovat tuloslistan kärjessä. Yksi hyvä mittari dokumentin relevanssille on, kuinka lähellä toisiaan hakusanat esiintyvät. Läheisyyshakuun on esitetty useita algoritmeja ja lisäksi on tutkittu, kuinka tällaisten arvojen huomiointi vaikuttaa hakutulosten tarkkuuteen.

1 Johdanto

Haut suuriin tekstikokoelmiin tuottavat erittäin paljon hakutuloksia. Näistä tuloksista pitäisi pystyä erottamaan olennaisimmat dokumentit ja tulokset pitäisi järjestää jonkinlaiseen tärkeysjärjestykseen. Tällaisen järjestyksen muodostamiseksi on esitetty useita menetelmiä. Järjestys voi esimerkiksi perustua siihen, kuinka usein hakusanat esiintyvät dokumentissa, tai voidaan käyttää linkkiperusteista arvottamista kuten esim. PageRank [1].

Dokumentin olennaisuutta voidaan mitata myös tutkimalla, kuinka lähellä toisiaan hakusanat esiintyvät. Jos hakusanat esiintyvät lähekkäin, niiden käyttö liittyy todennäköisesti yhteen. Jos hakusanat taas esiintyvät kaukana toisistaan, niitä saatetaan hyvinkin käyttää aivan erillisissä yhteyksissä.

Hakukoneet järjestävät useimmiten hakutulokset sellaiseen järjestykseen, että dokumentit, joissa hakusanat esiintyvät lähekkäin, tulevat ennen muita dokumentteja. On kuitenkin epäselvää, laskevatko hakukoneet minkäänlaisia läheisyysarvoja vai syntyykö tällainen järjestys jonkin muun järjestysalgoritmin tuloksena.

Läheisyyshakua voidaan myös käyttää erillisenä hakumenetelmänä, jolloin käyttäjä määrittelee, kuinka lähellä toisiaan hakusanojen tulee esiintyä. Tässä esitelmässä keskitytään erilaisiin menetelmiin, joilla läheisyysarvoja voidaan määrittää.

Läheisyyshakuun liittyviä julkaisuja etsiessä kannattaa huomata, että terminologia ei ole aivan vakiintunutta. Useimmiten, kun puhutaan läheisyyshakualgoritmeista, käytetään termiä läheisyyshaku (proximity search). Tällä kuitenkin joissain yhteyksissä tarkoitetaan samankaltaisten dokumenttien hakua. Tällöin annettuna on kokoelma objekteja (esim. verkkosivuja) ja jokin etäisyysfunktio ja tarkoituksena on löytää ne objektit, jotka ovat riittävän lähellä kyselyobjektia. Tästä johtuen etenkin tiedonlouhintaan liittyvillä julkaisufoorumeilla käytetään usein termejä sanojen yhteisesiintymä (word co-occurrence) ja fraasien etsintä (phrase searching), kun puhutaan läheisyyshausta.

2 Määritelmiä

Läheisyyshaku voidaan määritellä muutamalla hieman toisistaan poikkeavalla tavalla. Yksinkertaisin näistä on kenties **läheisyyshaku tietyllä ikkunan leveydellä**. Tällöin syötteenä on annettu k hakusanaa ja ikkunan leveys d. Tarkoituksena on etsiä annetusta dokumentista kaikki sellaiset d-levyiset ikkunat, joissa kaikki k hakusanaa esiintyvät. On myös mahdollista, että ikkunan leveyttä ei ole annettu, vaan sopiva leveys määritetään hakusanojen perusteella.

Toinen tapa määritellä läheisyyshaku on **minimaalisten esiintymien läheisyyshaku**. Tällöin syötteenä saadaan k hakusanaa ja algoritmin tulee etsiä kaikki sellaiset ikkunat, joissa kaikki hakusanat esiintyvät, mutta jotka eivät sisällä toisia tällaisia ikkunoita.

Näitä perusongelmia voidaan vielä muunnella. Jos hakusanoja on paljon, ei välttämättä ole mielekästä etsiä ikkunoita, joissa kaikki hakusanat esiintyvät, vaan voidaan etsiä ikkunoita, joissa esiintyy ainakin $k' \leq k$ hakusanaa.

Toisaalta käyttäjä voi hakea myös esim. hakusanoilla "Johnson and Johnson". Tällöin on oleellista, että sana Johnson esiintyy kahdesti. **Yleinen läheisyyshaku** onkin määritelty seuraavasti: Syötteenä saadaan k hakusanaa ja kullekin hakusanalle kerroin eli kuinka monta kertaa sen pitää esiintyä. Tavoitteena on nyt löytää pienin ikkunan koko, jolla hakusanat esiintyvät kertoimensa mukaisesti kyseisen levyisessä ikkunassa.

Useimmiten tekstikokoelmat indeksoidaan käyttäen tietorakenteena käänteistiedostoa (inverted file). Näin ollen läheisyyshakualgoritmit saavat syötteenään dokumentin sijaan kuhunkin hakusanaan liittyvän (järjestetyn) positiolistan.

Yllä esitetyt ongelmat ovat hyvin samankaltaisia kuin rinnakkaisten episodien hakuun [7, 8] liittyvät ongelmat ja monet ratkaisualgoritmitkin ovat hyvin samankaltaisia.

3 Ikkunan koon määrityksestä

Jos käyttäjä ei anna ikkunan kokoa, ensimmäinen ongelma on määrittää sopiva koko hakusanojen perusteella. Intuitiivisesti voitaisiin ajatella, että ikkunan koon pitäisi kuvata jotain semanttista kokonaisuutta, kuten lausetta tai kappaletta. Tällaisilla ikkunan koille saadaan tarkempia tuloksia kuin pelkkään sanojen esiintymistiheyteen perustuvilla menetelmillä [9].

Toisaalta riippuu myös hakusanojen merkityksestä, kuinka lähellä toisiaan niiden tulisi esiintyä. Liu et. al [5] ovat luokitelleet hakusanoja tämän perusteella neljään ryhmään ja määrittäneet kullekin ryhmälle sopivan ikkunan koon siten, että dokumentti, jossa hakusanat esiintyvät ja joka on relevantii, kuuluuu suurella todennäköisyydellä hakutulokseen, ja dokumentti, jossa hakusanat esiintyvät, mutta joka ei ole relevantti, kuuluu pienellä todennäköisyydellä hakutulokseen. Ryhmäjako ja vastaavat ikkunan koot olivat seuraavat:

- **Erisnimet:** Ihmisten, paikkojen ja organisaatioiden nimet. Erisnimet sijaitsevat käytännössä peräkkäin ja samassa järjestyksessä kuin annetussa haussa. Ikkunan kooksi sopii siis k.
- **Sanakirjafraasit:** Sanakirjoista löytyvät fraasit. Hakusanat esiintyvät melko lähellä toisiaan, mutta välissä voi olla paljonkin muita sanoja. Ikkunan koon tulee olla noin 15.
- **Yksinkertaiset fraasit:** Yksinkertaisessa fraasissa ei ole sisäkkäistä substantiiveista koostuvaa fraasia. Esimerkiksi "school uniform" on yksinkertainen fraasi. Tässä tapauksessa hakusanat voivat esiintyä kaukanakin toisistaan. Ikkunan kooksi voidaan valita noin 50.

- **Kompleksiset fraasit:** Kaikki sellaiset fraasit, jotka sisältävät useampia yksinkertaisia tai sanakirjafraaseja. Tässä tapauksessa eri fraasin osat voivat esiintyä hyvin eri kohdissa dokumenttia. Ikkunan kooksi ehdotetaan 80, mutta voisi olla mielekkäämpää vaatia, että riittää, että vain osa hakusanoista esiintyy ikkunassa.

4 Algoritmeja

Alla on esitetty algoritmeja, jotka ratkaisevat jonkin kappaleessa 2 esitetyistä läheisyyshaun versioista. Eri versiot läheisyyshausta ovat sen verran lähellä toisiaan, että kukin algoritmi on melko helppo muuntaa ratkaisemaan jokin toinen ongelmaversio.

Kaikki alla esitetyt algoritmit ottavat syötteenään kullekin hakusanalle w_i positiolistan p_i ($1 \leq i \leq k$). Yleisen löheisyyshaun tapauksessa syötteenä on myös hakusanojen kertoimet R_i. Lisäksi syötteenä saatetaan antaa myös, kuinka monen hakusanan tulee esiintyä ikkunassa, jotta kyseessä olisi esiintymä. Tätä merkitään k':lla.

4.1 Plane-sweep algoritmi

Plane-sweep -algoritmi [10] ratkaisee minimaalisten esiintymien läheisyyshaun. Syötteenä on siis kullekin hakusanalle w_i positiolista p_i ja algoritmi tuottaa listan minimaalisista esiintymistä. Plane-sweep -algoritmi olettaa, että positiolistat ovat järjestettyjä; Jos näin ei ole, listat voidaan ensin järjestää.

Aluksi kustakin positiolistasta poistetaan ensimmäinen alkio ja näistä muodostetaan mahdollinen esiintymä. Olkoon nyt tämän mahdollisen esiintymän vasemmanpuolisin alkio sanan w_l positio. Jos vastaava positiolista p_l on tyhjä, mahdollinen esiintymä lisätään minimaalisiin esiintymiin ja algoritmi palauttaa minimaaliset esiintymät.

Jos sanan w_l positiolistan seuraava alkio tulee mahdollisen esiintymän viimeisen alkion jälkeen, lisätään mahdollinen esiintymä löydettyihin minimaalisiin esiintymiin. Tämän jälkeen poistetaan mahdollisen esiintymän ensimmäinen alkio. Lisäksi poistetaan tätä alkiota vastaavasta positiolistasta ensimmäinen alkio ja lisätään se mahdolliseen esiintymään. Tämän jälkeen palataan tutkimaan esiintymän vasemmanpuolista alkiota.

4.2 Hajoita-ja-hallitse algoritmi

Sadakane ja Imai [10] esittävät myös algoritmin, joka perustuu hajoita-ja-hallitse tekniikkaan. Tämä algoritmi ei oleta, että positiolistat olisivat järjestettyjä, joten se lienee hieman tehokkaampi, jos indeksista saadut listat eivät ole järjestettyjä.

Aluksi positiolistoista määritetään mediaani v. Sitten kaikki positiolistat jaetaan kahteen osaan: toisessa on mediaanin vasemmalla puolella olevat positiot ja toisessa oikealla puolella olevat. Sitten etsitään esiintymiä, jotka sisältävät positioita sekä oikeasta että vasemmasta listasta. Tämä tehdään käytännössä plane-sweep -algoritmilla, jolle annetaan vasemman puolen viimeiset positiot ja oikean puolen ensimmäiset positiot.

Jos kaikki hakusanat esiintyvät mediaanin vasemmalla puolella, sieltä etsitään esiintymiä rekursiivisesti. Samoin, jos kaikki hakusanat esiintyvät mediaanin oikealla puolella, hakua jatketaan rekursiivisesti sieltä.

4.3 Kimin algoritmi

Kim et. al [2, 3] esittelevät plane-sweep -algoritmista muunnoksen yleiseen läheisyyshakuun. Tässä tapauksessa järjestettyjen positiolistojen lisäksi syötteenä saadaan kullekin hakusanalle kerroin R_i, joka siis määrittelee, kuinka monta kertaa kyseisen sanan tulisi esiintyä ikkunassa.

Aluksi eri hakusanojen positiolistat limitetään keskenään. Lopputuloksena saadaan lista L, jonka alkiot ovat pareja (i, j), jossa i on hakusanan tunnus ja j on positio.

Algoritmi perustuu tämän hetkisen ikkunan määrittelyyn ja ikkunan liikuttamiseen oikealle listan L mukaisesti. Aluksi ikkunan vasen laita on listan L positiossa $p_l = 0$ ja oikea laita $p_r = -1$. Ikkunaan ei siis kuulu yksikään hakusanan esiintymä. Kullekin hakusanalle w_i lasketaan, kuinka monta kertaa se esiintyy tämän hetkisessä ikkunassa. Tätä muuttujaa merkitään c_i:llä. Jotta esiintymät tunnistettaisiin oikein, tarvitaan vielä laskuri h, joka kertoo, kuinka moni hakusana esiintyy riittävän monta kertaa ikkunassa. Laskurit c_i ja h alustetaan nollaksi.

Algoritmi siirtää aina ikkunan oikeaa laitaa p_r, kunnes ikkunassa on mahdollinen esiintymä eli kukin hakusana w_i esiintyy vähintään R_i kertaa. Tämän jälkeen ikkunan vasenta laitaa p_l siirretään eteenpäin, kunnes ikkunassa ei enää ole mahdollista esiintymää. Tällöin edellinen ikkuna sisälsi minimaalisen esiintymän. Tämän jälkeen siirrytään taas siirtämään ikkunan oikeaa laitaa.

Minimaaliset esiintymät palautetaan, kun ikkunan oikea laita valuu listan L yli.

4.4 Loppuosataulukot ja ortogonaaliset välikyselyt

Läheisyyshaku annetulla ikkunan koolla d voidaan tehdä myös loppuosataulukoilla ja ortogonaalisilla välikyselyillä [6]. Loppuosataulukko voidaan rakentaa järjestämällä kaikki tekstin loppuosat leksikograafiseen järjestykseen ja asettamalla loppuosataulukon i:nteen alkioon i:nnen loppuosan alkukohta tekstissä. Käytännössä loppuosataulukon rakentamiseen on huomattavasti tehokkaampia algoritmeja. Loppuosataulukon lisäksi tarvitsemme tietorakenteen, johon talletetaan kaksiulotteisen avaruuden pisteitä ja josta on helppo laskea ortogonaaliseen kaksiulotteiseen väliin kuluuvat pisteet. Tällainen on esimerkiksi kd-puu.

Esikäsittelyvaiheessa tekstistä rakennetaan ensin loppuosataulukko. Sitten kutakin tekstin positioparia (x, y), missä $0 < |y - x| \leq d$, vastaava loppuosapositiopari $(\mathrm{suf}(x), \mathrm{suf}(y))$, missä siis $\mathrm{suf}(x)$ tarkoittaa x:stä alkavan loppuosan indeksiä loppuosataulukossa, talletetaan sopivaan tietorakenteeseen. Hakuvaiheessa loppuosataulukosta voidaan nopeasti etsiä kahta hakusanaa vastaavat loppuosapositiovälit. Näillä väleillä voidaan sitten tehdä kyselyjä esikäsittelyvaiheessa rakennettuun tietorakenteeseen.

5 Läheisyystiedon käyttö dokumenttien järjestämisessä

Yllä esitettyjä algoritmeja voidaan käyttää läheisyysarvojen määrittämiseen. Jos käyttäjä määrää ikkunan leveyden, tätä voidaan käyttää hakutulosten rajaamiseen ja, vaikka käyttäjä ei määrittäisi ikkunan leveyttä, sopiva leveys voidaan päätellä ja käyttää tätä hakutulosten rajaamiseen. Toisaalta, jos tiedetään, kuinka lähellä toisiaan hakusanat sijaitsevat dokumenteissa, tätä tietoa voidaan käyttää, kun dokumenteille määritetään tärkeysjärjestys.

Dokumenteille voidaan laskea läheisyysarvot positiolistoista aina, kun niitä tarvitaan. Toisaalta, jos läheisyysarvoja tarvitaan vain joillain ikkunan koilla, nämä voidaan myös tallettaa dokumenttia indeksoitaessa. Ilmeisesti siitä, kuinka nämä talletetaan, ei ole juuri tehty tutkimusta.

Läheisyystiedon käytön vaikutusta hakutulosten tarkkuuteen on sen sijaan tutkittu jonkin verran. Kwok [4] on tutkinut läheisyystiedon vaikutusta kaksisanaisten kyselyjen käsittelyssä järjestämällä

uudestaan hakutulokset läheisyytiedon perusteella. Hakutulosten tarkkuus parani hänen tutkimuksissaan jonkin verran.

Mittendorf et. al [9] taas ovat tutkineet läheisyystiedon vaikutusta probabilistiseen hakuun (probabilistic retrieval). He vertasivat hakutuloksia muutamilla eri ikkunan pituuksilla vaatien, että ainakin kaksi hakusanaa esiintyy ikkunassa. Heidän tutkimuksissaan haun tarkkuus parani etenkin pitkillä kyselyillä, jos käytettiin vain läheisyyshakua, mutta, kun läheisyyshaku yhdistettiin tavalliseen termihakuun, tarkkuus parantui vain, jos eri metodien painokertoimet määritettiin hakukohtaisesti.

Viitteet

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[2] S.-R. Kim and J. Hong. An effecient and practical algorithm for the many-keyword proximity problem by offsets. In *Proc. RSFDGrC'05, Lecture Notes in Computer Science*, volume 3642, pages 477–483, 2005.

[3] S.-R. Kim, I. Lee, and K. Park. A fast algorithm for the generalized k-keyword proximity problem given keyword offsets. *Information Processing Letters*, 91(3):115–120, 2004.

[4] K. L. Kwok. Higher precision for two-word queries. In *Proc. of SIGIR'02*, pages 395–396, 2002.

[5] S. Liu, F. Liu, C. Yu, and W. Meng. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *Proc. SIGIR'04*, pages 266–272, 2004.

[6] U. Manber and R. Baeza-Yates. An algorithm for string matching with a sequence of don't cares. *Information Processing Letters*, 37(1):133–136, 1991.

[7] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proc. KDD'96*, pages 146–151. AAAI Press, 1996.

[8] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. KDD'95*, pages 210–215. AAAI Press, 1995.

[9] E. Mittendorf, B. Mateev, and P. Schäuble. Using the co-occurrence of words for retrieval weighting. *Information Retrieval*, 3:243–251, 2000.

[10] K. Sadakane and H. Imai. Fast algorithms for k-word proximity search. *IEICE Trans. Fundamentals*, E84-A(9):312–319, 2001.