Sekvenssihahmojen louhinta XML-dokumenteista

Harri Härkönen

25.05.2008

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Tiedonlouhintamenetelmillä pyritään löytämään a priori tuntematonta informaatiota tiedosta. Tällaista informaatiota ovat esimerkiksi assosiaatiot eri asioiden välillä. Menetelmiä, joilla pyritään peräkkäisten assosiaatioiden ja säännönmukaisuuksien löytämiseen kutsutaan sekvenssihahmojen louhinnaksi. Ensimmäinen ja kenties tunnetuin sekvenssihahmoja louhiva algoritmi on AprioriAll.

XML on yleinen tapa ratkaista tiedon varastointiin ja siirtoon liittyviä ongelmia. XML-muotoisen tiedon yleistyessä syntyy myös tarve pystyä soveltamaan tiedonlouhintamenetelmiä XML-muotoiseen tietoon. Suoraviivaisinta tämä on tehdä toteuttamalla käytetyt menetelmät, kuten AprioriAll-algoritmi, XML-käsittelykielten avulla. Esimerkki XML-käsittelykielestä on XSLT 2.0. XML-pohjaisilla menetelmillä sekvenssihahmojen louhinta XML-muotoisesta tiedosta voidaan tehdä ilman esikäsittelyä, jolloin informaatio XML-muotoiseen tietoon sisältyvistä suhteista ja yhteyksistä voidaan säilyttää.

Avainsanat: AprioriAll-algoritmi, sekvenssihahmo, XML, XSLT 2.0

SISÄLLYSLUETTELO

1 JOHDANTO

Tietojenkäsittelytieteen keskeisimpiä ongelmia on tietojärjestelmien integraatio. Tämä on usein vaikeaa, koska yhtyeensopimattomien rajapintojen vuoksi tiedonsiirto tietojärjestelmien välillä on tehotonta ja virhealtista, ja erilaisista tiedon tallennustavoista johtuen tietomallien yhdistäminen on monimutkaista. Viimeisen vuosikymmenen aikana XML:stä on tullut yleinen tapa ratkaista tiedonsiirtoon ja tallennukseen liittyviä ongelmia. XML:n etuja on, että se tarjoaa standardin, yksinkertaisen ja joustavan tavan välittää ja tallentaa tietoa.

Samalla kun XML yleistyy tiedon tallennusmuotona, lisääntyy myös tarve pystyä tehokkaasti louhimaan tietoa XML:stä. Yksi tiedonlouhinnan osa-alue on assosiaatiosääntöjen johtaminen, jossa pyritään a priori tuntemattomien suhteiden ja yhteyksien löytämiseen tiedosta. Assosiaatiosääntö voidaan ilmaista esimerkiksi seuraavasti: "90 % asiakkaista, jotka ostivat leipää ja voita, ostivat myös maitoa".

Inhimillisessä arkikokemuksessa asiat järjestyvät tavallisesti yhtenäisiksi tapahtumakuluiksi (ajallisesti), ja/tai moniulotteisten objektien osiksi (avaruudellisesti). Tietojärjestelmät taas usein tallettavat tietoa tavalla, joka on luonteeltaan peräkkäistä. Siksi pelkkien assosiaatioiden löytäminen tiedosta paljastaa vain osan säännönmukaisuuksiin sisältyvästä hyödyllisestä informaatiosta. Menetelmiä, joilla pyritään peräkkäisten assosiaatioiden ja säännönmukaisuuksien löytämiseen kutsutaan sekvenssihahmojen louhinnaksi.

Sekvenssihahmo voidaan ilmaista esimerkiksi seuraavasti: "5 % asiakkaista osti 'Säätiön', sen jälkeen 'Säätiön ja imperiumin', ja sen jälkeen 'Toisen Säätiön'". Assosiaatiosääntöön sisältyy siis tässä tapauksessa myös tieto alkioiden välisestä järjestyksestä. Ensimmäiset menetelmät, jolla sekvenssihahmoja voitiin louhia, esittelivät Agrawal ja Srikant (1995). Yksi heidän esittelemistään algoritmeista on AprioriAll-algoritmi, joka on tunnetun assosiaatiosääntöjä louhivan Apriori-algoritmin sovellutus sekvenssihahmojen louhintaan (Agrawal & Srikant, 1995).

Menetelmät ja välineet, joilla XML-muotoiseen tietoon voidaan tehdä tietojen käsittelyyn liittyviä operaatioita, ovat uusia ja vasta kehityksen alkuvaiheessa. Tärkeimmät XML-muotoisen tiedon käsittelyyn tarkoitetut kielet ovat W3C-konsortion kehittämät XML-elementteihin viittaamiseen tarkoitettu XPath (W3C, 2008d), XML-muodossa tallennetun tiedon kyselykieleksi suunniteltu XQuery-käsittelykieli (W3C, 2008c), sekä XML:n udelleenmuotoilua toteuttava XSLT-käsittelykieli (W3C, 2008e).

Tieteellisessä kirjallisuudessa esitetyt lähestymistavat XML-muotoisen tiedon tiedonlouhintaan voidaan jakaa kahteen toisestaan poikkeavaan kategoriaan (Zhang & al., 2004; Scardina & al., 2004):

1. XML-muotoisen datan esikäsittely, jolloin XML-data muunnetaan ensin relaatiotietokannaksi, ja käytetään perinteisiä kyselykieliä tiedonlouhintaan.
2. XML-käsittelykielen kehitys, jossa pyritään käyttämään ja kehittämään XML-käsittelykieliä suoraan natiivin XML-muotoisen tiedon tiedonlouhintaan.

Ensimmäinen lähestymistapa tuottaa yleensä liikaa relaatioita ja kadottaa samalla tärkeää informaatiota tietoon sisältyvistä suhteista ja yhteyksistä. Esimerkiksi tieto XML-elementtien eksplisiittisestä hierarkkisesta suhteesta voidaan menettää (Zhang & al., 2004). Jälkimmäinen lähestymistapa on vielä suhteellisen vähän tutkittu, mutta koska XML yleistyy kaikilla tietojenkäsittelyn osa-aluilla nopeaa vauhtia, on tärkeää, että teknologioita, joilla tiedonlouhintaa XML-muotoiseen tietoon voidaan suorittaa ilman esikäsittelyä, tutkitaan ja kehitetään (Wan & Dobbie, 2004).

Tässä tutkielmassa perehdytään kirjallisuuden avulla sekvenssihahmojen louhintaan XML-muotoisen tiedon sisällöstä ja rakenteesta, sekä esitellään XSLT 2.0 -käsittelykielellä toteutettu sekvenssihahmoja louhiva AprioriAll-algoritmi. Tutkielman tarkoituksena on tutustua sekvenssihahmojen louhinnan ongelmiin XML-muotoisesta tiedosta ja tutkia AprioriAll-algoritmi toteuttamalla, kuinka XSLT 2.0 soveltuu sekvenssihahmojen louhintaan.

Aluksi luvussa 2 perehdytään kirjallisuuden perusteella sekvenssihahmojen louhinnan peruskäsitteisiin, sekä esitellään niiden louhintaan käytetty AprioriAll-algoritmi. Luvussa 3 esitellään kirjallisuuden perusteella XML:n erityispiirteet tiedonlouhinnan kannalta. Luvussa 4 esitellään ja arvioidaan AprioriAll-algoritmille tekemäni XSLT 2.0 -toteutus. Lopuksi luvussa 5 esitetään loppupäätelmät.

2 SEKVENSSIHAHMOJEN LOUHINTA

Tässä luvussa perehdytään kirjallisuuden perusteella sekvenssihahmojen louhinnan peruskäsitteisiin ja ratkaisumenetelmiin. Luvun ensimmäisessä kohdassa esitellään ongelma. Seuraavaksi toisessa kohdassa määritetään tarkemmin sekvenssihahmojen louhinta ja esitellään peruskäsitteet. Kolmannessa kohdassa esitellään laajennukset sekvenssihahmojen louhinnan ongelmaan, ja neljännessä kohdassa tutustutaan sekvenssihahmojen louhinnan yleisiin ratkaisuperiaatteisiin. Lopuksi viidennessä kohdassa esitellään tarkemmin sekvenssihahmojen louhintaan tarkoitettu AprioriAll-algoritmi.

2.1 Sekvenssihahmojen louhinnan ongelman kuvaus

Toistuvat hahmot (frequent patterns) ovat joukkoja, tapahtumasarjoja tai rakenteita, jotka esiintyvät tiedossa etukäteen määriteltyä raja-arvoa useammin (Han & Kamber, 2001). Esimerkiksi joukko tuotteita, kuten maito ja leipä, jotka esiintyvät toistuvasti yhdessä asiakkaiden ostotapahtumissa, muodostavat yhdenlaisen toistuvan hahmon, eli *usein esiintyvän tietoalkiojoukon* (frequent itemset). Toistuvien hahmojen louhinta on keskeinen tiedonlouhinnan osa-alue, jolla on käyttöä esimerkiksi tiedon indeksoinnissa, luokittelussa ja klusteroinnissa.

Toistuvien hahmojen louhinnan ongelman esittelivät ensimmäisenä Agrawal & al. (1993) ostoskorianalyysiin tarkoitetun assosiaatiosääntöjen etsinnän muodossa. Assosiaatiosääntöjen etsinnässä haetaan mielenkiintoisia yhteyksiä ja korrelaatioita aineiston arvojen välillä.

Sekvenssihahmojen louhinta on läheistä sukua assosiaatiosääntöjen louhinnalle, ja sen esittelivät ensimmäisinä Agrawal ja Srikant (1995). Sekvenssihahmojen louhinnan lähtökohtana on tietokanta, joka sisältää tietosekvensseiksi kutsuttuja sekvenssien joukkoja. Jokainen tietosekvenssi on joukko transaktioita, jotka sisältävät täsmällisesti määriteltyjä tietoalkioita. Jokaiseen transaktioon on liitetty tapahtuma-aika. Sekvenssihahmo koostuu tietoalkiojoukkojen listoista. Transaktioon liittyvien tietoalkioiden lukumäärää ei oteta

huomioon, vaan ainoastaan tieto siitä, sisältyykö tietoalkio transaktioon vai ei. Kaikilla transaktiolla on yksilöllinen tapahtuma-ajankohta. Sekvenssihahmojen louhinnan ongelma on löytää kaikki ne peräkkäiset sekvenssit, joiden kokonaismäärä ylittää etukäteen määritellyn osuuden kaikista sekvensseistä tietokannassa, ja jotka eivät sisälly mihinkään muuhun sekvenssiin.

Sekvenssihahmo kirjakauppaostoksissa voidaan ilmaista esimerkiksi seuraavasti: "5% asiakkaista osti 'Säätiön', sen jälkeen 'Säätiön ja imperiumin', ja sen jälkeen 'Toisen säätiön'".

Tapahtumien ei tarvitse kuitenkaan olla välittömästi toisiaan seuraavia. Vaikka asiakas olisi ostanut muitakin kirjoja yhtä aikaa em. kirjojen kanssa, tai niiden välillä, löytyy sama sekvenssihahmo silti asiakkaan ostotapahtumista. Sekvenssihahmon elementtien ei myöskään tarvitse olla ainoastaan yhden alkion sisältäviä joukkoja, vaan se voidaan ilmaista myös seuraavasti: ”Asiakas osti ’Säätiön’ ja ’Rengasmaailman’, sen jälkeen ’Säätiön ja imperiumin’ ja ’Rengasmaailman insinöörit’, ja sen jälkeen ’Toisen säätiön’”.

Sekvenssihahmoelementtien kaikkien yksilöllisten alkioiden on kuitenkin löydyttävä tiedosta, jotta hahmoa voidaan tukea (Srikant & Agrawal, 1996).

2.2 Sekvenssihahmojen louhinnan määritelmä ja peruskäsitteet

Sekvenssihahmojen louhinnan ongelma voidaan formaalisti esittää seuraavasti (Zaki, 1997):

Olkoon I = $\{i_1, i_2,\ldots, i_m\}$ yksilöllisten attribuuttien joukko, jossa on *m* attribuuttia, ja jossa jokaista attribuuttia kutsutaan *tietoalkioiksi* (item). *Tietoalkiojoukko* (itemset) on epätyhjä ja järjestämätön tietoalkioiden kokoelma. Ilman yleisyyden menettämistä voidaan olettaa, että tietoalkiot ja tietoalkiojoukot on järjestetty kasvavaan järjestykseen. *Sekvenssi* on järjestetty lista, joka sisältää tietoalkiojoukkoja. Tietoalkiojoukkoa *i* kuvataan merkinnällä $(i_1 i_2 ... i_k)$, missä i_j on tietoalkio. Alkiojoukkoa, jossa on *k* alkiota, kutsutaan *k*-

tietoalkiojoukoksi. Sekvenssillä α tarkoitetaan jonoa ($\alpha_1 \mapsto \alpha_2 \mapsto_{...} \mapsto \alpha_n$), jossa sekvenssielementti α_i on tietoalkiojoukko. Sekvenssiä, jossa on *k*-tietoalkiota ($k = \Sigma_j \left| \alpha_j \right|$) kutsutaan *k-sekvenssiksi*. Tietoalkio voi esiintyä ainoastaan kerran yhdessä tietoalkiojoukossa, mutta voi esiintyä useamman kerran yhden sekvenssin eri tietoalkiojoukoissa.

Sekvenssi $\alpha = (\alpha_1 \mapsto \alpha_2 \mapsto_{...} \mapsto \alpha_n)$ (indeksijoukko $N = \{1,2,\ldots,n\}$) on toisen sekvenssin $\beta = (\beta_1 \mapsto \beta_2 \mapsto_{...} \mapsto \beta_n)$ (indeksijoukko $M = \{1,2,\ldots,m\}$) *alisekvenssi,* jos kaikille $x \in N$, on olemassa indeksi $j_x \in M$ siten, että $a_x \subseteq b_{j_x}$, ja mille tahansa $x,y \in N$, jos $x < y$, niin $j_x < j_y$. Agrawalin ja Srikantin (1995) mukaan Sekvenssin α sanotaan siis *sisältyvän* sekvenssiin β. Alisekvenssi merkitään $\alpha \preceq \beta$.

Esimerkiksi sekvenssi ($B \mapsto AC \mapsto D$) on sekvenssin ($AB \mapsto E \mapsto ACE \mapsto BE \mapsto DE$) alisekvenssi, koska sekvenssielementit $B \subseteq AB$, $AC \subseteq ACD$ ja $D \subseteq DE$. Sen sijaan sekvenssi ($AB \mapsto E$) ei sisälly sekvenssiin (ABE), joten sekvenssi ($AB \mapsto E$) se ei ole sekvenssin (ABE) alisekvenssi, eikä myöskään päinvastoin.

Sekvenssi α on *enimmäissekvenssi,* eli *maksimaalinen* (maximal), jos α ei sisälly mihinkään muuhun sekvenssiin (Agrawal & Srikant, 1995).

Transaktiolla T on yksilöllinen avain, ja se sisältää joukon tietoalkioita, siten että $T \subseteq I$. *Asiakkaalla* C on samoin yksilöllinen avain sekä lista transaktiota $\{T_1, T_2,\ldots,T_n\}$. Ilman yleisyyden menettämistä voidaan olettaa, että asiakkaalla on yhtenä ajankohtana vain yksi transaktio, jolloin transaktion ajankohtaa voidaan käyttää transaktion yksilöivänä avaimena. Transaktiot on lajiteltu ajankohdan mukaan nousevaan järjestykseen, jolloin asiakaan transaktiot muodostavat *asiakassekvenssin* $T_1 \mapsto T_2 \mapsto_{...} \mapsto T_n$. Tietokanta D sisältää joukon asiakassekvenssejä.

Asiakassekvenssi C sisältää sekvenssin α, jos $\alpha \preceq C$, eli jos sekvenssi α on asiakassekvenssin C alisekvenssi. Sekvenssin α *tuki* (support) on niiden asiakkaiden, joilla sek-

venssi α löytyy, ja kaikkien asiakkaiden osamäärä, eli:

$$tuki(\alpha,D) = \text{DIV}\ (|\ \{C \in D \mid C\ sisältää\ \alpha\text{:n}\}\ |, |D|)$$

Tietokannan *D* sisältäville sekvensseille määritellään *minimituki* (minimum support), jolla tarkoitetaan raja-arvoa, jonka ylittävien sekvenssien sanotaan olevan *usein toistuvia* (frequent), tai *suuria* (large) (Agrawal & Srikant, 1995). Toisin sanoen sekvenssi on usein toistuva, jos $tuki(\alpha,D) \geq minimituki$. Usein toistuvien *k*-sekvenssien joukkoa ilmaistaan merkinnällä F_k, ja usein toistuvien sekvenssien joukkoa merkinnällä $F = \bigcup_k F_k$.

Kun tietokannan sisältämistä sekvensseistä etsitään kaikki ne sekvenssit, jotka täyttävät minimituen ehdon, ja jotka ovat maksimaalisia, on tietokannan sisältämät sekvenssihahmot löydetty (Agrawal & Srikant, 1995).

2.3 Sekvenssihahmojen louhinnan ongelman laajentaminen

Sekvenssihahmojen louhinnassa on tärkeää pyrkiä poistamaan tuloksesta ne sekvenssihahmot, joiden esiintyminen tiedossa ei ole oleellista. Ilman karsintaa sekvenssihahmojen määrä voi paisua liian suureksi ollakseen käytännöllinen tiedon analysoinnissa. Ensimmäinen luonnollinen oletus on, että ne sekvenssihahmot, jotka toistuvat kaikkein useimmin, ovat myös kaikkein mielenkiintoisimpia. Alkuperäisessä ongelman määrityksessä voidaan rajata sekvenssihahmoista pois ne, jotka eivät täytä minimituen ehtoa. On kuitenkin tilanteita, jossa minimituen ehto on liian laaja, tai ei tuota kaikkea mielenkiintoista tietoa.

Jotta sekvenssihahmojen louhintaa on saatu sovellettua uusille alueille, on alkuperäistä sekvenssihahmojen louhinnan ongelmaa myöhemmin edelleen täydennetty, joko lisäämällä rajoitteita sekvenssihahmojen esiintymiseen tiedossa, tai tarkentamalla sekvenssihahmojen muotoa ja määritelmää (Masseglia & al., 2005).

Pei & al. (2002) jakavat sekvenssihahmojen louhinnassa käytetyt rajoitteet kahteen eri

kategoriaan: (1) rajoitteisiin, jotka eivät liity tuen laskentaan, vaan toistuviin sekvenssihahmoihin itseensä, ja (2) rajoitteisiin, jotka liittyvät tukeen, eli ts. niitä käytetään määrittämään, kuinka jokin sekvenssi sopii sekvenssihahmoon.

Ensimmäisen kategorian rajoitteita ovat:

1. Tietoalkiorajoitteet (item constraint), jotka määrittävät, mitä ovat ne tietoalkiot tai tietoalkiojoukot, jotka otetaan huomioon sekvenssihahmojen louhinnassa. Esimerkiksi web-lokia käsiteltäessä halutaan ainoastaan löytää ne sekvenssihahmot, jotka liittyvät online-kirjakaupan vierailuihin.
2. Pituusrajoitteet (length constraint), jotka määrittävät halutun sekvenssihahmon pituuden, jossa pituus voi olla joko tietoalkion esiintymistiheys tai transaktioiden lukumäärä. Pituusrajoite voidaan määritellä myös erillisten tietoalkioiden lukumääränä, tai tietoalkioiden maksimaalisena lukumääränä transaktiossa. Esimerkiksi biosekvenssianalyysissa käyttäjä voi haluta löytää ainoastaan ne sekvenssihahmot, jotka käsittävät vähintään 50 transaktiota.
3. Superhahmo (super-pattern) -rajoitetta käytetään hyväksi haluttaessa löytää ne hahmot, jotka sisältävät tietyn joukon alihahmoja (sub-patterns).
4. Aggregaattirajoitteella (aggregate constraint) tarkoitetaan rajoitetta, jossa alkiota käsitellään jollain matemaattisella funktiolla, kuten summa, keskiarvo, minimi, maksimi, keskihajonta, jne. Esimerkiksi markkina-analyysissa voidaan haluta löytää ne sekvenssihahmot, jossa hahmon kaikkien alkioiden keskimääräinen arvo on yli 1000 EUR.
5. Säännöllisen lausekkeen muodostamat rajoitteet (regular expression constraints) ovat rajoitteita, jossa jollain säännöllisellä lausekkeella rajoitetaan alkioiden esiintyminen hahmossa.

Toiseen kategoriaan kuuluvia rajoitteita voidaan määritellä ainoastaan sellaisiin tietovarastoihin, joissa jokaisella transaktiolla on oma aikaleima. Toisen kategorian rajoitteita ovat:

1. Kestoon liittyvät rajoitteet (duration constraints), jotka määrittävät, että hahmon täytyy esiintyä tietokannassa usein, ja siten, että ensimmäisen ja viimeisen transaktion välinen aika ei ole suurempi tai pienempi kuin etukäteen määritelty rajoite.
2. Aikaväleihen liittyvät rajoiteet (gap constraints), jotka määrittävät, että kahden transaktion välinen aikaväli ei saa olla määritettyä arvoa suurempi.

Esimerkki kestoon ja aikaväleihin liittyvästä rajoitteesta on Srikantin ja Agrawalin (1996) esittelemät *minimaalisen* ja *maksimaalisen aikavälin* (minimal / maximal time period), sekä *liukuvan aikaikkunan* (sliding time window) rajoiteet.

Minimaalisen ja maksimaalisen aikavälin rajoitteella tarkoitetaan etukäteen elementtien (tietoalkioiden tai tietoalkiojoukkojen) esiintymisen välille asetettua aikarajaa. On esimerkiksi mahdollista, ettei kirjakaupalle ole oleellista merkitystä, ostiko joku asiakas ensin ’Säätiön’ ja vasta kolme vuotta myöhemmin ’Säätiön ja imperiumin’. Joissain tapauksissa on siis hyödyllistä pystyä asettamaan ehto, jossa sekvenssi tukee sekvenssihahmoa ainoastaan, mikäli sekvenssin tapahtumat ovat tietyn aikavälin sisällä. Monissa tapauksissa on epäolennaista esiintyykö sekvenssihahmon alkiot kahdessa erillisessä transaktiossa, jos vain tapahtuman ajankohdat sijoittuvat suhteellisen pieneen aikaikkunaan. Jokainen hahmon elementti voi siis sisältyä transaktioiden unioniin, jos vain pienimmän ja suurimman transaktion ajankohta mahtuu liukuvaan aikaikkunaan.

Toisen kategoriaan kuuluvia rajoitteita, eli sekvenssihahmojen määritelmään liittyviä rajoitteita, esittelivät Srikant ja Agrawal (1996). Srikantin ja Agrawalin (1996) mukaan alkuperäinen ongelmanmääritys ei ota huomioon käyttäjän määrittelemiä luokitteluja. Tietojen luokittelu on keskeinen tiedonhallinnan piirre, ja tietokannoissa on usein etukäteen määriteltyjä luokitteluja, joihin yksittäinen tietoalkio sijoitetaan. Toistuvien sekvenssihahmojen etsinnän kannalta on siis myös mielenkiintoista etsiä hahmoja eri luokkien välillä. Kirjaesimerkissä sekvenssihahmoja voidaan etsiä esimerkiksi kirjailijan tai kirjastoluokituksen mukaan (Kum & al., 2003).

Mannila & al. (1997) tutkivat *toistuvia jaksoja sekvensseissä* (frequent episodes in sequences). Jaksot ovat sellaisten tapahtumien asyklisiä verkkoja, joiden solmut spesifioivat ajallisen ennen ja jälkeen -suhteen ilman ajallisia rajoitteita. Pinto & al. (2001) esittelivät menetelmän moniulotteisen ja monitasoisen tiedon upottamiseen transformoituun sekvenssitietokantaan sekvenssihahmojen louhintaa varten.

Suljetulla sekvenssihahmolla tarkoitetaan sekvenssihahmoa, joka ei sisälly mihinkään muuhun sekvenssihahmoon, jolla on täsmälleen sama tuki (Yan & al., 2003).

Tietokantojen kasvaessa sekvenssihahmojen inkrementaalinen louhinta on oleellista, sillä aikaisemmat sekvenssihahmot voivat olla epäkelpoja, ja uusia voi ilmaantua. Inkrementaalisessa sekvenssihahmojen louhinnassa etsitään mahdollista tukea vanhoille sekvenssihahmoille ainoastaan uusien lisättyjen tietojen avulla käyttäen hyväksi vanhoja sekvenssihahmon tietoja (Cheng & al., 2004).

Monissa sovelluksissa, kuten laskennallisessa biologiassa, hahmot, jotka eivät ole toistuvia, voivat siitä huolimatta olla merkityksellisiä, jos niiden esiintymistiheys ylittää aikaisemman oletuksen. Perinteinen mittaustapa, kuten minimituki, ei välttämättä ole ideaalinen malli yritettäessä löytää *yllättäviä sekvenssihahmoja* (surprising patterns). Minimituki kohtelee kaikkia hahmoja tasa-arvoisesti, ja jokaisella hahmolla on sama painoarvo määriteltäessä merkittävää sekvenssihahmoa. Painoarvo ei riipu hahmon esiintymistodennäköisyydestä. Yang & al. (2001) esittelivät *infromaatiosaannon* (information gain) käyttöä sekvenssihahmojen louhinnassa. Sen avulla mitataan sekvensseihin sisältyvien hahmojen yllättävyyttä hahmon kaikkien esiintymisten yllättävyyden asteen kumulatiivisena summana.

Alkuperäinen sekvenssihahmojen louhinnan ongelma käsittelee tietoalkioiden esiintymistä tietokannassa ainoastaan kaksiarvoisesti - tietoalkio joko esiintyy tai ei esiinny. Tällä voidaan katsoa olevat kaksi rajoitusta (Ouyang & Huang, 2007): (1) ne eivät ota huomioon kvantitiivisia attribuutteja, kuten tietoalkioiden lukumäärää, ja (2) ne ottavat huomioon ainoastaan suorat sekvenssihahmot (direct sequential patterns), joilla tarkoitetaan

sekvenssihahmoja, joiden tietoalkiot ovat täsmälleen samoja. Sumeiden sekvenssihahmojen louhinnan tutkimuksessa kehitetään menetelmiä, joilla näitä rajoituksia pyritään kiertämään.

2.4 Sekvenssihahmojen louhintamenetelmät

Tietokantojen *tiheydellä* (database density) tarkoitetaan olemassa olevien hahmojen ja mahdollisten sekvenssien suhdetta (Antunes & Oliveira, 2004). Käytännössä tietokannan tiheys riippuu käytetystä tuesta, sillä tiheys on sitä suurempi, mitä alhaisempi minimituki on, koska silloin toistuvia sekvenssejä on enemmän. Toinen tiheyteen vaikuttava parametri on tietoalkiojoukkojen lukumäärä tietokannassa. Suurempi alkioiden lukumäärä johtaa potentiaalisesti suurempaan määrään erilaisia sekvenssejä, koska monien erilaisten sekvenssien esiintyessä tietokannassa niiden todennäköisyys olla usein toistuva on alhainen. Toisaalta taas alhaisempi alkioiden lukumäärä tuottaa pienemmän määrän potentiaalisia sekvenssejä, jotka toistuvat useammin tietokannassa, ja siten kasvattavat hahmojen lukumäärää ja tietokannan tiheyttä. Potentiaalisesti suuresta sekvenssien lukumäärästä huolimatta vain pienellä osajoukolla on tarpeeksi suuri tuki tietokannassa. Nämä vastakkaisuudet huomioon ottaen sekvenssihahmojen louhinnan ongelma on siinä, mitä sekvenssejä kannatta kokeilla, ja kuinka tehokkaasti löytää ne, jotka ovat usein toistuvia.

Sekvenssihahmojen louhintaan on tarjolla useita algoritmeja. Algoritmit poikkeavat kuitenkin toisistaan ratkaisuperiaatteiltaan ja tehokkuudeltaan. Ratkaisuperiaatteiden erot johtuvat usein siitä, mitä sekvenssihahmojen määritelmää ajatellen ne on kehitetty, ja kuinka ongelmanasettelussa on otettu huomioon erilaiset rajoitteet. Siksi eri menetelmät ja algoritmit löytävät aina myös hieman toisistaan poikkeavia sekvenssihahmoja.

Sekvenssihahmojen louhintaan käytetyt algoritmit voidaan ratkaisuperiaatteensa mukaan jakaa kahteen kategoriaan (Han & Kamber, 2001):

1. Aprioripohjaisiin algoritmeihin (apriori-based algorithms), jota perustuvat hahmokandidaattien generointiin ja niiden pätevyyden testaamiseen. Näistä ensim-

mäinen oli Agrawalin ja Srikantin (1995) esittelemä AprioriAll-algoritmi, joka esitellään tarkemmin seuraavassa luvussa. AprioriAll-algoritmin toteutus XSLT 2.0 -käsittelykielellä sekvenssihahmojen louhintaan XML-muotoisesta tiedosta esitetään luvussa 4.

2. Hahmonkasvatusmenetelmään (pattern-growth method), jotka niin ikään käyttävät hyväkseen apriori-periaatetta, mutta ratkaisevat ongelman soveltamalla hajoita ja hallitse -periaatteetta sekä louhintaan että tietokantaan. Keskeinen idea on välttää kandidaattien generointi ja testaus kokonaan, ja keskittää etsintä rajoitettuun alkuperäisen tietokannan osaan. Yleisesti hahmonkasvatusmenetelmät voidaan nähdä DFS-puun (depth-first search tree) läpikäyntialgoritmeina, sillä ne muodostavat jokaisen etsityn hahmon erikseen rekursiivisesti. Esimerkki hahmonkasvatusmenetelmään perustuvasta algoritmista on PrefixSpan (Pei & al., 2001a).

Hahmonkasvatusmenetelmät ovat apriori-menetelmiä tehokkaampia, kun läpikäytävän tietokannan tiheys on suuri. Sen sijaan harvoissa tietokannoissa apriori-pohjaiset menetelmät ovat tehokkaampia. Kun tuki puolestaan pienenee, ovat hahmonkasvatusmenetelmät taas apriori-pohjaisia tehokkaampia (Pei & al., 2001b). Kummassakin lähestymistavassa on siis suhteellisia vahvuuksia ja heikkouksia toiseen verrattuna, ja kumpikin tuottaa erilaisia ongelmia erilaisissa tilanteissa. Ensinnäkin kumpikin tapa vaatii suuren määrän muistia, ja etukäteen vaaditun muistin määrää on vaikea ennustaa. Apriori-menetelmässä muistinkulutuksen kannalta keskeinen ongelma on kandidaattien suuri määrä, ja hahmonkasvatusmenetelmässä rekursion vaatima tila. Toiseksi todellisissa tilanteissa sama tietokanta voi olla tiheä tai väljä riippuen näkökulmasta. Esimerkiksi teletietojen analysoinnissa yksityispuhelujen muodostamat hahmot voivat olla täysin erilaisia yrityspuheluiden muodostamiin hahmoihin verrattuna, koska ne ovat luonteeltaan erilaisia.

Edellä esitetyt sekvenssihahmojen louhintaan käytetyt menetelmät ja algoritmit on alun perin suunniteltu relaatiomalliin tallennetun tiedon läpikäyntiin. Nykyaikaisissa tietojärjestelmissä XML:ää käytetään hyväksi kaikissa osa-alueissa tiedon esityksestä tiedon tal-

lennukseen ja tiedonsiirtoon. XML:n joustavuus sekä rakenteeltaan että semantiikaltaan lisää tiedonlouhinnan haastavuutta, eikä edellä esitettyjen menetelmien soveltuvuus sekvenssihahmojen louhintaan XML-muotoisesta tiedosta ole itsestään selvää. Luvussa 3 perehdytään tarkemmin sekvenssihahmojen louhintaan XML-muotoisesta tiedosta.

2.5 Apriori-periaate ja AprioriAll-algoritmi

Tietokannan kaikkien alkioiden keskinäisten suhteiden ja niiden kombinaatioiden lukumäärä tietokannassa voi olla erittäin suuri. Agrawalin ja Srikantin (1994) esittelemässä assosiaatiosääntöjen johtamiseen tarkoitetussa Apriori-algoritmissa käytettiin ensimmäisenä hyväksi toistuvien tietoalkiojoukkoihin sisältyvää *alaspäin suunnatun sulkeuman* (downward closure property) ominaisuutta. Keskeisenä ideana on, että tietokannan läpikäynnissä voidaan sulkea pois osa käsitellyistä tietoalkiojoukoista niiden alaspäin suunnatun sulkeuman ominaisuuden, eli apriori-ominaisuuden, perusteella.

Apriori-ominaisuus tarkoittaa, että tietoalkiojoukko, jonka jokin alijoukko ei ole usein esiintyvä, ei voi itsekään olla usein esiintyvä tietoalkiojoukko (Agrawal & Srikant 1994). Tämän perusteella toistuvia ja erikokoisia tietoalkiojoukkoja voidaan löytää etsimällä ensin kaikki toistuvat yhden tietoalkion kokoiset tietoalkiojoukot, generoida niistä kandidaatit kahden tietoalkion kokoisille ja tarkastaa, mitkä kandidaateista löytyvät tietokannasta, ja generoida niistä edelleen kandidaatit kolmen tietoalkion kokoisille tietoalkiojoukoille jne. Prosessia jatketaan niin kauan, kunnes *k*-alkiota sisältäviä tietoalkiojoukkoja ei enää löydy.

Agrawalin ja Srikantin (1995) esittelemä AprioriAll-algoritmi käyttää samaa periaatetta toistuvien sekvenssihahmojen etsintään. Ratkaisuperiaate koostuu viidestä vaiheesta: lajitteluvaihe, suurten tietoalkiojoukkojen generointivaihe, muunnosvaihe, sekvenssivaihe ja enimmäissekvenssien, eli maksimaalisten sekvenssien vaihe.

2.5.1 Lajitteluvaihe

Kuvassa 1 on alkuperäinen tietokanta, johon transaktiot on tallennettu tapahtumajärjes-

tyksessä (Transaction Time).

Transaction Time	Customer Id	Items Bought
June 10 '93	2	10, 20
June 12 '93	5	90
June 15 '93	2	30
June 20 '93	2	40, 60, 70
June 25 '93	4	30
June 25 '93	3	30, 50, 70
June 25 '93	1	30
June 30 '93	1	90
June 30 '93	4	40, 70
July 25 '93	4	90

Kuva 1. Alkuperäinen tietokanta (Agrawal ja Srikant, 1995).

Lajitteluvaiheessa (sort phase) tietokanta lajitellaan käyttäen ensisijaisena avaimena asiakkaan yksilöivää tunnistetta, ja toissijaisena avaimena transaktion tapahtuma-aikaa. Lajittelun tuloksena syntyy uudelleen järjestetty tietokanta, joka sisältää asiakkaittain tapahtumajärjestyksessä olevat transaktiot, eli toisin sanoen asiakassekvenssit. Kuvassa 2 on asiakastunnuksen (Customer Id) ja transaktioajankohdan mukaan lajiteltu tietokanta, ja kuvassa 3 asiakassekvensseiksi muutettu tietokanta.

Customer Id	TransactionTime	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10, 20
2	June 15 '93	30
2	June 20 '93	40, 60, 70
3	June 25 '93	30, 50, 70
4	June 25 '93	30
4	June 30 '93	40, 70
4	July 25 '93	90
5	June 12 '93	90

Kuva 2. Lajiteltu tietokanta (Agrawal ja Srikant, 1995) .

Customer Id	Customer Sequence
1	⟨ (30) (90) ⟩
2	⟨ (10 20) (30) (40 60 70) ⟩
3	⟨ (30 50 70) ⟩
4	⟨ (30) (40 70) (90) ⟩
5	⟨ (90) ⟩

Kuva 3. Asiakassekvenssiversio tietokannasta (Agrawal ja Srikant, 1995).

2.5.2 Suurten tietoalkiojoukkojen generointivaihe

Suurten tietoalkiojoukkojen generointivaiheessa (litemset phase) etsitään kaikki tietokantaan sisältyvät *1,2,…,k* -tietoalkion kokoiset tietoalkiojoukot. Tässä vaiheessa tietoalkiojoukkojen etsintään voidaan käyttää hyväksi Agrawalin ja Srikantin (1994) assosiaatiosääntöjen generointiin esittämää Apriori-algoritmia. Tietoalkiojoukon tuki määritellään kuitenkin niiden asiakkaiden osamääränä kaikista asiakkaista (tai absoluuttisena asiakkaiden lukumääränä), joiden transaktiossa tietoalkiojoukko esiintyy, toisin kuin assosiaatiosääntöjen generoinnin yhteydessä, jossa tuki määritellään suoraan tietoalkiojoukon esiintymiskertojen mukaan. Tietoalkiojoukon saamaa tukea siis kasvatetaan ainoastaan kerran asiakasta kohden, vaikka sama tietoalkiojoukko esiintyisi useammassa asiakkaan transaktiossa.

Suuret tietoalkiojoukot sijoitetaan kasvavaan kokonaislukuun. Tämän ansioista jokainen suuren tietoalkiojoukon vertailu voidaan koosta riippumatta tehdä vakioajassa, koska jokaista tietoalkiota ei erikseen tarvitse verrata keskenään. Kuvassa 4 on kuvattu kaikki tietoalkiojoukot, jotka esiintyvät vähintään kahdella asiakkaalla, sekä tietoalkiojoukkojen sijoitus kokonaislukuun (Mapped To).

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

Kuva 4. Suuret tietoalkiojoukot (Agrawal ja Srikant, 1995).

2.5.3 Muunnosvaihe

Jokaisen suuren tietoalkiojoukon mahdollinen sisältyminen asiakassekvenssiin on testattava erikseen. Jotta testaus voidaan suorittaa mahdollisimman nopeasti, jokainen asiakassekvenssi muunnetaan toisenlaiseen esitysmuotoon. Tämä tehdään muunnosvaiheessa (transformation phase). Kaikkien asiakassekvenssien transaktioiden tietoalkiojoukot korvataan kaikkien niiden suurten tietoalkiojoukkojen joukolla, jotka ko. transaktioon sisältyvät. Jos transaktio ei sisällä yhtään suurta alkiojoukkoa, sitä ei oteta mukaan muunnettuun asiakassekvenssiin. Se kuitenkin on edelleen mukana laskettaessa asiakkaiden kokonaismäärää. Asiakassekvenssi esitetään muunnoksen jälkeen listana, joka koostuu suurten tietoalkiojoukkojen joukoista. Suuret tietoalkiojoukot esitetään muodossa $\{l_1,l_{2,\ldots,}l_n\}$, jossa jokainen l_i on suuri tietoalkiojoukko.

Kuvassa 5 on esitetty muunnosvaiheen tulokset. Toisessa sarakkeessa (Original Customer Sequence) ovat asiakkaan alkuperäiset sekvenssit. Kolmannessa sarakkeessa ovat muunnetut asiakassekvenssit (Transformed Customer Sequence), jossa jokainen alkuperäinen asiakassekvenssi on korvattu siihen sisältyvien suurten tietoalkiojoukkojen joukolla. Esimerkiksi asiakkaan, jonka asiakastunnus on 4, sekvenssi <(30) (40 70) (90)> on korvattu sekvenssillä <{(30)} {(40), (70), (40 70)} {(90)}>. Sekvenssin toinen alkio (40 70) sisältää suuret alkiojoukot (40), (70) ja (40 70). Toisaalta asiakkaan 2 muunnettu asiakassekvenssi <{(30)} {(40), (70), (40 70)}> ei sisällä alkuperäisen asiakassekvenssin sekvenssiä (10 20), koska se ei sisällä yhtään suurta tietoalkiojoukkoa. Kuvan 5 neljännessä sarakkeessa (After Mapping) on muunnosvaiheen lopullinen tulos, jossa jokainen muun-

nettu asiakassekvenssi on korvattu sille sijoituksessa annetulla kokonaisluvulla.

Customer Id	Original Customer Sequence	Transformed Customer Sequence	After Mapping
1	⟨(30) (90)⟩	⟨{(30)} {(90)}⟩	⟨{1} {5}⟩
2	⟨(10 20) (30) (40 60 70)⟩	⟨{(30)} {(40), (70), (40 70)}⟩	⟨{1} {2, 3, 4}⟩
3	⟨(30 50 70)⟩	⟨{(30), (70)}⟩	⟨{1, 3}⟩
4	⟨(30) (40 70) (90)⟩	⟨{(30)} {(40), (70), (40 70)} {(90)}⟩	⟨{1} {2, 3, 4} {5}⟩
5	⟨(90)⟩	⟨{(90)}⟩	⟨{5}⟩

Kuva 5. Muunnettu tietokanta (Agrawal ja Srikant, 1995).

2.5.4 Sekvenssivaihe

Sekvenssivaihe (sequence phase) toteutetaan seuraavan algoritmin avulla:

L_1 = {suuret 1-tietoalkiojoukot}//suurten tietoalkiojoukkojen generointivaiheen tulos
for (k = 2; $L_{k-1} \neq \emptyset$; k++) **do**
    begin
    C_k = L_{k-1} tietoalkiojoukoista generoidut uudet kandidaatit
    foreach tietokannan asiakassekvenssi c **do**
        Kasvata kaikkien niiden kandidaattijoukkoon C_k sisältyvien kandidaattien laskuria, jotka sisältyvät asiakassekvenssiin c
    L_k = Kandidaatit C_k:ssa, joilla vähimmäistuki
    end
Tulos = Maksimaaliset sekvenssit $\bigcup_k L_k$:ssa

Jokaisella läpikäynnin kierroksella käytetään edellisen kierroksen suuria tietoalkiojoukkoja kandidaattien generointiin, ja sen jälkeen lasketaan niiden tuki koko tietokannassa. Läpikäynnin lopussa kandidaattien tuen avulla määritetään, mitkä kandidaateista kuuluvat ko. suureen tietoalkiojoukkoon. Ensimmäisellä kierroksella käydään läpi suurten tapahtumajoukkojen generointivaiheen tuottama yhden tietoalkion kokoiset tietoalkiojoukot. Kuvassa 6 (johdettu kuvasta 5) on esitetty kaikki yhden alkion kokoiset tietoalkiojoukot, sekä niille laskettu tuki.

Sequence	Support
⟨ 1 ⟩	4
⟨ 2 ⟩	2
⟨ 3 ⟩	3
⟨ 4 ⟩	2
⟨ 5 ⟩	3

Kuva 6. Yhden tietoalkion tietoalkiojoukot.

Apriori-kandidaattien generointifunktio saa parametrina tietoalkiojoukon *L(k-1)*, eli kaikki *k-1* kokoiset suuret tapahtumajoukot. Ensimmäisessä vaiheessa tehdään liitos tietoalkiojoukkojen *L(k-1)* ja *L(k-1)* välillä seuraavasti:

INSERT
into C_k
SELECT
p.litemset$_1$, p.litemset$_2$,…,p.litemset$_{k-1}$,q.litemset$_{k-1}$
FROM
L_{k-1} p, L_{k-1} q
WHERE
p.litemset$_1$ = q.litemset$_1$,…,p.litemset$_{k-2}$ = q.litemset$_{k-2}$

Kuvassa 7 on kaikki kahden tietoalkion kokoiset tietoalkiojoukot, sekä tuki asiakkaiden lukumääränä. Tuki on laskettu kuvassa 5 esitetyn muunnetun tietokannan perusteella. Mukana ovat vain ne tietoalkiojoukot, joilla on vähintään kahden asiakkaan tuki.

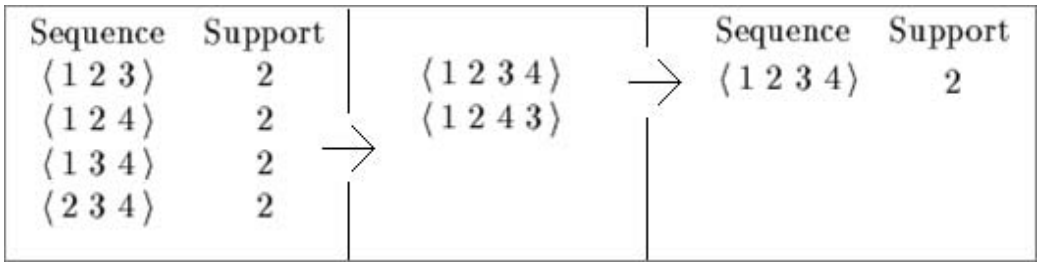
Sequence	Support
⟨ 1 2 ⟩	2
⟨ 1 3 ⟩	3
⟨ 1 4 ⟩	2
⟨ 1 5 ⟩	2
⟨ 2 3 ⟩	2
⟨ 2 4 ⟩	2
⟨ 3 4 ⟩	2

Kuva 7. Kahden alkion kokoiset tietoalkiojoukot.

Seuraavaksi poistetaan kaikki sekvenssit $c \in C_k$, jossa jokin c:n k - 1 alisekvenssi ei sisälly tietoalkiojoukkoon L_{k-1}.

forall sekvenssi $c \in C_k$ **do**
    forall sekvenssin c (k - 1)-alisekvensseille s **do**
        if ($s \notin L_{k-1}$) **then**
            poista c kandidaattijoukosta C_k;

Kuvassa 8 on ensimmäisessä sarakkeessa esitetty kaikki generoidut kolmen tietoalkion kokoiset tietoalkiojoukot, joita tukee vähintään kaksi asiakasta. Toisessa sarakkeessa on esitetty niiden liitoksen tuloksena syntyneet neljän tietoalkion kokoiset tietoalkiojoukot. Viimeisessä sarakkeessa on esitetty puhdistettu tulos, josta on poistettu kaikki ne sekvenssit, joiden kolmen tietoalkion kokoinen alisekvenssi ei sisälly kolmen tietoalkion kokoisiin tietoalkiojoukkoihin. Sekvenssi (1 2 4 3) on poistettu kuvan 8 kolmannessa sarakkeessa esitetystä tuloksesta, sillä alisekvenssi (2 4 3) ei sisälly kolmen tietoalkion kokoisiin sekvensseihin. Tuki on laskettu kuvassa 5 esitetyn muunnetun tietokannan perusteella.



Kuva 8. Sekvenssin generointi.

2.5.5 Enimmäisekvenssien vaihe

Enimmäissekvenssien vaiheessa (maximal phase) etsitään kaikki maksimaaliset sekvenssit suurten sekvenssien joukosta. Kun kaikki suuret tietoalkiojoukot on löydetty, voidaan seuraavan algoritmin avulla löytää kaikki enimmäissekvenssit.

for (k = pisimmän sekvenssin pituus; $k > 1$; k --) **do**
    foreach k-sekvenssi s_k **do**
        poista sekvenssijoukosta S kaikki s_k:n alisekvenssit

Algoritmin suorituksen jälkeen jäljelle jäävät sekvenssit ovat <1 2 3 4> ja <1 5>. Kuvassa 9 on esitetty lopullinen tulos, jossa ovat kaikki löydetyt sekvenssihahmot, ts. maksimaali-

set asiakassekvenssit, jotka löytyvät vähintään kahdelta asiakkaalta. Sekvenssejä <(30)>, <(40)>, <(70)>, <(90)>, <(30) (40)>, <(30) (70)> ja <(40 70)> tukee vähintään kaksi asiakasta, mutta ne eivät kuulu lopulliseen vastaukseen, koska ne sisältyvät suurempaan sekvenssiin, jota tukee myös vähintään kaksi asiakasta, eli jonka tuki on suurempi kuin 25%. Ainoastaan sekvenssit <(30) (90)> ja <(30) (40 70)> ovat maksimaalisia.

Sequential Patterns with support > 25%
⟨ (30) (90) ⟩
⟨ (30) (40 70) ⟩

Kuva 9. Lopullinen tulos (Agrawal ja Srikant, 1995).

2.5.6 AprioriAll-algoritmin suorituskyvyn parantaminen

AprioriAll-algoritmin keskeinen heikkous on sen generoima suuri hahmokandidaattien määrä. Jos alkuperäisestä tietokannasta löydetään 10 000 yhden tietoalkion kokoista tietoalkiojoukkoa, niin kahden tietoalkion kokoisia tietoalkiojoukkoja generoidaan 10 000 000, joista jokaisen tuki täytyy laskea ja tallentaa kolmen tietoalkion kokoisten tietoalkiojoukkojen generointia varten. Ja jos halutaan löytää 100 tietoalkion pituiset sekvenssihahmot, niin kandidaatteja täytyy generoida $2^{100} \approx 10^{30}$ (Han & Kamber, 2001).

Toinen AprioriAll-algoritmin heikkous on sen tarvitsema koko tietokannan läpikäyntien määrä. Jos suurimman sekvenssihahmon tietoalkioiden lukumäärä on *n*, niin AprioriAll-algoritmin täytyy käydä tietokanta läpi *n* + 1 kertaa (Han & Kamber, 2001).

Apriori-menetelmien tehokkuutta voidaan parantaa seuraavien menetelmien avulla (Han & Kamber, 2001):

1. Tiedon tiivistykseen perustuvilla tietoalkiojoukkojen laskennalla (hash-based itemset counting), jossa jokainen *k*-alkion kokoinen tietoalkiojoukko, jota vastaavan hajautustaulun laskuri on tukirajan alapuolella, ei voi olla usein toistuva.

2. Transaktioita vähentämällä (transaction reduction), jolloin niitä transaktioita, jotka eivät sisällä yhtään usein toistuvaa *k*-tietoalkiojoukkoja, ei oteta mukaan seuraavissa läpikäynneissä.
3. Osittamalla (partitioning). Osituksessa tietokanta jaetaan osiin käytettävän muistikapasiteetin perusteella. Ensimmäisessä vaiheessa jokaisesta osajoukosta analysoidaan toistuvat osajoukot. Ne yhdistetään uudeksi joukoksi, josta lasketaan osituksessa löytyneiden joukkojen esiintymistodennäköisyydet. Todennäköisyyden ollessa suurempi kuin asetettu raja-arvo, hyväksytään löytynyt tietoalkiojoukko lopputulokseen.
4. Otosten muodostamisella (sampling). Menetelmässä muodostetaan otos analysoitavasta perusjoukosta ja lasketaan sen usein toistuvat osajoukot. Niiden löydyttyä lasketaan koko perusjoukosta osajoukkojen esiintymistodennäköisyydet.
5. Dynaamisella tietoalkiojoukkojen laskennalla (dynamic itemset counting). Menetelmässä uusi kandidaatti lisätään ainoastaan siinä tapauksessa, että kaikkien sen alijoukkojen arvioidaan olevan usein toistuvia.

3 SEKVENSSIHAHMOJEN LOUHINTA XML-MUOTOISESTA TIEDOSTA

Tässä luvussa esitellään keskeiset käsitteet ja ongelmat, jotka liittyvät sekvenssihahmojen louhintaan XML-muotoisesta tiedosta. Ensimmäisessä kohdassa esitellään lyhyesti XML tiedontallennusmuotona, XML-teknologiat ja XML:n sovellusalueet. Seuraavassa kohdassa käydään läpi XML-muotoiseen tietoon kohdistuvan tiedonlouhinnan erityispiirteet ja esitellään sekvenssihahmojen louhinnan ongelma XML-muotoisen tiedon yhteydessä. Lopuksi kolmannessa kohdassa esitellään XML-muotoiseen tietoon sovellettavissa olevat tiedonlouhinnan ja sekvenssihahmojen louhinnan menetelmät.

3.1 XML tiedontallennusmuotona

Perinteiseen relaatiotietokantaan tallennettu tieto on rakenteista, jolla on etukäteen määrätty skeema, joka määrittää tiedon muodon, tietoyksiköiden suhteet toisiinsa, ja tiedon sisällön rajoitteet (Buneman, 1997). Rakenteiseen tietoon suoritetaan lisäys-, päivitys-, poisto- ja kyselyoperaatiota skeemaa hyödyntävillä kyselykielillä, joista yleisin on SQL (Structured Query Language). Rakenteisten tietovarastojen rinnalle on viimeisen vuosikymmenen aikana internetin ja siihen liittyvien teknologioiden yleistymisen myötä ilmestynyt puolirakenteinen tieto. Puolirakenteisen tieto on tietoa, joka hyödyntää jotain yleistä rakennemallia (tavallisesti puu tai verkko), mutta tiedolla ei ole skeemaa, ei ainakaan sillä tarkkuudella, mitä relaatiomallin tietokannat vaativat. Lisäksi osa tiedoista (esim. kenttien nimet), jotka yleensä on kuvattu skeemassa, kuvataan puolirakenteisessa datassa itsessään. Esimerkkejä puolirakenteisesta tiedosta ovat rakenteiset dokumentit, kuten www-sivut, sähköpostit, uutisryhmäarkistot ja ohjelmakoodit.

Yleisin ja kenties tärkein rakenteinen dokumenttiformaatti on XML (eXtensible Markup Language) (W3C, 2008a). Myös XML:llä voi olla skeema, mutta se on relaatiomallia joustavampi. Relaatiomallissa sisäkkäisten tietoalkioiden ja erityisesti monesta moneen -suhteiden esittäminen voi olla työlästä, XML:ssä olennaisesti helpompaa.

XML-muodossa oleva tieto koostuu sisäkkäisistä ja peräkkäisistä elementeistä, jotka

muodostavat puolirakenteiselle tiedolle tyypillisen puumaisen rakenteen, jonka elementit ovat solmuja. Elementit ilmaistaan erityisillä merkeillä (< ja />), jotka erottavat rakenteen tekstimuodossa olevasta informaatiosta. Puun solmu puolestaan sisältää alisolmujen lisäksi tekstiä sekä mahdollisesti attribuutteja. Elementit muodostavat dokumentin loogisen rakenteen. Attribuuteilla puolestaan merkitään yksittäisiin elementteihin liittyvää metatietoa. Lisäksi dokumentilla on entiteeteistä (entity) muodostuva fyysinen rakenne. Entiteetit voivat olla mitä tahansa tietoyksiköitä, kuten yksittäisiä merkkejä, dokumenttifragmentteja tai viittauksia toisiin tiedostoihin. XML:n elementit ja attribuutit ovat XML:n laatijan vapaasti valittavissa (Hunter & al., 2007).

XML:n tekstipohjaisuus on yksi sen keskeisistä eroista muihin binäärisiin tiedon tallennusmuotoihin. XML-muotoista tietoa voidaan lukea ja muotoilla tekstinkäsittelyn mahdollistavilla työvälineillä, vaikka työvälineessä ei varsinaista XML-tukea olisikaan.

3.1.1 XML-käsittelykielet

Termillä XML ei useinkaan tarkoita pelkästään XML-kieltä, vaan koko joukkoa World Wide Web Consortiumin (W3C, 2008b) kehittämiä XML-pohjaisia teknologioita, tekniikoita ja käsittelykieliä. Teknologioita ja tekniikoita ovat mm. XML-dokumentin rakenteen määritykseen tarkoitettu DTD (Document Type Definintion), XML-skeema, järjestelmien väliseen viestintään käytetyt web-palvelutekniikka ja SOAP (Simple Object Access Protocol). XML-muotoisen tiedon käsittelyyn tarkoitetut käsittelykielet ovat XQuery (W3C, 2008c), sekä tämän tutkielman kannalta keskeisimmät XPath (W3C, 2008d) ja XSLT (W3C, 2008e).

XPath (XML Path Language) on ei-XML-pohjainen merkintäkieli, jolla osoitetaan XML-dokumenttien osiin ja käytetään hyväksi luotaessa XML-dokumentin rakenteeseen perustuvaa tietoa. XPath-lauseilla poimitaan halutut asiat käsiteltävästä XML-dokumentista tulokseen. XPath-lausekkeella voidaan XML:stä erottaa elementtisolmujen joukko. XPathin määrittämillä funktioilla voidaan taas evaluoida jonkin lausekkeen totuusarvo, tai erottaa sen sisältämä numeerinen tai tekstuaalinen arvo. Elementtisolmujen joukkoja il-

maisevaa lauseketta kutsutaan *valintalausekkeeksi* (location path), joka puolestaan muodostuu sarjasta *valinta-askeleita* (location step). Jokaisen valinta-askeleen valintaehto on kolmeosainen: (1) haara (axis), joka rajaa valinnan lähtösolmun suhteen määriteltävään osaan puuta, (2) solmu (node), joka rajaa valinnan tietynlaiseen solmuun joko tyypin tai nimen mukaan, sekä (3) mahdolliset ehtolausekkeet, jotka asettavat lisäehtoja valittaville solmuille (Kay, 2004a). Jokainen valinta-askel tuottaa tuloksenaan järjestetyn solmujoukon, ja seuraava askel jatkaa valintaa edellisen askelen tulosjoukosta eteenpäin, soveltamalla seuraavaa valintaehtoa, jonka lähtökohtana on vuorollaan kukin tulosjoukon solmu.

XSLT 2.0 (Extensible StyleSheet Language Transformations) on ohjelmointikieli, jonka avulla XML-muotoista tietoa voidaan käsitellä muuntamalla sitä muodosta toiseen. XSLT 2.0 on XML-syntaksia noudattava deklaratiivinen kieli, jonka perustavana ovat muunnossäännöt. Muunnossäännöt määritellään attribuuteilla (Kay, 2004b):

1. Kohde (match), joka määrittää kohteen, johon muunnossääntöä sovelletaan.
2. Nimi (name), jota käytetään haluttaessa aktivoida jollain muulla attribuutilla määritelty muunnossääntö.
3. Moodi (mode), joka nimeää ympäristön, jossa sääntöä sovelletaan.
4. Prioriteetti (priority), joka määrittää konfliktoivien sääntöjen valinnassa käytettävän painoarvon.

Sääntöjen kohteena olevat solmut määritellään XPath-valintalausekkeilla. XSLT 2.0:ssa käytetään template-rakennetta muunnosten suorittamiseen. *Template* on mallipohja, jossa luodaan elementtejä templatessa määriteltyjen sääntöjen ja syötteenä saatujen parametrien pohjalta.

Vaikka XSLT 2.0 on ensisijaisesti tarkoitettu XML-muotoisen tiedon esitysmuodon määrittelyyn, käytetään sitä useasti myös yksinkertaisiin tietokantaoperaatioiden tapaisiin toimintoihin, ja myös muihin sellaisiin tarkoituksiin, jotka eivät vaadi kaikkia yleisten ohjelmointikielten ominaisuuksia (Møller & al., 2005). Termiä *tyylitiedosto* (stylesheet) käytetään usein kuvaamaan muunnosta, joka kuvataan XSLT-käsittelykielellä. Tyypillisiä

XSLT:n sovelluskohteita ovat XML-aineiston muunnokset toiseen XML-formaattiin, HTML-sivuiksi ja monikäyttöiseen, myös arkistointiin soveltuvaan PDF-muotoon (Kay, 2004b).

XQuery 1.0 on yleinen käsittelykieli, jolla voi hakea tietoa XML-muotoisesta tiedoista, muokata tiedon tulostusta, sekä palauttaa vastaus uutena XML-dokumenttina. XQuery käyttää XPath-valintalausekkeita, ja pystyy myös hyödyntämään XPathin funktioita. XQuery on alun perin tarkoitettu SQL:n tapaiseksi kyselykieleksi XML-pohjaisiin tietokantoihin (Brundage, 2004). Sekä XSLT 2.0 että XQuery 1.0 ovat tällä hetkellä W3C 'recommendation' -statuksen omaavia. Molemmat kielet ovat samanlaisia ilmaisukyvyltään, mutta XSLT 2.0:ssä on monia sellaisia ominaisuuksia, jotka puuttuvat XQuery 1.0:sta. Näitä ovat esimerkiksi elementtien ryhmittely ja numeroiden ja päivämäärien formatointi (Kay, 2004b).

3.1.2 XML:n sovellusalueet

XML:n laajaa levinneisyyttä selittää ennen kaikkea se, että XML:ää ei ole tarkoitettu ainoastaan johonkin rajattuun sovellusalueeseen. XML tarjoaa tietorakenteen, jota voidaan laajentaa käytännössä kaikenlaisen tiedon kuvaamiseen, ja liitännäisteknologioita, jota voidaan itse soveltaa (Hunter & al., 2007). Yksinkertaisimmillaan XML soveltuu HTML:n korvaajaksi, koska kaikki, mitä HTML:llä voidaan tehdä, voidaan tehdä myös XML:llä. XML on kuitenkin yleisempi metakieli, joten se ei ole rajoittunut ainoastaan esitettäväksi tarkoitettujen dokumenttien merkintään, vaan XML-dokumentteja voi käyttää miltei mihin tahansa tiedon varastoitiin, kuvaamiseen ja siirtoon liittyvään. XML:n korkean abstraktiotason ansiosta XML dokumentit voivat sisältää mitä tahansa sellaista tietoa, mitä viittauksia sisältävällä puumaisella tietorakenteella voidaan ylipäänsä mallintaa. Tämä sisältää myös perinteisen relaatiomallin mukaisen tietorakenteen, ts. tietueista koostuvat relaatiot.

XML tarjoaa mahdollisuuden dokumenttien täsmälliseen tyypittämiseen, joten se soveltuu tiedon esitysmuodon standardointiin, järjestelmien välisen kommunikaation formaa-

tiksi, sekä tiedon arkistointitekniikaksi. Lisäksi XML on periaatteessa laite- ja ohjelmistoriippumatonta. XML-ohjelmien ja XML-prosessorirajapintojen ansiosta XML toimii myös pohjana standardoitujen kehysten ja toisen tason metakielten määrittelyyn eri sovelluksiin (Hunter & al., 2007).

XML-perustekniikan pulmina voidaan pitää esim. tekstimuotoisuutta ja monisanaisuutta, rajoittuneita merkkaus- ja tyyppimäärityskieliä sekä käsittelyn "hitautta" (verrattuna optimoituihin relaatiotietokantoihin) (Boldakov & Grinev, 2006).

3.1.3 XML tietokannoissa

Alun perin tietokannoissa ei ollut erillistä tukea XML:lle, jolloin kakki XML-muotoinen tieto, joka tallennettiin tai haettiin tietokannasta, täytyi ensin muuntaa relaatiomalliin sopivaan muotoon. XML:n käytön yleistyessä tietokantaohjelmistojen valmistajat tarjosivat kukin erilaista XML-tukea, joiden avulla XML:ää oli mahdollista ohjelmistoja tai kyselyitä tietyllä tavalla konfiguroimalla generoida suoraan tietokantakyselyistä, ja tallentaa XML-muotoinen tieto tietokantojen tauluihin (Wilde, 2006).

XML:n edelleen yleistyessä syntyi SQL/XML-standardi, jonka mukana tuli XML-tietotyyppi ja XML-muotoista tietoa oli mahdollista tallentaa suoraan tietokantaan, ja saada se suoraan SQL-kyselyn tuloksena. Seuraava askel XML-muotoisen tiedon tallennuksessa tietokantoihin ovat täysiveriset XML-tietokannat, eli natiivit XML-tietokannat. XML-tietokannat ovat erityisiä XML-dokumenttien tiedostojärjestelmiä, jotka on optimoitu XML-muotoisen tiedon tallennukseen ja hakuun. Tämä askel on vielä laajassa mittakaavassa toteutumaton, sillä natiivin XML-tietokannan kehityksessä on edelleen avoimia tutkimusaiheita, jotka liittyvät tietokannan rakenteeseen ja käytettävään käsittelykieleen (Wilde, 2006). Esimerkiksi XML-skeema on tarkoitettu XML-dokumenttien skeeman kuvauskieleksi, mutta ei tietokantojen skeeman kuvauskieleksi. Keskeisenä puutteena XML-skeemakielessä on eheyssääntöjä määrittävien piirteiden olemattomuus. Yleisimmästä XML-muotoisen kielen käsittelykielistä XQuery:stä puuttuu puolestaan päivitysominaisuudet.

3.2 Tiedonlouhinta XML:stä

XML:n puolirakenteisesta olemuksesta johtuen osa tiedosta on XML:n rakenteessa, ja osa sen tietosisällössä. Ennen kuin sekvenssihahmoja ryhdytään louhimaan XML-muotoisesta tiedosta, on ensin määriteltävä, mitä sekvenssihahmojen louhinnalla, ja yleisemmin tiedonlouhinnalla, XML-muotoisesta tiedosta itse asiassa tarkoitetaan. Nayak & al. (2002) jakavat suoraviivaisesti XML-tiedonlouhinnan kahteen eri kategoriaan: XML-tiedon rakenteen ja tietosisällön louhintaan.

3.2.1 XML:n rakenteen louhinta

XML:n elementit ja niiden sisäinen järjestys muodostavat XML-muotoisen tiedon rakenteen. XML-rakenteen louhinta on siis XML:n rakenteen määrittelyyn kohdistuvaa tiedonlouhintaa. Nayak & al. (2002) jakavat XML-rakenteen louhinnan edelleen kahteen kategoriaan: XML:n intrastruktuurin louhintaan ja XML:n interstrukstuurin louhintaan.

3.2.1.1 XML:n intrastruktuurin louhinta

XML:n intrastruktuurin louhinnan kohteena on XML:n sisäinen rakenne. Uutta tietoa voidaan löytää XML:n dokumenttien tyyppimäärittelystä (Nayak & al., 2002).

Tiedonlouhinnassa luokittelulla tarkoitetaan niitä ennustavia menetelmiä, joilla tietoalkiolle pyritään määrittämään jokin ennalta määrätyistä luokista (Han & Kamber, 2001). XML-dokumenttien sisäisten rakenteiden luokittelun tuloksia voidaan soveltaa uuden XML-dokumentin sijoittamiseen ennalta määriteltyyn XML-dokumenttien luokkaan. Dokumentin tyyppimäärittely voidaan tulkita XML-dokumenttien luokan kuvaukseksi. Luokittelussa tyyppimäärittelyjen kokoelma otetaan opetusjoukoksi, ja uusi XML-dokumentti luokitellaan sen mukaisesti. Yksinkertaisinta on suorittaa luokittelu XML-dokumentteihin, jotka ovat valideja ja oikein muodostettuja (well-formed) (Nayak & al.,

2002).

Klusteroinnilla tarkoitetaan tiedonlouhinnassa menetelmiä, jossa tietoalkiot ryhmitellään äärelliseen määrään klustereita niiden keskinäisen samanlaisuuden perusteella, ja samanlaisuus puolestaan määritellään yleensä etäisyysfunktiolla (Han & Kamber, 2001). XML-muotoisen tiedon tiedonlouhinnassa klusterointia voidaan käyttää etsittäessä yhtäläisyyksiä eri XML-dokumenttien väliltä. Luokittelussa käsitellään kokoelmaa tyyppimäärittelyjä, ja ryhmitellään ne itsesimilaarisuuden perusteella. Näitä yhtäläisyyksiä käytetään edelleen uusien tyyppimäärittelyjen generointiin (Nayak & al., 2002).

Assosiaatiosääntöjen louhintaa voidaan käyttää yhdessä esiintyvien elementtien suhteiden kuvaamisessa (Nayak & al., 2002). Muuttamalla XML:n puurakenne assosiaatiosääntöjen etsinnässä yleisesti käytettyyn transaktiomalliin voidaan XML-dokumentin rakenteesta löytää esimerkiksi seuraavanlainen sääntö: “Jos XML dokumentti sisältää <customer> -elementin, niin 80 % niistä sisältää myös <transaction> -elementin.”

Sekvenssihahmojen louhinta on assosiaatiosääntöjen louhinnan erityismuoto, ja sen soveltamista XML-muotoiseen tietoon käsitellään tarkemmin myöhemmin kohdassa 3.3.

3.2.1.2 XML:n interstruktuurin louhinta

XML:n interstruktuurin louhinnassa tutkitaan XML-dokumenttien välisiä rakenteita. (Nayak & al., 2002). Uutta tietoa löydetään esimerkiksi solmujen välisistä suhteista webissä. Luokittelua voidaan käyttää XML-dokumenttien nimiavaruuksien yhteydessä - kun jokin XML-dokumentin tyyppimäärittely on liitetty johonkin nimiavaruuteen, voidaan tätä tietoa käyttää hyväksi ko. nimiavaruudesta peräisin olevien XML-dokumenttien luokittelussa. Tyyppimäärittelyjen klusteroinnilla tarkoitetaan puolestaan samankaltaisten tyyppimäärittelyjen louhintaa. Klustereita käytetään määrittämään tyyppimäärittelyjen hierarkioita.

3.2.2 XML:n sisällön louhinta

XML-muotoisen tiedon sisällöllä tarkoitetaan tässä yhteydessä tekstimuodossa olevaa tietoa, joka esiintyy (mahdollisesti) XML:n aloitus- ja lopetuselementtien välissä. XML:n tietosisällön louhinta on tiedonlouhintaa em. tekstimuodossa olevan tiedon arvoista. XML:n puolirakenteisuus asettaa kuitenkin vakavia haasteita sisällön louhinnalle, koska perinteisten tiedonlouhintamenetelmien soveltaminen ei ole suoraviivaista (Zhang & al., 2004).

XML-sisällön louhinta voidaan edelleen jakaa kahteen kategoriaan: sisällön analyysiin (content analysis) ja rakenteisuuden selkeyttämiseen (structural clarification) (Nayak & al., 2002).

3.2.2.1 Sisällön analyysi

Sisällön analyysissa voidaan XML tietosisältöön suorittaa samanlaista louhintaa kuin mihin tahansa muuhun tekstidokumenttiin. Luokittelu suoritetaan XML:n tietosisältöön, ja XML:n sisältö määritetään kuuluvaksi ennalta määriteltyyn luokkaan. Vertailujen määrän rajaamiseksi uuden XML-dokumentin rakennemäärittely voidaan luokitella olemassa olevien rakennemäärittelyjen perusteella (Nayak & al., 2002).

XML sisällön klusteroinnissa löydetään mahdolliset uudet luokittelut. Rakennemäärittelyjen hyväksikäyttö johtaa nopeampaan klusterointiin: samanlaisten rakennemäärittelyjen mukaisissa dokumenteissa on todennäköisimmin samanlaista tietosisältöä (Nayak & al., 2002). Esimerkiksi kaikissa rakennemäärittelyissä, jotka käsittelevät kulkuneuvoja, voidaan kuvitella olevan eri tietojoukko, joilla kuvataan autoja, veneitä, lentokoneita jne. Kuitenkin myös sellaiset rakennemäärittelyt, joiden rakenne näyttää erilaiselta, voi olla samanlainen sisältö. XML:n sisällön louhinnassa on siis samoja ongelmia kuin tekstimuotoisen tiedon louhinnassa ja analyysissä. Synonyymit ja monimerkityksellisyys voivat aiheuttaa ongelmia, mutta näissä tapauksissa XML:n elementtien nimiä ja sijaintia dokumentissa voidaan käyttää hyväksi tietosisällön tulkinnassa.

3.2.2.2. Rakenteen selkeyttäminen

XML-dokumentin sisältö voi tarjota mahdollisuuden rakennemäärittelyjen klusteroinnille, koska erilaiset rakennemäärittelyt omaavilla XML-dokumentilla voi olla samanlainen sisältö. Samoin rakennemäärittelyt voivat tarjota apua sisällön klusteroinnille. Kaksi XML-dokumenttia, joilla on erilainen sisältö, voidaan asettaa samaan klusteriin niiden rakennemääritysten perusteella. Sisältö voi myös osoittautua tärkeäksi erilaisilta näyttävien rakennemääritysten klusteroinille. Rakennemäärittelyt voivat kuvata samaa asiaa eri tavoin, tai päinvastoin. Esimerkiksi seuraavan esimerkin ensimmäinen tulkinta tarkoittaa arvoasemaa, ja jälkimmäinen levyasemaa.

```
<asema>Johtaja</asema>
<asema>C:</asema>
```

Sisällön samanlaisuus ei tee eroa elementtien nimen semanttiselle merkitykselle. Tiedonlouhinta XML-muotoisen tiedon sisällöstä voi selkeyttää XML-elementin merkitystä (Nayak & al., 2002).

3.3 Tiedonlouhintamenetelmien käyttö XML:n louhintaan

Tieteellisessä kirjallisuudessa esitetyt lähestymistavat XML-muotoisen tiedon tiedonlouhintaan voidaan jakaa kahteen toisestaan poikkeavaan kategoriaan (Zhang & al., 2004; Scardina & al., 2004):

1) XML-muotoisen datan esikäsittely, jolloin XML-data muunnetaan ensin relaatiotietokannaksi, ja käytetään perinteisiä kyselykieliä tiedonlouhintaan.
2) XML-käsittelykielen kehitys, jossa pyritään käyttämään ja kehittämään XML-käsittelykieliä suoraan natiivin XML-muotoisen tiedon tiedonlouhintaan.

3.3.1 Perinteisten menetelmien soveltamisen ongelmat

Perinteisten tiedonlouhintamenetelmien soveltaminen puolirakenteiseen XML-muodossa kuvattuun tietoon ei ole suoraviivaista (Edmonds, 2003). Ensinnäkin menetelmät on alun perin suunniteltu ensisijaisesti relaatiomallia silmällä pitäen. Vaikka relaatiomallin mukainen tieto voidaan esittää XML-muodossa, ei kaiken XML-muodossa olevan tiedon esittäminen relaatiomallissa ole mahdollista. Relaatiomallissa tieto esitetään relaatioina, jossa jokainen relaatio muodostuu relaation skeemasta ja relaation instansseista. Tieto relaatiotietokannassa on yksiulotteista, ja elementtien välillä ei ole sisältyvyyssuhdetta. Relaatiomalliin pohjautuvassa tiedonlouhinnassa käydään suoraviivaisesti läpi tietokantaan tallennettu tieto ilman, että tiedon sisäinen rakenne otetaan huomioon.

XML-muodossa oleva tieto on rakenteeltaan kompleksisempaa, koska se kuvataan tabulaarisen rakenteen sijasta puumallina. Puun solmut edustavat elementtejä tai attribuutteja, ja särmät, jotka sisältävät solmuja, edustavat vanhempi-lapsi suhdetta elementtien välillä. Jokaisella särmällä on elementin nimi. Sisältyvyyssuhde (containment) on elementtien pääasiallinen suhde. Toisin kuin relaatiomallissa, yksittäisillä tietoalkioilla ei aina ole täsmälleen samaa rakennetta. XML-muotoisella tiedolla ei välttämättä ole etukäteen määriteltyä tarkkaa skeemaa, ja alielementtien puuttuminen tai toistuminen on sallittua (Zhang & al., 2004).

Toiseksi XML-muotoisella tiedolla ei ole symmetrisyyden vaatimusta. Alipuiden ei tarvitse sisältää samaa määrää lapsielementtejä. Jos XML muunnetaan oppivien vektoreiden (learning vectors) malliin, tulee mallista todennäköisesti erittäin harva, jossa suuret osat eivät sisällä mitään dataa. Tämän muotoisen datan käsittely perinteisillä tiedonlouhintamenetelmillä on haasteellista (Edmonds, 2003; Feng & Dillon, 2004).

Kolmanneksi XML-data on hierarkkista, joten elementin sijainti puussa voi olla merkityksellistä tiedon sisällön kannalta. Esimerkiksi analysoitaessa geneettisiä ominaisuuksia sukupuussa, on sekä yksilön ominaisuuksilla että vanhempien ja isovanhempien ominaisuuksilla merkitystä (Feng & Dillon, 2004).

Neljänneksi tiedon esittäminen XML muodossa vaatii enemmän muistitilaa (Feng & Dillon, 2004).

3.3.2 Tiedonlouhinta XML:stä relaatiomalliin muuntamalla

Varsinainen XML-muotoisen tiedon muuntaminen relaationmalliin voidaan tehdä kahdella tavalla. Ensimmäinen vaihtoehto on jäsentää XML osiin erillisellä sovelluksella ja tallettaa tiedot eri tauluihin (shredding, decomposition), ja XML-muotoisen tiedon muodostaminen relaatiomallin tallennetusta mallista voidaan tehdä vastaavasti koostaen (composition) (Florescu & Kossman, 1999). XML:n skeemaspesifikaatio elementteineen ja attribuutteineen tallennetaan yhdessä XML:n sisällön kanssa tietokantaan arvoina. XML:n tallentaminen XML-skeemasta riippumattomiin relaatiotietokantakaavoihin mahdollistaa uudelleenkäytettävyyden, jolloin minkä tahansa XML-dokumentti voidaan tallentaa samaan relaatiomalliin (Kappel & al., 2004).

Haittapuolena menetelmässä on se, että mikä tahansa tallennettuun tietoon kohdistuvan kyselyn on ensin rekonstruoitava XML-skeema, mahdollisesti useiden liitosten avulla, ennen kuin “todelliseen” tietosisältöön päästään käsiksi. Tästä syystä kyselyiden rakentaminen on hankalaa ja kyselyiden suorituskyky heikko (Florescu & Kossman, 1999). Lisäksi alkuperäisten XML-dokumenttien informaatiosta saatetaan tallentamisen yhteydessä menettää osa (Kappel & al., 2004).

Toinen vaihtoehto XML-muotoisen tiedon muuntamiselle relaatiomalliin on sijoittaa XML-dokumentit vastaavaan relaatiomalliin (Zhang & al., 2004). Näihin XML-skeemasta johdettuihin relaatiotietokantoihin tehdyt kyselyt ovat suorituskykyisiä, mutta niihin voidaan tallentaa vain tietyn ennalta määrätyn XML-skeeman mukaisia dokumentteja. Jos XML:n muodon määräävä skeemaa ei ole olemassa, tai XML skeema on jatkuvasti muuttuva, on relaatiomallin muodostaminen vaikeaa. Myös tässä tavassa alkuperäisen XML-dokumentin informaatio saatetaan menettää muunnoksen yhteydessä. Lisäksi relaatiomallin sopeuttaminen vaihtelevaan XML-skeemaan tekee siitä vaikeasti ylläpidettävän, ja samalla sen ymmärrettävyys kärsii (Kappel & al., 2004).

Kummassakin muunnostavassa on lisäksi haittana se, että tiedon rakenteen muuttaminen on manuaalinen, monimutkainen, ja aikaa ja resursseja kuluttava prosessi (Ding & Sundarraj, 2006).

3.3.3 XML-käsittelykielten käyttö sekvenssihahmojen louhintaan

Käyttämällä XML:n käsittelykieliä voidaan XML-muodossa olevaan tietoon suorittaa kyselyjä ja operaatiota ilman esi- tai jälkikäsittelyä. XML:n rakennetta ja sen sisältämää tietoa ei esimerkiksi tarvitse muuntaa toiselle käsittelykielelle sopivaan muotoon. XML-käsittelykielillä voidaan tehdä kyselyjä kaikkiin XML-muodossa tallennettuun tietoon, niin tietokannan XML-tietotyyppiä olevaan kenttään tallennettuun tietoon, kuin XML-dokumentteihinkin. Kaikille XML-teknologiaan pohjautuville käsittelykielille on yhteistä se, että niiden avulla voidaan suorittaa kyselyjä ja operaatioita sekä tiedon sisältöön, että sen rakenteeseen. Käsittelykielellä suoritetussa kyselyssä kyselyn tulos riippuu sekä elementtien sijainnista XML:ssä, että sen arvosta (Zhang & al., 2004).

Kaikki kyselykielet, riippumatta tiedon esitysmuodosta, on suunniteltu vastaamaan inhimillistä tiedontarvetta. Käyttäjä kuvaa kyselyssä ainoastaan haettavan tiedon spesifikaation, mutta ei sitä, kuinka tieto haetaan. Järjestelmän on vastauksen muodostaakseen järjestettävä haku vaiheisiin ja sen jälkeen suoritettava eri vaiheet (Zhang & al., 2004).

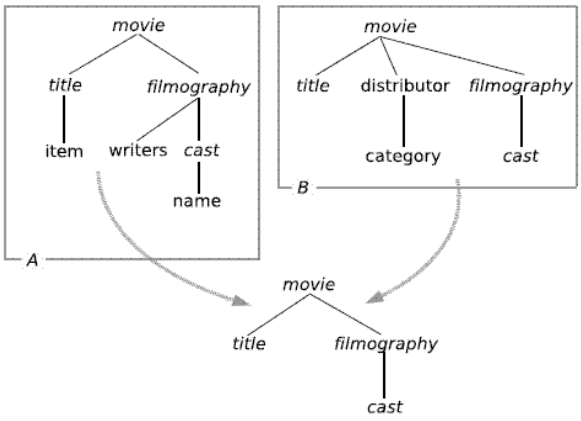
XML-käsittelykielten soveltuvuus vaativien algoritmien, kuten sekvenssihahmojen louhintaan käytettävien algoritmien, ohjelmointikieleksi on kuitenkin epävarmaa, sillä niiden ilmaisukyky ja tehokkuus on rajoittunut. Wanin ja Dobbien (2004) mukaan esimerkiksi XQuery-käsittelykielen tulisi sisältää tietojen päivitys-toiminnallisuus, ennen kuin assosiaatiosääntöjen etsintään kielellä toteutetusta Apriori-algoritmista saadaan tehokas. Joka tapauksessa XML-käsittelykielten käyttö tiedonlouhintaan on toistaiseksi melko vähäisesti tutkittua.

3.3.4 Toistuvien rakenteiden louhinta XML:stä

Suurin osa XML-muotoisen tiedon louhintaa käsittelevä tieteellinen kirjallisuus koskee XML:n rakenteen louhintaa. XML-muotoinen tieto on usein muodoltaan epätäydellistä - se voi sisältää esimerkiksi puuttuvia rakenteita. Koska XML-dokumentti esitetään puurakenteena, on puun rakenteiden louhinnan algoritmeja sovellettu XML-dokumenttien rakenteiden louhintaan (Zaki, 2005). Toistuvien rakenteiden louhinta käyttää hyväkseen sekvenssihahmojen louhinnassa käytettyjä menetelmiä. Muita XML rakenteen louhintaan liittyviä tutkimusalueita ovat XML-skeeman louhinta, dokumenttityyppimääritysten louhinta ja klusterointi, sekä XML-dokumenttien klusterointi.

Seuraavaksi esitetään toistuvien rakenteiden louhinnan periaate XML-dokumenteista Garbonin & al. (2004) mukaan. Toistuvalla rakenteella tarkoitetaan tässä yhteydessä rakennetta, joka esiintyy vähintään minimituen edellyttämässä määrässä tutkittavista XML-dokumentissa.

Kuvassa 10 on kuvattu toistuvan alirakenteen louhinta XML-dokumenteista.



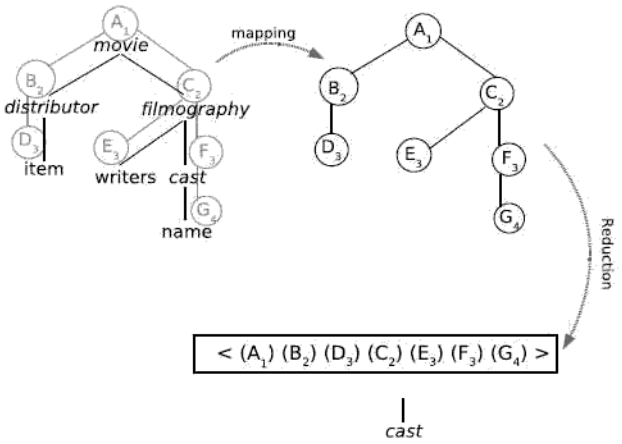
Kuva 10. Toistuvien alirakenteiden etsintä XML-dokumenteista (Garboni & al., 2004).

Kuvassa 10 kuviot A ja B esittävät puumuodossa kuvattuja XML dokumentteja. Esimerkiksi kuvion A kuvaama XML-dokumentti voisi olla seuraava:

```
<movie>
      <title>
            <item/>
      </title>
      <filmography>
            <writers/>
            <cast>
                  <name/>
            </cast>
      </filmography>
</movie>
```

Toistuvien alirakenteiden etsinnässä pyritään löytämään ne alipuut, joilla on yhteisiä rakenteellisia osia. Dokumenteissa A ja B minimituen ollessa 100 %, yhteinen toistuva alirakenne on kuvan ala-osassa. XML-dokumentti, jossa on juurielementti `<movie>`, ja jolla on kaksi lapsielementtiä `<title>` ja `<filmography>`, ja jota seuraa `<cast>` elementti, löytyy sekä dokumentista A että B. Kuvan 10 alaosan kuvaama puu on myös laajin toistuva alirakenne.

Toistuvien alirakenteiden etsintä XML-muotoisesta tiedosta rinnastuu sekvenssihahmojen louhintaan. Kumpaakin voidaan käyttää samoja ratkaisuperiaatteita. Etsittäessä toistuvia alirakenteita, sijoitetaan ensin XML-dokumentin solmut yksilöllisiin avaimiin, jonka jälkeen jokaiseen avaimeen liitetään tieto sen tasosta puussa. Viimeiseksi puun syvyyssuuntaisella (depth-first) haulla muodostetaan sekvenssi. Tätä prosessia kuvataan kuvassa 11.



Kuva 11. Toistuvien alirakenteiden etsintä XML dokumenteista (Garboni & al., 2004).

Alkuperäinen XML-dokumentin rakenne (kuvassa vasemmalla) sijoitetaan ensin merkittyyn (labeled) puuhun. Esimerkiksi `<filmography>`-elementti saa arvon “C2” joka vastaa `<filmography>`-elementin yksilöllistä avainta (C) puun tasolla kaksi (2). Seuraavaksi muodostetaan vastaava sekvenssi syvyyssuuntaisen läpikulun jälkeen. Kun kaikki sekvenssien joukot on muodostettu, voidaan normaalia sekvenssihahmojen louhinta-algoritmia käyttää löytämään puussa esiintyvät sekvenssihahmot. Kun löydetyt hahmot sijoitetaan takaisin puurakenteeseen, saadaan tulos.

4 APRIORIALL-ALGORITMIN XSLT 2.0 -TOTEUTUS

Tässä luvussa esitellään sekvenssihahmojen louhinnan toteutus XML:n tietosisällöstä ja rakenteesta XSLT 2.0 -käsittelykielellä. Ensimmäisessä kohdassa esitetään toteutusten lähtökohdat. Toisessa kohdassa esitellään aikaisemmin luvussa 2 kuvattu AprioriAll-algoritmin toteutus. Kolmannessa kohdassa esitellään rakenteen louhinnan toteutus. Viimeisessä kohdassa esitetään toteutuksen suorituskykytestauksen tulokset.

4.1 XSLT 2.0 -toteutuksen lähtökohdat

Seuraavissa kohdissa esiteltävässä AprioriAll-algoritmin toteutuksissa on sovellettu edellisissä luvuissa kuvattuja menetelmiä XSLT 2.0 -käsittelykielellä (W3C, 2008e). Pyrkimyksenä on ollut tuottaa ratkaisut, jotka ovat puhtaita XSLT 2.0 -toteutuksia, eivätkä vaadi millään muulla ohjelmointikielellä toteutettua eri vaiheiden tulosten välivarastointia tai käsittelyä. Toteutuksessa on myös pyritty käyttämään XSLT 2.0:n ominaisuuksia siten, että toteutus voidaan suorittaa eri XSLT-prosessoreissa. Tavoitteena on ollut selvittää, kuinka XSLT 2.0 soveltuu AprioriAll-algoritmin toteutukseen. Wan ja Dobbien (2004) ovat toteuttaneet assosiaatiosääntöjen etsintään tarkoitetun Apriori-algoritmi XQuery-käsittelykielellä, mutta tutkielman tekijän tiedossa ei ole AprioriAll-algoritmin, tai muiden sekvenssihahmojen louhintaan käytettyjen algoritmien toteutuksia XML-pohjaisilla käsittelykielillä.

Toteutuksessa on käytetty kaikissa transformaatiossa ja aliohjelmissa template-elementtiä niiden toteutukseen. XSLT 2.0 -käsittelykielessä on myös function-elementti, jota usein käytetään aliohjelmien toteutukseen. Kayn (2004b) mukaan template- ja function-elementeillä ei ole kuitenkaan eroa siinä, kuinka ne tuottavat tuloksen. Niiden suorituskyky on myös samanlainen. Toinen XSLT 2.0 -kielestä käyttämätön ominaisuus toteutuksessa on sequence-elementti, jonka sijasta on käytetty copy-of-elementtiä. Sequence-elementti poikkeaa muista XSLT 2.0 -elementeistä siten, että se palauttaa viitteen olemassa olevaan elementtiin, toisin kuin copy-of-elementti, joka tuottaa elementistä kopion (Kay, 2004b). Sequence-elementtiä käytetään usein function-elementin tuottaman tulok-

sen evaluointiin (Kay, 2004b). Sequence-elementin käytöllä on mahdollista vähentää muistin kulutusta. AprioriAll-algoritmissa tuotetaan kuitenkin jatkuvasti uusia elementtejä, jolloin copy-of elementin käyttö sequence-elementin sijasta on mielestäni perusteltua.

Seuraavissa kohdissa on kuvattu algoritmin toteuttavat templatet ja lähdekoodi keskeisiltä osiltaan. Jotkin toteutuksen osat on kuvattu XSLT-pseudotemplaten avulla. Tällä on tarkoitettu sitä, että kuvattu template ei ole aito suoritettava koodi, vaan kuvaa XSLT 2.0 -käsittelykieltä läheisesti muistuttavilla elementeillä ratkaisun keskeiset ideat. Kun XSLT-pseudotemplatea on käytetty, on siitä myös erikseen mainittu. Täydellinen Altova XSLT-prosessorilla (Altova, 2008) ajettava lähdekoodi on liitteessä 1.

4.2 Sekvenssihahmojen louhinta XML:n sisällöstä

Tässä kohdassa esitellään AprioriAll-algoritmin toteutus XSLT 2.0 -käsittelykielellä.

4.2.1 Lajitteluvaihe

Listauksessa 1 on esitetty alkuperäinen tietokanta XML-muodossa. Juurielementin `<data>` sisään on sijoitettu kaikki asiakkaiden tekemät ostotapahtumat tapahtumajärjestyksessä. Asiakaselementti `<customer>` sisältää asiakaan ostotapahtumaelementin `<transaction>`, ja jokainen ostotapahtuma sisältää ko. ostotapahtumassa ostettujen tuotteiden tunnukset `<item>`-elementissä. Jokaisella asiakaselementillä on attribuutti `id`, joka kertoo asiakkaan asiakastunnuksen. Jokaisella ostotapahtumaelementillä on puolestaan attribuutti `time`, joka kuvaa oston tapahtuma-aikaa päivämäärän tarkkuudella. Alla olevassa esimerkissä päivämäärä on ilmaistu muodossa *vvvvkkpp* (*vvvv* = vuosi, *kk* = kuukausi ja *pp* = päivä).

```
<data>
<customer id="2">
    <transaction time="19930610">
        <item>10</item>
        <item>20</item>
    </transaction>
</customer>
<customer id="5">
    <transaction time="19930612">
```

```
        <item>90</item>
    </transaction>
</customer>
<customer id="2">
    <transaction time="19930615">
        <item>30</item>
    </transaction>
</customer>
<customer id="2">
    <transaction time="19930620">
        <item>40</item>
        <item>60</item>
        <item>70</item>
    </transaction>
</customer>
<customer id="4">
    <transaction time="19930625">
        <item>30</item>
    </transaction>
</customer>
<customer id="3">
    <transaction time="19930625">
        <item>30</item>
        <item>50</item>
        <item>70</item>
    </transaction>
</customer>
<customer id="1">
    <transaction time="19930625">
        <item>30</item>
    </transaction>
</customer>
<customer id="1">
    <transaction time="19930630">
        <item>90</item>
    </transaction>
</customer>
<customer id="4">
    <transaction time="19930630">
        <item>40</item>
        <item>70</item>
    </transaction>
</customer>
<customer id="4">
    <transaction time="19930725">
        <item>90</item>
    </transaction>
</customer>
</data>
```

Listaus 1. Alkuperäinen tietokanta.

Ensimmäinen tehtävä on lajitella tiedot asiakastunnuksen ja tapahtuma-ajankohdan mu-

kaan. XSLT 2.0:ssa lajittelu voidaan tehdä ryhmittelemällä tiedot asiakastunnuksen mukaan `<xsl:for-each-group>` -elementin avulla seuraavasti:

```
<xsl:for-each-group select="customer" group-by="@id">
```

Jokaisen `id`-attribuutin perusteella luodun asiakasryhmän sisällä valitaan asiakkaan transaktiot kaikista tapahtumista, ja lajitettaan ne tapahtuma-ajankohdan mukaan `<xsl:for-each>` ja `<xsl:sort>` -elementtien avulla seuraavasti:

```
<xsl:for-each
            select="../customer/transaction
                  [../@id = current-grouping-key()]">
      <xsl:sort select="@time" data-type="number"/>
```

Lajittelun tuloksena syntyy uusi XML-dokumentti, joka on esitetty listauksessa 2.

```
<data>
      <customer id="1">
            <transaction time="19930625">
                  <item>30</item>
            </transaction>
            <transaction time="19930630">
                  <item>90</item>
            </transaction>
      </customer>
      <customer id="2">
            <transaction time="19930610">
                  <item>10</item>
                  <item>20</item>
            </transaction>
            <transaction time="19930615">
                  <item>30</item>
            </transaction>
            <transaction time="19930620">
                  <item>40</item>
                  <item>60</item>
                  <item>70</item>
            </transaction>
      </customer>
      <customer id="3">
            <transaction time="19930625">
                  <item>30</item>
                  <item>50</item>
                  <item>70</item>
            </transaction>
      </customer>
      <customer id="4">
```

```
            <transaction time="19930625">
                  <item>30</item>
            </transaction>
            <transaction time="19930630">
                  <item>40</item>
                  <item>70</item>
            </transaction>
            <transaction time="19930725">
                  <item>90</item>
            </transaction>
      </customer>
      <customer id="5">
            <transaction time="19930612">
                  <item>90</item>
            </transaction>
      </customer>
</data>
```

Listaus 2. Lajittelun tulos.

4.2.2 Suurten tietoalkiojoukkojen generointivaihe

Suurten tietoalkiojoukkojen generoinnissa etsitään ensin kaikki yhden alkion kokoiset tietoalkiojoukot, joiden tuki on vähintään minimituen suuruinen. XSLT 2.0 -toteutuksessa voidaan jälleen käyttää hyväksi ryhmittelyelementtiä, jonka jälkeen voidaan jokaisen ryhmittelyavaimena olevan elementin `<item>` asiakaskohtainen tuki laskea muuttujan arvoksi seuraavasti:

```
<xsl:variable name="support"
                        select="count(//data/customer[transaction/item
                                    = current-grouping-key()])"/>
```

Muuttuja `support` saa arvokseen `count()` -funktion palauttaman arvon, joka puolestaan saa parametrina elementtijoukon, johon valitaan kaikki ne `<customer>`-elementit, joiden transaktioista löytyy ryhmittelyavaimen mukainen `<item>`-elementti. Seuraavaksi välitetään kandidaatteja generoivalle templatelle edellisen vaiheen tulos, jota kutustaan rekursiivisesti niin kauan, kun minimituen täyttäviä alkiojoukkoja löytyy.

Kandidaatteja generoivassa templatessa muodostetaan `<xsl:for-each>` -elementin avulla kahdessa sisäkkäisessä silmukassa uudet kandidaatit liittämällä kaksi alkiojoukkoa yhteen. Ulommassa silmukassa käydään läpi kaikki *n*-kokoiset tietoalkiojoukot. Sisem-

mässä silmukassa otetaan käsittelyyn ainoastaan ne *n*-kokoiset alkiojoukot, (1) joiden sijainti (`position()`-funktion palauttama indeksi) on myöhempi, kuin ulommassa silmukassa käsittelyvuorossa oleva alkiojoukon sijainti, (2) joiden viimeistä edelliset alkiot ovat samoja kuin ulommassa silmukassa käsittelyvuorossa olevassa tietoalkiojoukossa, ja (3), joiden viimeinen alkio on erilainen kuin silmukassa käsittelyvuorossa olevassa tietoalkiojoukossa.

```
<xsl:for-each select="$litemsets/litemset">
      <xsl:variable name="tag" select="item" />
      <xsl:variable name="tail" select="item[position() = last()]" />
      <xsl:variable name="head" select="item[position() != last()]" />
      <xsl:variable name="position" select="position()" />

            <xsl:for-each select="
                  ../litemset[(position() &gt; $position) and
                  (count(item[$head=node()]) = count($head)) and
                  ($tail != item[position() = last()])] ">
```

Ulomman silmukan läpikäytävän joukon sijainti on asetetaan `position`-muuttujaan, läpikäytävät elementit `tag`-muuttujaan, viimeistä edelliset alkiot `head`-muuttujaan, ja viimeinen alkio `tail`-muuttujaan. Esimerkiksi elementtien

```
<item>10</item><item>20</item>
```

ja

```
<item>20</item><item>40</item>
```

liitoksen tuloksena syntyy uusi elementti

```
<item>10</item><item>20</item><item>20</item><item>40</item>
```

josta XSLT 2.0 -funktion `distinct-values` avulla poistetaan ylimääräiset alkiot, jolloin tulokseksi saadaan kandidaattielementti

```
<item>10</item><item>20</item><item>40</item>.
```

Syntynyt kandidaattielementti asetetaan `purged`-muuttujan arvoksi, ja otetaan mukaan tulosjoukkoon, jos sen tuki ylittää minimituen rajan. Tuen laskenta ja lisäys tulosjouk-

koon tehdään seuraavasti:

```
<xsl:if test="count
            ($data/data/customer[transaction
                  [count(item[node() = $purged/item])
            = count($purged/item)]]) &gt;= $minsupport">
      <litemset>
            <xsl:copy-of select="$purged" />
      </litemset>
</xsl:if>
```

XPath:ssa vertailu yhtäsuuruusmerkillä tuottaa arvon tosi, jos vertailtavien elementtijoukkojen yksikin elementti on samanlainen (W3C, 2008d). Siksi vertailu on tässä tapauksessa tehty niin, että erikseen on laskettu samanlaisten elementtien lukumäärä, ja verrattu sitä kandidaatin elementtien lukumäärään. Menetelmä on sovellus XSLT:ssä elementtisolmujen joukkojen leikkauksen tuottamiseen Kayn (2004a) käyttämästä menetelmästä. Edellä esitetty ehtolause voidaan ilmaista seuraavasti: ”Jos kaikkien niiden customer-elementtien lukumäärä, joiden transaktiossa esiintyvien ja kandidaattielementtien samojen alkioiden lukumäärä on yhtä suuri kuin kandidaattielementtien alkioiden lukumäärä, on suurempi kuin minimituki, niin ehto on tosi.” Suurten tietoalkiojoukkojen generointi tuottaa tulokseksi listauksessa 3 kuvatun XML-dokumentin.

```
<data>
      <litemset>
            <item>30</item>
      </litemset>
      <litemset>
            <item>40</item>
      </litemset>
      <litemset>
            <item>70</item>
      </litemset>
      <litemset>
            <item>90</item>
      </litemset>
      <litemset>
            <item>40</item>
            <item>70</item>
      </litemset>
</data>
```

Listaus 3. Suuret tietoalkiojoukot.

Toisin kuin Agrawalin & al. (1995) esittämässä AprioriAll-algoritmissa, ei suuria tietoalkiojoukkoja ole vielä tässä vaiheessa tarpeellista sijoittaa nousevaan kokonaislukuun, koska XSLT 2.0:ssa se voidaan tehdä yksinkertaisesti muunnosvaiheen yhteydessä.

4.2.3 Muunnosvaihe

XSLT 2.0 -toteutuksessa tietoalkiojoukkojen sijoittaminen kokonaislukuun voidaan tehdä muunnosvaiheen yhteydessä. Muunnosvaiheen toteuttava template suoritetaan lajitteluvaiheen tuloksena syntynyttä XML:ää vasten. Template (listaus 4) lukee myös suuret tietoalkiojoukot sisältävän tiedoston ”litemsets.xml”. Templatessa käydään `<xsl:for-each>`-elementin avulla sisäkkäisissä silmukoissa läpi `<customer>` -elementit ja `<transaction>`-elementit. Jokaisen `<transaction>`-elementin sisältämät `<item>`-elementit välitetään `mapping`-templatelle parametrissa `items`. Template käy läpi silmukassa suuret tietoalkiojoukot ja lajittelee ne viimeisen alkion mukaan[1]. Jos jokin suurista tietoalkiojoukoista löytyy parametrina saaduista `<item>`-elementeistä, korvataan se suurten tietoalkiojoukon sijainti-indeksin arvolla. Kun suuret tietoalkiojoukot käydään läpi silmukassa aina samassa järjestyksessä, tulee korvaavaksi sijoitusarvoksi myös aina sama kokonaisluku.

```
<xsl:variable name="data" select="document('litemsets.xml')" />
<xsl:template name="mapping">
<xsl:param name="items"/>

<xsl:for-each select="$data/data/litemset">
      <xsl:sort select="item[position() = last()]"/>
      <xsl:if test="count(item[$items = node()]) = count(item)">
            <item><xsl:value-of select="position()"/></item>
      </xsl:if>
</xsl:for-each>

</xsl:template>
```

Listaus 4. Sijoituksen tuottava template.

[1] Lajittelu ei ole välttämätöntä, mutta tässä tapauksessa sitä käytetään, että sijoitus saadaan vastaamaan Agrawalin ja Srikantin (1995) esittämää sijoitus-vaihtoehtoa.

Templaten elementissä <xsl:if> määritellyssä ehtolausekkeessa lasketaan ensimmäisessä XSLT:n funktiossa count silmukassa käsittelyssä olevan suurten tietoalkiojoukkojen <litemset>-elementin ja käsiteltävän <transaction>-elementin samojen <item>-elementtien lukumäärät. Toisessa count-funktiossa lasketaan käsiteltävän <transaction>-elementin samojen <item>-elementtien lukumäärä. Jos count-funktioden tulos on sama, saa <item>-elementti sijoitusarvon. Tämä ehtolauseke myös karsii pois ne elementit, jotka eivät sisällä suuria tietoalkiojoukkoja. Muunnosvaiheen tuloksena saadaan listauksessa 5 kuvattu XML-dokumentti.

```
<data>
	<customer>
		<transaction>
			<item>1</item>
		</transaction>
		<transaction>
			<item>5</item>
		</transaction>
	</customer>
	<customer>
		<transaction>
			<item>1</item>
		</transaction>
		<transaction>
			<item>2</item>
			<item>3</item>
			<item>4</item>
		</transaction>
	</customer>
	<customer>
		<transaction>
			<item>1</item>
			<item>3</item>
		</transaction>
	</customer>
	<customer>
		<transaction>
			<item>1</item>
		</transaction>
		<transaction>
			<item>2</item>
			<item>3</item>
			<item>4</item>
		</transaction>
		<transaction>
			<item>5</item>
		</transaction>
	</customer>
	<customer>
		<transaction>
```

```
                <item>5</item>
            </transaction>
        </customer>
</data>
```

Listaus 5. Muunnosvaiheen tulos

4.2.4 Sekvenssivaihe

Sekvenssivaiheen ensimmäisellä kierroksella käydään läpi suurten tapahtumajoukkojen generointivaiheen tuottamat yhden alkion kokoiset tietoalkiojoukot. Tämä voidaan XSLT 2.0 -toteutuksessa tehdä vastaavalla tavalla kuin suurten alkiojoukkojen generointivaiheessa, mutta sillä poikkeuksella, että alkion tuki lasketaan asiakkaiden mukaan:

```
<xsl:variable name="support"
                select="count(//data/customer
                [transaction/item = current-grouping-key()])"/>
```

Generoidut tietoalkiojoukot välitetään parametrina (*2+n*) -kokoisia tietoalkiojoukkoja generoivalle templatelle `generate_litemsets`. Listauksessa 6 esitetty XSLT-pseudotemplate toteuttaa kandidaattien generoinnin ja puhdistuksen lopullisesta tuloksesta.

```
<xsl:template name="generate_litemsets">

<xsl:for-each select="$litemsets/litemsets/litemset[p]">
      <xsl:for-each select="$litemsets/litemsets/litemset[q]">

            <xsl:variable name="union"/>
            <xsl:call-template name="prune" select="$union"/>

            <xsl:if test="alisekvenssejä ei löytynyt">
                  <xsl:call-template name="support"   select="$union"/>
            </xsl:if>
      </xsl:for-each>
</xsl:for-each>


<xsl:if test="count($litemsets/litemsets/litemset) &gt; 0">

      <xsl:call-template name="generate_litemsets"/>
            <xsl:with-param name="litemsets" select="$litemsets"/>
      </xsl:call-template>
```

```
        <litemsets>
              <xsl:copy-of select="$litemsets/litemsets/litemset"/>
        </litemsets>

</xsl:if>

</xsl:template>
```

Listaus 6. Suuret alkiojoukot generoiva pseudotemplate.

Ulommassa silmukassa käydään läpi suuret (*k-1*)-kokoiset tietoalkiojoukot. Sisemmässä silmukassa valitaan läpikäyntiin ne suuret tietoalkiojoukot, joiden ensimmäiset (*k-1*) tietoalkiota ovat samat kuin ulommassa silmukassa läpikäyntivuorossa olevassa tietoalkiojoukossa. Seuraavaksi ulommassa silmukassa läpikäyntivuorossa olevassa tietoalkiojoukosta ja sisemmässä silmukassa läpikäyntivuorossa olevan tietoalkiojoukon viimeisestä tietoalkiosta muodostetaan liitoksen avulla kandidaatti. Templatessa `prune` tarkistetaan, löytyykö kandidaatin alisekvensseistä sellaisia, jotka eivät kuulu suuriin tietoalkiojoukkoihin, jonka jälkeen templatessa `count_support` lasketaan tuki. Jos tuki ylittää minimituen rajan, kandidaatti lisätään tulokseen. Kandidaatit generoivaa templatea kutsutaan rekursiivisesti niin kauan, kunnes tietoalkiojoukkoja ei enää muodostu.

Alisekvenssien esiintymisen tarkastava `prune`-template on esitetty listauksessa 7. Template saa parametrinaan kandidaattisekvenssin, sekä kaikki suuret tietoalkiojoukot. Silmukassa käydään läpi jokainen kandidaattisekvenssin tietoalkio, ja muodostetaan uusi alisekvenssi ilman ko. tietoalkiota. Näin voidaan muodostaa kaikki (*k-1*)-kokoiset alisekvenssit. Muodostettua alisekvenssiä verrataan ehtolauseessa kaikkiin suuriin tietoalkiojoukkoihin, ja jos löydetään sellainen sekvenssi, jonka tietoalkioiden lukumäärä on sama kuin kandidaatin tietoalkioiden lukumäärä-1, ja jotka ovat täsmälleen samanlaisia, lisätään muodostettu alisekvenssi templaten tuottamaan tulokseen. Sekvenssien samanlaisuuden vertailu tehdään XPath 2.0 -funktion `deep-equal` avulla (W3C, 2008d). Jos templaten tuloksena saadaan alisekvenssejä, voidaan kutsuvassa kandidaatteja generoivassa templatessa jättää kandidaatti lisäämättä suuriin tietoalkiojoukkoihin.

```
<xsl:template name="prune">

<xsl:param name="candidate"/>
<xsl:param name="litemsets"/>

<xsl:for-each select="$candidate/union/item">
      <xsl:variable name="subseq"
                  select="remove($candidate/union/item, position())"/>
      <xsl:if test="count($litemsets/litemsets/litemset
                  [(count(item) = (count($candidate/union/item)-1)) and
                  deep-equal(item,$subseq)]
                  ) = 0">
            <subseq><xsl:copy-of select="$subseq"/></subseq>
</xsl:for-each>

</xsl:template>
```

Listaus 7. Alisekvenssien esiintymisen tarkastava template.

Tuen laskenta kandidaatille tehdään listauksessa 8 kuvatun XSLT-pseudotemplaten avulla. Ideana on verrata kandidaatin jokaista tietoalkiota järjestyksessä asiakkaan transaktioiden tietoalkioihin. Jos sama tietoalkio löytyy, kasvatetaan laskurin arvoa, ja otetaan seuraavat tietoalkiot sekä kandidaatista, että asiakkaan transaktioista käsittelyyn. Jos samaa tietoalkiota ei löydy, otetaan seuraava asiakkaan tietoalkio käsittelyyn. Kun kaikki kandidaatin tietoalkiot on käsitelty, otetaan ensimmäinen tietoalkio uudelleen käsittelyyn ja jatketaan seuraavasta asiakkaan tietoalkiosta. Tätä toistetaan niin kauan, kunnes kaikki asiakkaan tietoalkiot on käsitelty. Jos tässä vaiheessa laskurin arvo on sama, kuin kandidaatin alkioiden lukumäärä, on sekvenssi löydetty asiakkaan transaktioiden tietoalkioista, ja kandidaatin tuen laskuria kasvatetaan. Etsintää toistetaan kunnes kaikki asiakkaat on käsitelty.

```
<xsl:template name="support">

<!-- käsittelyssä olevan asiakkaan laskuri k-->
<xsl:param name="k"/>
<!-- kandidaatti -->
<xsl:param name="kandidaatti"/>
<!-- käsittelyssä olevan kandidaatin alkion positiolaskuri i -->
<xsl:param name="i"/>
<!-- Käsiteltävän asiakkaan k transaktion alkion positiolaskuri j -->
<xsl:param name="j"/>
<!-- samojen alkioiden laskuri c -->
<xsl:param name="c"/>
<!-- puoltolaskuri s -->
```

```
<xsl:param name="s"/>


<xsl:when test="k < asiakkaiden lkm">
      <xsl:when test="j < asiakkaan tietoalkiojoukon alkioiden lkm">
            <xsl:when test="i <= kandidaatin alkioiden lkm">
                  <xsl:when test="asiakkaan tietoalkiojoukon alkio j =
                              kandidaatin alkio i">

                        <xsl:call-template name="position">
                        <xsl:with-param name="i" select="i + 1"/>
                        <xsl:with-param name="j" select="j + 1"/>
                        <xsl:with-param name="c" select="c + 1"/>
                        </xsl:call-template>
                  </xsl:when>
                  <xsl:otherwise>

                        <xsl:call-template name="support">
                        <xsl:with-param name="i" select="i"/>
                        <xsl:with-param name="j" select="j + 1"/>
                        </xsl:call-template>
                  </xsl:otherwise>
            </xsl:when>
            <xsl:otherwise>
                  <xsl:call-template name="position">
                  <xsl:with-param name="i" select="1"/>
                  <xsl:with-param name="j" select="$j + 1"/>
            </xsl:call-template>
      </xsl:when>

      <xsl:otherwise>
            <xsl:when test="c = kandidaatin alkioiden lkm">

                  <xsl:call-template name="support">
                        <xsl:with-param name="k" select="k + 1"/>
                        <xsl:with-param name="i" select="1"/>
                        <xsl:with-param name="j" select="1"/>
                        <xsl:with-param name="c" select="0"/>
                        <xsl:with-param name="s" select="s + 1"/>
                  </xsl:call-template>

            </xsl:when>
            <xsl:otherwise>

                  <xsl:call-template name="support">
                        <xsl:with-param name="k" select="k + 1"/>

                        <xsl:with-param name="i" select="1"/>
                        <xsl:with-param name="j" select="1"/>
                        <xsl:with-param name="counter" select="0"/>
                        <xsl:with-param name="support" select="s"/>
                  </xsl:call-template>
            </xsl:otherwise>

      </xsl:otherwise>
<xsl:otherwise>
```

```
      <xsl:if test="s > minimituki"/>
            <xsl:copy-of select="kandidaatti"/>
            <support><xsl:value-of select="s"/></support>
      </xsl:if>
</xsl:when>

</xsl:template>
```

Listaus 8. Tuen laskeva pseudotemplate.

Suurten tietojoukkojen generointivaiheessa syntyvä XML-dokumentti on kuvattu listauksessa 9.

```
<data>
   <litemsets>
      <litemset>
         <item>1</item>
         <item>2</item>
         <item>3</item>
         <item>4</item>
      </litemset>
   </litemsets>
   <litemsets>
      <litemset>
         <item>1</item>
         <item>2</item>
         <item>3</item>
      </litemset>
      <litemset>
         <item>1</item>
         <item>2</item>
         <item>4</item>
      </litemset>
      <litemset>
         <item>1</item>
         <item>3</item>
         <item>4</item>
      </litemset>
      <litemset>
         <item>2</item>
         <item>3</item>
         <item>4</item>
      </litemset>
   </litemsets>
   <litemsets>
      <litemset>
         <item>1</item>
         <item>2</item>
      </litemset>
      <litemset>
         <item>1</item>
```

```
        <item>3</item>
      </litemset>
      <litemset>
        <item>1</item>
        <item>4</item>
      </litemset>
      <litemset>
        <item>1</item>
        <item>5</item>
      </litemset>
      <litemset>
        <item>2</item>
        <item>3</item>
      </litemset>
      <litemset>
        <item>2</item>
        <item>4</item>
      </litemset>
      <litemset>
        <item>3</item>
        <item>4</item>
      </litemset>
  </litemsets>
</data>
```

Listaus 9. Suuret tietoalkiojoukot.

4.2.5 Enimmäissekvenssien vaihe

Enimmäissekvenssin etsinnän suorittava template saa syötteenään edellisessä vaiheessa syntyneen kaikki suuret tietoalkiojoukot sisältävän XML-dokumentin. Templatessa käydään läpi yksi kerrallaan tietoalkiojoukot, muodostetaan kustakin alisekvenssit, ja poistetaan ne läpikäytävistä tietoalkioista.

Oletetaan, että tietoalkiojoukon tietoalkioiden lukumäärä on *n*. Alisekvenssien generoinnin toteutuksen ideana on käydä kukin tietoalkiojoukko läpi 2^n - 2 kertaa. Alisekvenssien generoinnista jätetään pois alkuperäistä tietoalkiojoukkoa vastaava joukko ja tyhjä joukko. Kullakin läpikäyntikierroksella muodostetaan läpikäyntikierrosta ilmaisevasta luvusta binäärilukuesitys. Muodostettavaan alisekvenssiin valitaan ne tietoalkiojoukon tietoalkiot, joiden positioissa binäärilukuesityksessä on luku 1.

Esimerkiksi käsiteltäessä viiden tietoalkion kokoista tietoalkiojoukkoa

```
<item>1</item><item>5</item><item>2</item><item>3</item><item>4</item>
```

on läpikäyntikierroksia 2^5 - 2 = 30. Läpikäyntikierroksella 13 muodostettu binäärilukuesitys on 01101. Luku 1 on vasemmalta luettuna binäärilukuesityksessä positioissa 2, 3 ja 5. Alisekvenssiksi valitaan tietoalkiojoukon tietoalkiot vastaavista positioista

```
<item>5</item><item>2</item><item>4</item>
```

Niitä alisekvenssejä, joissa on vain yksi alkio, ei oteta mukaan tulokseen. Koska XSLT 2.0:sta puuttuu matemaattinen funktio (W3C, 2008e), jolla korotus voidaan tehdä, on se toteutettava rekursiivisen templaten (tai funktion) avulla. Sen jälkeen muodostetaan alisekvenssit niin ikään kahden rekursiivisen templaten avulla. Listauksessa 10 kuvattu template `counter` kutsuu ensin templatea `subseqs`, ja sen jälkeen itseään kasvattaen kierroslaskuria joka kierroksella, kunnes kierroslaskurin arvo on yhtä pienempi, kuin edellä suoritetun laskutoimituksen antama tulos.

```
<xsl:template name="counter">
<xsl:param name="i"/><!-- kierroslaskuri i  -->
<xsl:param name="p"/><!-- 2 ^ tietoalkiojoukkojen alkioiden lkm -->
<xsl:param name="k"/><!-- tietoalkiojoukkojen alkioiden lkm -->

<xsl:if test="$i &lt; ($p - 1)">
      <subseq>
      <xsl:call-template name="subseqs">
            <xsl:with-param name="i" select="$i"/>
            <xsl:with-param name="j" select="1"/>
            <xsl:with-param name="k" select="$k"/>
      </xsl:call-template>
      </subseq>
      <xsl:call-template name="counter">
            <xsl:with-param name="i" select="$i + 1"/>
            <xsl:with-param name="p" select="$p"/>
            <xsl:with-param name="k" select="$k"/>
      </xsl:call-template>
</xsl:if>
</xsl:template>
```

Listaus 10. Alisekvenssien generoinnin kutsu.

Listauksessa 11 kuvattu template `subsets` muodostaa kullakin käsittelykierroksella alisekvenssit

```
<xsl:template name="subseqs">
<xsl:param name="i"/>
<xsl:param name="j"/>
<xsl:param name="k"/>

<xsl:if test="$j &lt;= $k">
      <xsl:choose>
      <xsl:when test="not($i &lt; 1)">

            <xsl:call-template name="subseqs">
                  <xsl:with-param name="i" select="$i div 2"/>
                  <xsl:with-param name="j" select="$j + 1"/>
                  <xsl:with-param name="k" select="$k"/>
            </xsl:call-template>

            <xsl:if test="($i mod 2) &gt;= 1">
                  <item>
                  <xsl:value-of
                        select="item[position()=(($k + 1) - $j)]"/>
                  </item>
            </xsl:if>
      </xsl:when>
      <xsl:otherwise>

            <xsl:call-template name="subseqs">
                  <xsl:with-param name="i" select="0"/>
                  <xsl:with-param name="j" select="$j + 1"/>
                  <xsl:with-param name="k" select="$k"/>
            </xsl:call-template>
      </xsl:otherwise>
      </xsl:choose>
</xsl:if>
</xsl:template>
```

Listaus 11. Alisekvenssien generointi.

Muodostetut alisekvenssit tallennetaan muuttujan `removable_litemsets` arvoksi. Seuraavaksi templatessa käydään läpi listauksessa 12 esitetyllä tavalla kaikki alijoukkojen muodostamisessa käytettyä tietoalkiojoukkoa pienemmät ja yhtä suuremmat tietoalkiojoukot, ja otetaan tulokseen mukaan ne tietoalkiojoukot, joita ei löydy poistettavista alisekvensseistä.

```
<xsl:for-each select="litemset[(count(item) &lt; $maximal) and
                        (count(item) &gt; 1)]">

      <xsl:variable name="litemz" select="node()"/>

      <xsl:if test="count($removable_litemsets/litemset[
                        deep-equal(node(), $litemz)]) = 0">
            <litemset><xsl:copy-of select="$litemz"/></litemset>
      </xsl:if>
</xsl:for-each>
```

Listaus 12. Maksimaalisen sekvenssin lisäys tulokseen.

Etsintä tehdään XSLT 2.0 -funktion `deep-equal` avulla, joka palauttaa arvon tosi, jos kaksi sen parametrinaan saamaa solmua ovat täsmälleen samanlaisia (W3C, 2008e). Funktiolla `count` lasketaan kaikkien täsmälleen samanlaisten solmujen lukumäärä, jos niitä ei löydy, otetaan läpikäytävä tietoalkiojoukko mukaan tulokseen. Maksimaalisia sekvenssejä muodostavaa templatea kutustaan edellä esitetyllä tavalla muodostetulla tuloksella rekursiivisesti niin kauan, kunnes kaikki maksimaaliset sekvenssit on löydetty. Esimerkkiaineistolla muodostettu tulos on esitetty listauksessa 13.

```
<data>
      <litemsets>
            <litemset>
                  <item>1</item>
                  <item>2</item>
                  <item>3</item>
                  <item>4</item>
            </litemset>
            <litemset>
                  <item>1</item>
                  <item>5</item>
            </litemset>
      </litemsets>
</data>
```

Listaus 13. Enimmäissekvenssien etsinnän tulos.

4.3 Toistuvien rakenteiden louhinta

Toistuvien alirakenteiden louhinnan ensimmäisessä vaiheessa muodostetaan jokaisesta tutkittavasta XML-dokumentista asiakkaan `<transaktio>`-elementti, jonka tietoalkioihin

sijoitetaan puun solmut <item>-elementtiin. Puun solmut kuvataan asiakkaan, transaktion ja tietoalkion avulla, jotta edellisessä luvussa kuvattuja templateja voidaan suoraan käyttää hyväksi, vaikka esitystavan mukaan luonnollisempaa olisi kuvata jokainen XML-dokumentti esimerkiksi <document>-elementissä.

XML-dokumentti käydään läpi syvyyssuuntaisessa järjestyksessä listauksessa 14 kuvatun templaten avulla.

```
<xsl:template match="*">
      <item>
            <xsl:attribute name="path">
                  <xsl:for-each select="ancestor::*">
                        <xsl:value-of select="name()"/>/
                  </xsl:for-each>
                  <xsl:value-of select="name()"/>
            </xsl:attribute>
            <xsl:value-of select="name(.)"/>
            <xsl:value-of select="count(ancestor::*)"/>
      </item>
      <xsl:apply-templates select="*"/>
</xsl:template>
```

Listaus 14. Dokumentin läpikäynti syvyyssuuntaisessa järjestyksessä.

Syvyyssuuntaisen läpikäynnin edetessä jokaisesta solmusta muodostetaan <item>-elementti, jonka sisällöksi asetetaan merkkijono, joka muodostuu solmun nimestä ja solmun syvyyttä puussa (eli ts. edeltäjäsolmujen lukumäärää) ilmaisevasta kokonaisluvusta. Elementin attribuutin path arvoksi asetetaan puun polku. Tätä tietoa käytetään hyväksi myöhemmin muodostettaessa toistuvaa alirakennetta kuvaava XML-dokumentti.

Kohdassa 3.3.4 toistuvien alirakenteiden yhteydessä esitetystä kuvan 10 esimerkistä muodostettu XML-dokumentti on kuvattu listauksessa 15.

```
<data>
<customer>
<transaction>
      <item path="data">data0</item>
      <item path="data/movie">movie1</item>
      <item path="data/movie/title">title2</item>
      <item path="data/movie/title/item">item3</item>
```

```
      <item path="data/movie/film">film2</item>
      <item path="data/movie/film/writers">writers3</item>
      <item path="data/movie/film/cast">cast3</item>
      <item path="data/movie/film/cast/name">name4</item>
</transaction>
</customer>
<customer>
<transaction>
      <item path="data">data0</item>
      <item path="data/movie">movie1</item>
      <item path="data/movie/title">title2</item>
      <item path="data/movie/title/item">item3</item>
      <item path="data/movie/distributor">distributor2</item>
      <item path="data/movie/distributor/category">category3</item>
      <item path="data/movie/film">film2</item>
      <item path="data/movie/film/cast">cast3</item>
      <item path="data/movie/film/cast/name">name4</item>
</transaction>
</customer>
</data>
```

Listaus 15. XML-dokumenttien syvyyssuuntaisen läpikäynnin tulos.

Muodostettu XML-dokumentti annetaan syötteeksi sekvenssivaiheen toteuttavalle templatelle. Ainoa muutos kohdassa 4.2 esitettyyn ratkaisuun on se, että `path`-attribuutin arvo kopioidaan mukaan muodostettaviin `<item>` -elementteihin.

Sekvenssivaiheen tuloksena esimerkkisyötteestä syntyvä XML-dokumentti on kuvattu listauksessa 16. Minimitueksi on määritelty 2, eli rakenteen on esiinnyttävä kaikissa dokumenteissa.

```
<data>
   <litemsets>
      <litemset>
         <item path="data/movie">movie1</item>
         <item path="data/movie/title">title2</item>
         <item path="data/movie/film">film2</item>
         <item path="data/movie/film/cast">cast3</item>
      </litemset>
   </litemsets>
   <litemsets>
      <litemset>
         <item path="data/movie">movie1</item>
         <item path="data/movie/film">film2</item>
         <item path="data/movie/film/cast">cast3</item>
      </litemset>
      <litemset>
```

```
            <item path="data/movie">movie1</item>
            <item path="data/movie/title">title2</item>
            <item path="data/movie/film/cast">cast3</item>
         </litemset>
         <litemset>
            <item path="data/movie">movie1</item>
            <item path="data/movie/title">title2</item>
            <item path="data/movie/film">film2</item>
         </litemset>
         <litemset>
            <item path="data/movie/title">title2</item>
            <item path="data/movie/film">film2</item>
            <item path="data/movie/film/cast">cast3</item>
         </litemset>
      </litemsets>
      <litemsets>
         <litemset>
            <item path="data/movie/film">film2</item>
            <item path="data/movie/film/cast">cast3</item>
         </litemset>
         <litemset>
            <item path="data/movie">movie1</item>
            <item path="data/movie/film/cast">cast3</item>
         </litemset>
         <litemset>
            <item path="data/movie">movie1</item>
            <item path="data/movie/film">film2</item>
         </litemset>
         <litemset>
            <item path="data/movie">movie1</item>
            <item path="data/movie/title">title2</item>
         </litemset>
         <litemset>
            <item path="data/movie/title">title2</item>
            <item path="data/movie/film/cast">cast3</item>
         </litemset>
         <litemset>
            <item path="data/movie/title">title2</item>
            <item path="data/movie/film">film2</item>
         </litemset>
      </litemsets>
   </data>
```

Listaus 16. Sekvenssivaiheen tulos.

Enimmäissekvenssien generointivaihe saa syötteekseen sekvenssivaiheen tuloksen. Jälleen voidaan käyttää hyväksi kohdassa 4.2.5 esitettyä templatea, sillä poikkeuksella, että `deep-equal-`funktiota ei voida käyttää elementtien vertailussa hyväksi. Funktio `deep-equal` ottaa vertailussa huomioon myös attribuuttien arvot, joten toistuvien rakenteiden etsinnässä vertailu on suoritettava tietoalkioittain. Enimmäissekvenssien vaiheen tulos on esimerkkiaineistolla kuvattu listauksessa 17.

```
<data>
	<litemsets>
		<litemset>
			<item path="data/movie">movie1</item>
			<item path="data/movie/title">title2</item>
			<item path="data/movie/film">film2</item>
			<item path="data/movie/film/cast">cast3</item>
		</litemset>
	</litemsets>
</data>
```

Listaus 17. Enimmäissekvenssien etsinnän tulos.

Lopuksi voidaan elementtien attribuuttiin tallennetun elementin sijaintipolun perusteella muodostaa XML-dokumentti löydetystä toistuvasta rakenteesta. Lopullinen tulos on kuvattu listauksessa 18.

```
<tree>
	<data>
		<movie>
			<title />
			<film>
				<cast />
			</film>
		</movie>
	</data>
</tree>
```

Listaus 18. XML-dokumenttien toistuva rakenne.

4.4 XSLT 2.0 -toteutuksen suorituskyvyn analyysi

AprioriAll-algoritmin suorituskyvyn arvioimiseksi suoritettiin taulukossa 1 kuvatut testit. Taulukon 1 riveillä on kuvattu testin numero, ko. testin testiaineistossa olleiden asiakkaiden lukumäärä, kunkin asiakkaan keskimääräinen transaktioiden lukumäärä, kunkin transaktion keskimääräinen tietoalkioiden lukumäärä, testissä käytetty minimituki asiakkaiden lukumääränä, transformaatioon eli algoritmin suoritukseen käytetty aika sekunteina, sekä transformaatioprosessin keskusmuistista varaama suurin tila suorituksen kuluessa megatavuina. Taulukossa 2 on kuvattu AprioriAll-algoritmin sisällön louhinnassa käytetyn viiden eri vaiheen suhteelliset suoritusajat.

Testit on suoritettu tietokoneessa, jonka käyttöjärjestelmä on Microsoft® Windows Vista Home Premium (6.0.6000 N/A Build 6000), prosessori 800Mhz GenuineIntel (x64 Family 6 Model 15 Stepping 13), ja keskusmuistin koko 2 038 MB. XSLT-prosessorina on käytetty AltovaXML 2008 XSLT 2.0 Engine-prosessoria. Testiaineisto on muodostettu satunnaisesti.

Taulukko 1. AprioriAll-algoritmin XSLT 2.0 -toteutuksen suorituskykytestitulokset.

Testi-numero	**Asiak. lkm**	**Transakt. lkm**	**Tietoalk. lkm**	**Minimi-tuki (kpl)**	**Transf. Aika (s)**	**Muistin käyttö (MB)**
1	5	2	2	3	*0.61*	*8*
2	5	2	2	2	*1.84*	*32*
3	10	2	3	5	*3.56*	*95*
4	10	2	3	4	*7.94*	*228*
5	15	2	3	6	*9.24*	*296*
6	20	2	3	6	*16.02*	*436*
7	15	2	3	3	*33.04*	*1028*
8*	20	2	3	5	*>6 min.*	*99%*

* = virheellinen tulos

Suoritukseen käytetty aika on mitattu clTimer-ohjelman (CyLOG, 2008) avulla, ja keskusmuistin kulutuksen mittaukseen on käytetty Windows-käyttöjärjestelmän Resource Manager-ohjelmaa.

Testin tuloksista voidaan huomata, että transformaatioon käytetty aika ja transformaation käyttämä keskusmuistin määrä kulkevat käsi kädessä. Mitä enemmän algoritmi joutuu generoimaan kandidaattisekvenssejä, ja laskemaan niille tuen, sitä enemmän aikaa kuluu ja muistia varautuu. Kuten luvussa 2 mainittiin, tarkoitetaan tietokantojen tiheydellä olemassa olevien hahmojen ja mahdollisten sekvenssien suhdetta. Tiheyteen vaikuttavat (1) käytetty minimituki, sillä tiheys on sitä suurempi mitä alhaisempi minimituki on, koska silloin toistuvia sekvenssejä on enemmän, ja (2) tietoalkiojoukkojen lukumäärä tietokan-

nassa, sillä suurempi alkioiden lukumäärä johtaa potentiaalisesti suurempaan määrään erilaisia sekvenssejä (Antunes & Oliveira, 2004). Testituloksista voidaan todeta, että tiheyden kasvaessa algoritmin suoritukseen kulunut aika kasvaa. Taulukossa 1 kuvatussa testissä 8 on testin tuloksena syntynyt virheellinen tulos, joka johtuu siitä, että tietokoneen keskusmuistin kapasiteetti ei riittänyt sekvenssivaiheen tuloksen tuottamiseen. Algoritmin suoritus varasi enimmillään 99% koko keskusmuistin kapasiteetista. AprioriAll-algoritmin XSLT 2.0 -toteutuksen suorituskyvyn rajat tulevat siis vastaan jo hyvin pienillä tietomäärillä.

Algoritmin viiden eri vaiheen käyttämä suhteellinen suoritusaika on kuvattu taulukossa 2. Suhteellinen suoritusaika ilmaistaan prosenttiosuutena kokonaisajasta. Prosenttiosuudet on puolestaan laskettu taulukossa 1 kuvattujen testien 1 - 7 keskimääräisistä suoritusajoista. Sekvenssivaihe, jossa generoidaan kandidaattisekvenssit, ja lasketaan niiden tuki, on selkeästi koko toteutuksen suorituskyvyn kannalta haastavin osuus, sillä siihen kuluu keskimäärin 97% koko algoritmin suoritusajasta.

Taulukko 2. Algoritmin eri vaiheiden suhteelliset osuudet kokonaissuoritusajasta.

Vaihe	**Suhteellinen osuus kokonaissuoritusajasta**
Lajitteluvaihe	0.5 %
Suurten tietoalkioiden generointi	0.5 %
Muunnosvaihe	0.5 %
Sekvenssivaihe	97 %
Enimmäissekvenssien vaihe	2 %

5 LOPPUPÄÄTELMÄT

Testeissä käytetyt aineistot ovat hyvin pieniä, mutta kuitenkin niin suuria, että AprioriAll-algoritmin XSLT 2.0 -toteutuksen tehottomuus voidaan todeta. Suorituskyvyn heikkous ole ainoa hankala asia algoritmin toteuttamisessa XSLT 2.0:lla, sillä XSLT -prosessorien Altova (Altova, 2008), Saxon (Saxon, 2008) ja Xalan (Xalan, 2008) toiminnallisuudessa, eli siinä, kuinka ne suorittavat käsittelyä, on mielestäni useita merkittäviä eroja. AprioriAll-algoritmin XSLT 2.0 -toteutuksen kannalta keskeisten funktioiden `deep-equal` ja `count` tulokset eroavat tietyissä tapauksissa eri prosessoreissa toisistaan.

Havaintojeni mukaan esimerkiksi funktio `deep-equal` tuottaa alla olevalla enimmäissekvenssin toteutuksessa käytetyssä ehtolauseessa Saxon-prosessorissa aina tuloksen true, joka on väärin, mutta Altova-prosessorissa tulos on aina oikein.

```
<xsl:if test="count(
      $removable_litemsets/litemset[deep-equal(node(), $litemz)]) = 0">
```

Samoin ensimmäinen funktion `count` kustu alla olevassa suurten tietoalkiojoukkojen generointivaiheessa käytetyssä ehtolauseessa tuottaa Altova-prosessorissa aina tuloksen 0, kun taas Saxon-prosessorissa oikea tulos palautuu.

```
<xsl:if test="count($data/data/customer[
                  transaction[count(item[node() = $purged/item]) =
                              count($purged/item)]
                  ]) &gt;= $minsupport">
```

XSLT 2.0 -toteutuksen keskeinen ongelma on sekvenssivaiheen kuluttama suoritusaika ja sen vaatima muistitila. Yleisimmät ohjelmointikielet, kuten Java ja Visual Basic, ovat imperatiivisia ohjelmointikieliä, joiden keskeinen toimintaperiaate on muuttujien arvojen muuttaminen operaatioiden ja funktioiden avulla. XSLT 2.0 on puolestaan funktionaalinen ohjelmointikieli, joissa muuttujan arvo voidaan asettaa vain kerran kunkin rutiinin suorituksen aikana, ja niitä voidaan siten muuttaa ainoastaan saman rutiinin rekursiivisen kutsun kautta (Kay, 2004b). Rekursion suorittaminen puolestaan vaatii tunnetusti muisti-

tilaa, ja muistitilan tarve riippuu siitä, kuinka syvä rekursiosta kehittyy. XSLT 2.0 -toteutuksessa kandidaattien generoinnissa täytyy kuitenkin edetä rekursiivisesti. Tuen laskentaan kandidaatille joudutaan niin ikään käyttämään rekursioita, joten sekvenssivaiheessa käytetään kahta sisäkkäistä rekursiota. Tuen laskennassa joudutaan lisäksi jokaisen kandidaatille käymään koko tietokanta läpi, koska tuen arvoa voi tallentaa, vaan se on palautettava tuen laskevan template-kutsun arvona. Wanin ja Dobbien (2004) esittelemässä assosiaatiosääntöjen generointiin tarkoitetussa Apriori-algoritmin XQuery-toteutuksessa on kuvattu samanlainen ongelma yhtenä tehottomuuden syynä. Sekvenssivaiheen käsiteltävät tietojoukot ovat myös hyvin suuria. Näistä syistä sekvenssivaihe on XSLT 2.0 -toteutuksessa hyvin tehoton.

XSLT 2.0 -toteutusta on mahdollista tehostaa luvassa 2 mainituilla AprioriAll-algoritmin tehostamiskeinoilla, mutta mielestäni ne eivät poista perusongelmaa, eli kandidaattien generoinnin ja tuen laskennan aiheuttamaa muistinkulutusta. Mielestäni ainoa XSLT 2.0 -toteutuksen muistinkäytön ongelmien ratkaisu on jakaa sekvenssivaiheen toteutus edelleen pienempiin vaiheisiin, jotta kerralla muistissa pidettävien kandidaattien ja rekursiivisten kutsujen määrää voidaan vähentää. Käytännössä tämä tarkoittaa sekvenssivaiheen toteuttavan templaten jakamista osiin, ja niiden tuottamien välitulosten tallentamista keskusmuistin sijasta XML-dokumentteihin (tai SAX-streamiin, DOM-objektiin, jne.). Tämä puolestaan johtaa mielestäni siihen, että template-kutsuja täytyy hallinnoida jollain muulla ohjelmointikielellä toteutetun ohjelman avulla. Listauksessa 19 on esitetty pseudoalgoritmina eräs mahdollinen ratkaisu.

```
sekvenssit(1).xml = suuret_tietoalkiot.xml
for (k = 2; sekvenssit(k).xml ≠ ∅; k++) do
        begin
        kandidaatit(k).xml = XSLT(generoi_kandidaatit.xsl, sekvenssit(k-1).xml);
        sekvenssit(k).xml = XSLT(laske_tuki.xsl, kandidaatit(k).xml)
        end
```

Listaus 19. Sekvenssivaiheen osittaminen.

Tällä tavalla toteutetussa ratkaisussa kullakin kierroksella generoidaan kandidaatit yhdellä XSLT 2.0 -templatella, tallennetaan tulos kierroksen kandidaatit sisältävään XML-

dokumenttiin. XML-dokumentti puolestaan annetaan syötteenä tuen laskevalle templatelle, joka tuottaa kierroksen *k* sekvenssit omaan XML-dokumenttiin. Seuraavalla kierroksella kandidaatit muodostetaan tästä dokumentista. Kandidaattien generointia jatketaan niin kauan, kunnes *k*-sekvenssejä sisältävää XML-dokumenttia ei enää muodostu.

Mielestäni AprioriAll-algoritmin XSLT 2.0 -toteutuksen hyödyllisimmät ja käyttökelpoisimmat osat ovat ne osat, joissa lähtötieto voidaan käydä läpi yhdellä läpikäynnillä templaten sisällä, ja luoda tulosdokumentti tämän läpikäynnin perusteella. Nämä vaiheet ovat tietokannan lajitteluvaihe ja transformaatiovaihe. Niissä voidaan käyttää hyväksi myös XSLT 2.0:lle ominaisinta toiminnallisuutta - XML-dokumentin muuntamista toiseksi XML-dokumentiksi sen sisällön perusteella. Lisäksi toistuvien rakenteiden louhinnassa puun solmujen nimen ja syvyyden sijoittaminen sekvenssivaiheen lähtötiedoksi edellisessä luvussa kuvatulla tavalla on mielestäni tehokasta tehdä XSLT 2.0:n avulla, sillä `<xsl:apply-templates>`-elementti XSLT 2.0:ssa käy dokumentin läpi syvyyssuuntaisessa järjestyksessä (Kay, 2004b).

Sekvenssihahmojen louhinnan toinen ratkaisumenetelmä on luvussa 2 esitelty hahmonkasvatusmenetelmä. Se perustuu idealle välttää kandidaattien generointi ja testaus kokonaan, ja keskittää etsintä rajoitettuun alkuperäisen tietokannan osaan. Koska XML:n perusrakenne on puu, ja XSLT:n perusmenetelmä rekursio, on mielestäni mahdollista, että hahmonkasvatusmenetelmä sopii paremmin XSLT 2.0 -toteutukseksi sekvenssihahmojen louhintaan. Hahmonkasvatusmenetelmien keskeinen heikkous on kuitenkin niiden vaatima rekursioiden määrä projektoitujen tietokantojen luomisessa. Esimerkiksi PrefixSpan-algoritmi luo pahimmassa tapauksessa yhden projektoidun tietokannan jokaiselle sekvenssihahmolle (Pei & al., 2001a). AprioriAll-algoritmin XSLT 2.0 -toteutuksen tulosten perusteella joudutaan mielestäni myös hahmonkasvatusmenetelmän XSLT 2.0 -toteutuksessa samoihin ongelmiin muistin kulutuksen suhteen.

XSLT 2.0 -toteutus demonstroi mielestäni hyvin AprioriAll-algoritmin keskeiset heikkoudet - suuren generoitavien kandidaattisekvenssien ja tietokannan läpikäyntien määrän. Se myös osoittaa, että perinteisten sekvenssihahmojen louhinnan menetelmien soveltami-

sessa XML-muotoiseen tietoon, ja siihen liittyvien teknologioiden kehittämisessä, on vielä paljon avoimia tutkimusaiheita.

VIITELUETTELO

Agrawal, R., Imielinski, T., Swami, A. (1993) Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Conference 1993* (toim. Buneman, P., Jajodia, S.), ACM Press, Washington DC, USA, 207-216.

Agrawal, R., Srikant, R. (1994) Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of 20th International Conference on Very Large Data Bases* (toim. Bocca, J. B., Jarke, M., Zaniolo, C.), Morgan Kaufmann, Santiago de Chile, Chile, 487-499.

Agrawal, R., Srikant, R. (1995) Mining sequential patterns. *Proceedings of the 11th International Conference on Data Engineering* (toim. Yu, P., Chen, A.), IEEE Computer Society, Washington DC, USA, 3–14.

Altova (2008) *Altova XML*. WWW-sivusto, Altova,
http://www.altova.com/altovaxml.html (12.03.2008).

Antunes, C., Oliveira A. L. (2004) *Sequential pattern mining algorithms: Trade-offs between speed and memory*.
http://web.ist.utl.pt/claudia.antunes/artigos/mgts_pkdd2004.pdf (18.01.2008).

Boldakov, A. A., Grinev, M. N. (2006) Transformation of XML data using updates without side effects. *Programming and Computer Software* **32**(5), 255-267.

Brundage, M. (2004) *XQuery: The XML Query Language*. Addison-Wesley, Boston, MA, USA.

Buneman, P. (1997) Semistructured data. *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (toim.* Mendelzon, A., Özsoyoglu, Z. M. ACM Press, New York, NY, USA, 117–121.

Cheng, H., Yan, X., Han, J. (2004) IncSpan: Incremental mining of sequential patterns in large database. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (toim. Kim, W., Kohavi, R., Gehrke, J.), ACM Press, Seattle, USA, 527-532.

CYLOG (2008) *CyLog Tools*. WWW-sivusto, World Wide Web Consortium, http://www.cylog.org/tools/cmdline.jsp (09.04.2008).

Ding, Q., Sundarraj, G. (2006) *Association Rule Mining from XML Data*. *Proceedings of International Conference on Data Mining*, Las Vegas, Nevada, USA, 144-150.

Edmonds, A. (2003) *On data mining tree structured data in XML.* http://www.scientio.com/SiteCollectionDocuments/On_data_mining_tree_structured_data_in_XML.pdf (07.01.2008).

Feng, L., Dillon, T. (2004) Mining Interesting XML-Enabled Association Rules with Templates. *Lecture Notes in Computer Science, Knowledge Discovery in Inductive Databases*, **2005**(3377), 66-88.

Florescu, D., Kossmann, D. (1999) Storing and Querying XML Data Using an RDBMS *IEEE Data Engineering Bulletin, Special Issue on XML*, **22**(3), 27-34.

Han, J., Kamber, M. (2001) *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Fransisco, CA., USA

Hunter, D., Rafter, J., Fawcett, J., van der Vlist, E., Ayers, D., Duckett, J., Watt, A., McKinnon, L. (2007) *Beginning XML, 4th edition*. Wiley Publishing Inc, Indianapolis, Indiana, USA

Kappel, G., Kapsammer, E., Retschitzegger, W. (2004) Integrating XML and Relational

Database Systems. *World Wide Web* **7**(4), 343-384.

Kay, M. (2004a) *XPath 2.0 Programmer's Reference, 4th edition*. Wiley Publishing Inc, Indianapolis, Indiana, USA.

Kay, M. (2004b) *XSLT 2.0 Programmer's Reference, 4th edition*. Wiley Publishing Inc, Indianapolis, Indiana, USA.

Kum, H., Pei, J., Wang, W., Duncan, D. (2003) ApproxMAP: approximate mining of consensus sequential patterns. *Proceedings of the 3rd SIAM International Conference on Data Mining* (toim. Barbará, D., Kamath, C.) , SIAM, San Francisco, California, USA, 311-315.

Mannila, H., Toivonen, H., Verkamo, A. I. (1997) Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, **1**(3), 259-289.

Masseglia, F., Teisseire, M., and Poncelet, P. (2005) Sequential Pattern Mining. *Encyclopedia of Data Warehousing and Mining* (toim. Wang, J.), Idea Group Publishing, Covent Garden, London, UK.

Møller, A., Østerby Olesen, M., Schwartzbach, M.I. (2005) Static validation of XSL transformations. *ACM Transactions on Programming Languages and Systems,* **29**(4), 21.

Nayak, R., Witt, R., Tonev A. (2002) Data Mining and XML documents. *Proceedings of the 2002 International Conference on Internet Computing*. CSREA Press, Las Vegas, USA, 660-667.

Ouyang, W., Huang, Q. (2007) Mining Direct and Indirect Fuzzy Sequential Patterns in Large Transaction Databases. *Communications in Computer and Information Science Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques* (toim. Huang, D-S., Heutte, L., Loog, M.),

Springer, Berlin Heidelberg, Germany, 180-189.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu,.M.-C. (2001a) PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings of the 17th International Conference on Data Engineering.* IEEE Computer Society, Washington, DC, USA.

Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D., (2001b) H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. *Proceedings of the 2001 IEEE International Conference on Data Mining* (toim. Cercone, N., Young Lin, T., Wu, X.), IEEE Computer Society Washington, DC, USA, 441-448.

Pei, J., Han, J., Wang, W (2002) Mining sequential patterns with constraints in large database. *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*. ACM Press, McLean, Virginia, USA, 18-25.

Pinto, H., Han, J., Pei, J., Wang, K., Fraser, S., Chen, Q., Dayal, U. (2001) Multidimensional Sequential Pattern Mining. *Proceedings of the 10th Conference on Information and Knowledge Management* (toim. Paques, H., Liu, L., Grossman, D.), ACM Press, New York, NY, USA, 81–88.

Saxon (2008) *The XSLT and XQuery Processor*. WWW-sivusto, SAXON, http://saxon.sourceforge.net/ (09.02.2008).

Scardina, M., Chang, B., Wang, J. (2004) *Oracle Database 10g XML & SQL: Design, Build & Manage XML Applications in Java, C, C++ & PL/SQL*. McGraw-Hill Osborne Media, Emeryville, CA, USA.

Srikant, R., Agrawal, R. (1996) Mining Sequential Patterns: Generalizations and Performance Improvements. *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology* (toim. Apers, P., Bouzeghoub,

M., Gardarin, G.) Springer-Verlag, London, UK, 3-17.

Zaki, M. J. (1997) Fast Mining of Sequential Patterns in Very Large Databases. *Department of Computer Science, University of Rochester, Techical Report 668*. Department of Computer Science, Univeristy of Rochester, Rochester, New York, USA.

Zaki, M. J. (2005) Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, **17**(8), 1021 - 1035.

Yan, X., Han, J., Afshar, R. (2003) CloSpan: mining closed sequential patterns in large datasets. *Proceedings of the 3rd SIAM international conference on data mining* (toim. Barbar, D., Kamath, C.), SIAM, San Francisco, CA, USA, 166-177.

Yang, J., Wang, W., Yu, P.S. (2001) Infominer: Mining Surprising Periodic Patterns. *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining* (toim. Provost, F., Srikant, R.), ACM New York, NY, USA, 395-400.

W3C (2008a) *Extensible Markup Language (XML).* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/XML/ (02.01.2008).

W3C (2008b) *World Wide Web Consortium.* WWW-sivusto, World Wide Web Consortium, http://www.w3.org/ (16.04.2008).

W3C (2008c) *XQuery 1.0: An XML Query Language*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/xquery/ (02.01.2008).

W3C (2008d) *XML Path Language (XPath) 2.0*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/xpath20/ (02.01.2008).

W3C (2008e) *XSL Transformations (XSLT) Version 2.0*. WWW-sivusto, World Wide Web Consortium, http://www.w3.org/TR/xslt20/ (02.01.2008).

Wan, J. W. W., Dobbie, G. (2004) Mining association rules from XML data using XQuery. *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation* (toim.. Hogan, J., Montague, P., Purvis, M., Steketee, C.),Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 169-174.

Wilde, E. (2006) *XML-Centric Application Development*. TIK Report 242, Computer Engineering and Networks Laboratory, ETH Zürich, Switzerland.

Xalan (2008) The Apache Xalan Project. WWW-sivusto, The Apache XML Project, http://xalan.apache.org/ (09.04.2008).

Zhang, M., Yao, J.T. (2004) XML Algebra for Data Mining. *Proceedings of the SPIE - The International Society for Optical Engineering*, **5433**(25), 209-217.

Liite: AprioriAll-algoritmin XSLT 2.0 –toteutuksen lähdekoodi

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
sort.xml

KUVAUS:
Lajitteluvaihe

SYÖTE:
Alkuperäinen tietokanta
-->

<xsl:output method="xml"/>
<xsl:template match="data">

<data>
<xsl:for-each-group select="customer" group-by="@id">
<xsl:sort select="@id"/>

<customer>
      <xsl:attribute name="id">
            <xsl:value-of select="@id"/>
      </xsl:attribute>

      <xsl:for-each
            select="../customer/transaction
                  [../@id = current-grouping-key()]">
      <xsl:sort select="@time" data-type="number"/>

            <transaction>
                  <xsl:attribute name="time">
                        <xsl:value-of select="@time"/>
                  </xsl:attribute>
                  <xsl:copy-of select="*"/>
            </transaction>
      </xsl:for-each>

</customer>
</xsl:for-each-group>

</data>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
litemset.xml

KUVAUS:
Suurten alkiojoukkojen generointi

SYÖTE:
Lajitteluvaiheen tulos
-->

<xsl:output method="xml" />
<xsl:variable name="data" select="document('sorted_database.xml')" />
<xsl:variable name="settings" select="document('settings.xml')" />
<xsl:variable name="minsupport"
            select="$settings/settings/minsupport" />


<xsl:template name="sort" match="data">
<data>
<xsl:call-template name="generate_litemsets">
      <xsl:with-param name="litemsets">

            <xsl:for-each-group
                  select="customer/transaction/item" group-by="node()">
                  <xsl:sort select="node()" />

                  <xsl:variable name="support"
                        select="count(//data/customer[transaction/item
                                    = current-grouping-key()])" />

                  <xsl:if test="$support &gt;= $minsupport">
                        <litemset>
                              <item>
                                    <xsl:value-of select="node()" />
                              </item>

                        </litemset>
                  </xsl:if>
            </xsl:for-each-group>

      </xsl:with-param>
</xsl:call-template>
</data>
</xsl:template>


<xsl:template name="generate_litemsets">
<xsl:param name="litemsets" />
<xsl:copy-of select="$litemsets" />
```

```
<xsl:variable name="sets">

      <xsl:for-each select="$litemsets/litemset">

            <xsl:variable name="tag"
                        select="item" />
            <xsl:variable name="tail"
                        select="item[position() = last()]" />
            <xsl:variable name="head"
                        select="item[position() != last()]" />
            <xsl:variable name="position"
                        select="position()" />
            <xsl:for-each select="
                  ../litemset[(position() &gt; $position) and
                   (count(item[$head=node()]) = count($head)) and
                   ($tail != item[position() = last()])] ">

                  <xsl:variable name="purged">
                  <xsl:for-each
                        select="distinct-values($tag | item)">
                        <item>
                        <xsl:value-of select="."/>
                        </item>
                  </xsl:for-each>
                  </xsl:variable>

                  <xsl:variable name="counter">
                        <xsl:for-each
                              select="$data/data/customer">
                        <customers>
                              <xsl:for-each
                                    select="transaction">
                                    <xsl:if
                                    test="count(item[node()
                                    = $purged/item])
                                    = count($purged/item)">
                                    <supportc>
                                    <xsl:value-of
                                    select="count(item[node()
                                          = $purged/item])
                                          = count($purged/item)" />
                                    </supportc>
                                    </xsl:if>
                              </xsl:for-each>
                        </customers>
                  </xsl:for-each>
                  </xsl:variable>

                  <xsl:if
                        test="count($counter/customers/supportc)
                              &gt;= $minsupport">
                        <litemset>
                              <xsl:copy-of select="$purged" />
                        </litemset>
                  </xsl:if>
            </xsl:for-each>
      </xsl:for-each>
```

```
</xsl:variable>

<xsl:if test="count($sets/litemset) &gt; 0">
      <xsl:call-template name="generate_litemsets">
            <xsl:with-param name="litemsets" select="$sets" />
      </xsl:call-template>
</xsl:if>

</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">


<!--
TIEDOSTO:
litemset.xml

KUVAUS:
Muunnosvaihe

SYÖTE:
Alkuperäinen tietokanta
-->

<xsl:output method="xml"/>
<xsl:variable name="data" select="document('litemsets.xml')" />

<xsl:template match="data">
<data>
      <xsl:for-each select="customer">
      <customer>
            <xsl:for-each select="transaction">
                  <xsl:variable name="transaction">
                        <xsl:call-template name="mapping">
                              <xsl:with-param name="items" select="*"/>
                        </xsl:call-template>
                  </xsl:variable>
                  <xsl:if test="count($transaction/item) &gt; 0">
                        <transaction>
                              <xsl:copy-of select="$transaction"/>
                        </transaction>
                  </xsl:if>
            </xsl:for-each>
      </customer>
      </xsl:for-each>
</data>
</xsl:template>

<xsl:template name="mapping">
<xsl:param name="items"/>

<xsl:for-each select="$data/data/litemset">
      <xsl:sort select="item[position() = last()]"/>
      <xsl:if test="count(item[$items = node()]) = count(item)">
            <item><xsl:value-of select="position()"/></item>
      </xsl:if>
</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
recursion.xml

KUVAUS:
Sekvenssivaihe

SYÖTE:
Muunnosvaiheen tulos
-->

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:variable name="settings" select="document('settings.xml')" />
<xsl:variable name="minsupport" select="$settings/settings/minsupport"
/>
<xsl:variable name="database" select="document('mapped_database.xml')"
/>

<xsl:template match="/">
      <xsl:variable name="joukot">
      <xsl:call-template name="loop">
            <xsl:with-param name="litemsets">
            <litemsets>
            <xsl:for-each-group
                  select="data/customer/transaction/item"
                  group-by="node()">
                  <xsl:sort select="node()"/>

                  <xsl:variable name="support"
                              select="count(
                              //data/customer[transaction/item
                              = current-grouping-key()])"/>

                        <xsl:if test="$support &gt;= $minsupport">
                        <litemset>
                              <item>

                              <!--
                              Toistuvien rakenteiden etsinnässä
                              lisätään attribuutti
                              <xsl:attribute name="path">
                              <xsl:value-of select="@path"/>
                              </xsl:attribute>
                              -->
                              <xsl:value-of select="node()"/>
                              </item>
                        </litemset>
                        </xsl:if>
            </xsl:for-each-group>
            </litemsets>
```

```
            </xsl:with-param>
            <xsl:with-param name="data" select="data"/>
      </xsl:call-template>
      </xsl:variable>

      <data>
      <xsl:copy-of select="$joukot"/>
      </data>

</xsl:template>



<xsl:template name="loop">

      <xsl:param name="litemsets"/>
      <xsl:param name="data"/>
      <xsl:variable name="sets">
      <litemsets>
      <xsl:for-each select="$litemsets/litemsets/litemset">

            <xsl:variable name="ptag">
            <candidate>
            <xsl:for-each select="item">

                  <!--
                  Toistuvien rakenteiden etsinnässä
                  lisätään attribuutti
                  <item path="{@path}">
                  -->

                  <item>
                  <xsl:value-of select="node()"/>
                  </item>
            </xsl:for-each>
            </candidate>

            </xsl:variable>
            <xsl:variable name="phead"
                        select="item[position() != last()]"/>

            <xsl:for-each select="$litemsets/litemsets/litemset">
                  <xsl:variable name="qtail"
                              select="item[position() = last()]"/>
                  <xsl:if
                        test="count(item[$phead=node()])
                        = count($phead)
                        and not($ptag/candidate/item = $qtail)">

                              <xsl:variable name="union">
                              <union>
                              <xsl:for-each
                                    select="$ptag/candidate/item">

                                    <!--
                                    Toistuvien rakenteiden etsinnässä
                                    lisätään attribuutti
```

```
      <item path="{@path}">
      -->
      <item>
      <xsl:value-of select="node()"/>
      </item>
</xsl:for-each>

<!--
Toistuvien rakenteiden etsinnässä
lisätään attribuutti
<item path="{$qtail/@path}">
-->

<item>
<xsl:value-of select="$qtail"/>
</item>
</union>
</xsl:variable>

<xsl:variable name="delete">
<xsl:call-template name="prune">
<xsl:with-param name="candidate"
                  select="$union"/>
<xsl:with-param name="litemsets"
                  select="$litemsets"/>
</xsl:call-template>
</xsl:variable>


<xsl:if test="
      not(exists($delete/supportc))">

      <xsl:call-template
            name="count_support">

      <xsl:with-param name="customer"
            select="$data/customer"/>
      <xsl:with-param name="k"
            select="1"/>
      <xsl:with-param name="litem">
      <litemset>
      <xsl:for-each

      select="$ptag/candidate/item">

      <!--
      Toistuvien rakenteiden etsinnässä
      lisätään attribuutti
      <item path="{@path}">
      -->
      <item>
      <xsl:value-of select="node()"/>
      </item>
      </xsl:for-each>

      <!--
      Toistuvien rakenteiden etsinnässä
```

```
                                    lisätään attribuutti
                                    <item path="{$qtail/@path}">
                                    -->

                                    <item>
                                    <xsl:value-of select="$qtail"/>
                                    </item>
                                    </litemset>
                                    </xsl:with-param>
                                    <xsl:with-param name="i"
                                          select="1"/>
                                    <xsl:with-param name="j"
                                          select="1"/>
                                    <xsl:with-param name="counter"
                                          select="0"/>
                                    <xsl:with-param name="support"
                                          select="0"/>
                                    </xsl:call-template>
                              </xsl:if>
                        </xsl:if>
                  </xsl:for-each>
            </xsl:for-each>
            </litemsets>
            </xsl:variable>


            <xsl:if test="count($sets/litemsets/litemset) &gt; 0">

                  <xsl:call-template name="loop">
                        <xsl:with-param name="litemsets" select="$sets"/>
                        <xsl:with-param name="data" select="$data"/>
                  </xsl:call-template>

                  <litemsets>
                  <xsl:copy-of select="$sets/litemsets/litemset"/>
                  </litemsets>
                  <support>
                  <xsl:copy-of select="$sets/litemsets/support"/>
                  </support>
            </xsl:if>

      </xsl:template>

      <xsl:template name="count_support">

            <!-- mäpätty tietokanta -->
            <xsl:param name="customer"/>
            <!-- monesko asiakas käsittelyssä, laskuri -->
            <xsl:param name="k"/>
            <!-- käsiteltävä litemset -->
            <xsl:param name="litem"/>
            <!-- käsittelyssä olevan litemsetin alkion
                  positiolaskuri i -->
            <xsl:param name="i"/>
            <!-- Käsittelyssä olevan asiakkaan
                  transaktion alkion positiolaskuri j -->
            <xsl:param name="j"/>
```

```
<!-- asiakkaan litemsetien laskuri -->
<xsl:param name="counter"/>
<!-- puoltolaskuri -->
<xsl:param name="support"/>

<!-- jos käsittelemättömiä asiakkaita vielä jäljellä -->
<xsl:if test="$k &lt;= count($customer)">

      <!-- käsittelyvuorossa olevan asiakkaan litemsetit -->
      <xsl:variable name="temp"
            select="$customer[position() = $k]"/>
      <!-- käsittelyvuorossa olevan asiakkaan
            litemsettien alkiot -->
      <xsl:variable name="citem"
            select="$temp/transaction/item"/>

      <!--
      jos käsittelemättömiä litemsettejä asiakkaalla
      vielä jäljellä
      ja litemsettien alkioiden lkm >= litemsetin alkioiden lkm
      -->
      <xsl:choose>
      <xsl:when test="($j &lt;= count($citem))
      and (count($citem) &gt;= count($litem/litemset/item))">

            <!-- käsittelyvuorossa olevan asiakkaan
            litemsetin alkio j -->
            <xsl:variable name="cnode"
                  select="$citem[position() = $j]"/>

            <!-- jos käsittelemättömiä alkioita litemsettissä
            vielä jäljellä -->
            <xsl:if test="$i &lt;= count($litem/litemset/item)">

                  <!-- seuraava alkio i käsittelyyn -->
                  <xsl:variable name="lnode"
                        select="$litem/litemset/item
                        [position() = $i]"/>

                  <!-- jos alkiot ovat samoja -->
                  <xsl:if test="$lnode = $cnode">

                        <!-- kutsutaan rekursiivisesti templatea
                              kasvatetaan laskuria ja
                              otetaan seuraava
                              positio  i käsittelyyn-->
                        <xsl:call-template name="count_support">
                              <xsl:with-param name="customer"
                                    select="$customer"/>
                              <xsl:with-param name="k"
                                    select="$k"/>
                              <xsl:with-param name="litem"
                                    select="$litem"/>
                              <xsl:with-param name="i"
                                    select="$i + 1"/>
                              <xsl:with-param name="j"
                                    select="$j + 1"/>
```

```
                        <xsl:with-param name="counter"
                              select="$counter + 1"/>
                        <xsl:with-param name="support"
                              select="$support"/>
                  </xsl:call-template>
            </xsl:if>

            <!-- jos alkiot eivät ole samoja -->
            <xsl:if test="$lnode != $cnode">

                  <!-- kutsutaan rekursiivisesti templatea
                        otetaan seuraava positio
                        käsittelyyn.
                        laskuria ei kasvateta-->
                  <xsl:call-template name="count_support">
                        <xsl:with-param name="customer"
                              select="$customer"/>
                        <xsl:with-param name="k"
                              select="$k"/>
                        <xsl:with-param name="litem"
                              select="$litem"/>
                        <xsl:with-param name="i"
                              select="$i"/>
                        <xsl:with-param name="j"
                              select="$j + 1"/>
                        <xsl:with-param name="counter"
                              select="$counter"/>
                        <xsl:with-param name="support"
                              select="$support"/>
                  </xsl:call-template>
            </xsl:if>
      </xsl:if>

      <!-- jos kaikki alkiot i käsitelty -->
      <xsl:if test="$i &gt; count($litem/litemset/item)">


                  <!-- kutsutaan rekursiivisesti templatea
                        otetaan seuraava
                        positio j käsittelyyn.
                        asetetaan i alkuun.
                        Laskuria ei kasvateta-->
                  <xsl:call-template name="count_support">
                        <xsl:with-param name="customer"
                              select="$customer"/>
                        <xsl:with-param name="k"
                              select="$k"/>
                        <xsl:with-param name="litem"
                              select="$litem"/>
                        <xsl:with-param name="i"
                              select="1"/>
                        <xsl:with-param name="j"
                              select="$j + 1"/>
                        <xsl:with-param name="counter"
                              select="$counter"/>
                        <xsl:with-param name="support"
                              select="$support"/>
```

```
                  </xsl:call-template>
      </xsl:if>
</xsl:when>
<!-- jos kaikki alkiot j käsitelty -->

<xsl:otherwise>

      <!-- jos kaikki asiakkaan litemsetit  käsitelty,
            ja puoltoa on löytynyt
            (laskurin counter arvo yhtäsuuri kuin
            litemsetin alkioiden lkm)-->
      <xsl:choose>
      <xsl:when
            test="($counter =
                  count($litem/litemset/item))">

            <!-- kutsutaan rekursiivisesti templatea
                  otetaan seuraava litemset käsittelyyn.
                  asetetaan alkio laskurit ja
                  positiot alkuun,
                  kasvatetaan puoltoa yhdellä-->
            <xsl:call-template name="count_support">
                  <xsl:with-param name="customer"
                        select="$customer"/>
                  <xsl:with-param name="k"
                        select="$k + 1"/>
                  <xsl:with-param name="litem"
                        select="$litem"/>
                  <xsl:with-param name="i" select="1"/>
                  <xsl:with-param name="j" select="1"/>
                  <xsl:with-param name="counter"
                        select="0"/>
                  <xsl:with-param name="support"
                        select="$support + 1"/>
            </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
      <!-- jos kaikki asiakkaan litemsetit  käsitelty,
            ja puoltoa ei löytynyt

            kutsutaan rekursiivisesti templatea
            otetaan seuraava litemset käsittelyyn.
            asetetaan alkio laskurit ja positiot alkuun
            -->
            <xsl:call-template name="count_support">
                  <xsl:with-param name="customer"
                        select="$customer"/>
                  <xsl:with-param name="k"
                        select="$k + 1"/>
                  <xsl:with-param name="litem"
                        select="$litem"/>
                  <xsl:with-param name="i" select="1"/>
                  <xsl:with-param name="j" select="1"/>
                  <xsl:with-param name="counter"
                        select="0"/>
                  <xsl:with-param name="support"
                        select="$support"/>
```

```
                        </xsl:call-template>
                  </xsl:otherwise>
                  </xsl:choose>
            </xsl:otherwise>
            </xsl:choose>
      </xsl:if>

      <!-- kun kaikki asiakkaat käsitelty,
            asetetaan puollon arvo litemsetille-->
      <xsl:if test="$k &gt; count($customer)">

            <xsl:if test="$support &gt;= $minsupport">
                  <xsl:copy-of select="$litem"/>
                  <support>
                  <xsl:value-of select="$support"/>
                  </support>
            </xsl:if>


      </xsl:if>

</xsl:template>

<xsl:template name="prune">
<xsl:param name="candidate"/>
<xsl:param name="litemsets"/>

<xsl:for-each select="$candidate/union/item">
      <xsl:variable name="subseq"
            select="remove($candidate/union/item, position())"/>
      <xsl:if
            test="count($litemsets/litemsets/litemset[(count(item)
            = (count($candidate/union/item)-1))
            and deep-equal(item,$subseq)]) = 0">
            <supportc></supportc>
      </xsl:if>
</xsl:for-each>
</xsl:template>


</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
maximal.xml

KUVAUS:
Enimmäissekvenssien vaihe

SYÖTE:
Sekvenssivaiheen tulos
-->

<xsl:output method="xml"/>
<xsl:template match="/">
<xsl:variable name="result">
<xsl:call-template name="do">
<xsl:with-param name="maximal"
      select="count(data/litemsets
            [position() = 1]/litemset[position() = 1]/item)"/>
<xsl:with-param name="position" select="1"/>
<xsl:with-param name="data" select="/"/>
</xsl:call-template>
</xsl:variable>

<xsl:copy-of select="$result"/>
</xsl:template>



<xsl:template name="do">
<xsl:param name="maximal"/>
<xsl:param name="data"/>
<xsl:param name="position"/>


      <xsl:variable name="removable_litemsets">
      <xsl:for-each select="$data/data/litemsets/litemset
                                    [(count(item) = $maximal)
                                    and (position() = $position)]">
            <xsl:call-template name="loop">
                  <xsl:with-param name="maximal" select="$maximal"/>
                  <xsl:with-param name="subseqs">
                        <subseqs>
                        <xsl:call-template name="counter">
                        <xsl:with-param name="i" select="1"/>
                        <xsl:with-param name="p">
                              <xsl:call-template name="pow">
                              <xsl:with-param name="n"
                                    select="$maximal"/>
                              <xsl:with-param name="base" select="1"/>
                              </xsl:call-template>
```

```
                  </xsl:with-param>
                  <xsl:with-param name="k" select="$maximal"/>
                  </xsl:call-template>
                  </subseqs>
            </xsl:with-param>
      </xsl:call-template>
</xsl:for-each>
</xsl:variable>

<xsl:variable name="output">
<data>
<litemsets>
<xsl:copy-of select="$data/data/litemsets/litemset
                              [(count(item) &gt;= $maximal)]"/>
</litemsets>

<xsl:for-each select="$data/data/litemsets">
<litemsets>

<xsl:for-each select="litemset[(count(item) &lt; $maximal)
                              and (count(item) &gt; 1)]">
            <xsl:variable name="litemz" select="node()"/>

            <!--

            Toistuvien rakenteiden etsinnässä
            käytetään tätä kommentoitua lohkoa

            <xsl:variable name="temp1">
            <temp>
            <xsl:for-each select="$litemz">
                  <item><xsl:value-of select="node()"/></item>
            </xsl:for-each>
            </temp>
            </xsl:variable>

            <xsl:variable name="temp2">
            <xsl:for-each select="$removable_litemsets/litemset">
                  <xsl:variable name="temp3">
                  <temp>
                  <xsl:for-each select="temp/node()">
                        <item>
                        <xsl:value-of select="node()"/>
                        </item>
                  </xsl:for-each>
                  </temp>
                  </xsl:variable>
                  <xsl:if
                        test="deep-equal($temp3, $temp1)">
                        <litemset>
                        <xsl:copy-of select="$temp1"/>
                        </litemset>
                  </xsl:if>

            </xsl:for-each>
            </xsl:variable>
```

```
            <xsl:if test="count($temp2/litemset) = 0">
                  <litemset>
                  <xsl:copy-of select="$litemz"/>
                  </litemset>
            </xsl:if>

             -->

            <!--
            Toistuvien rakenteiden etsinnässä
            seuraavaa if-lohkoa ei käytetä,
            vaan asetetaan se kommentteihin
             -->

            <xsl:if
                  test="count($removable_litemsets/litemset
                        [deep-equal(node(), $litemz)]) = 0">
                        <litemset>
                        <xsl:copy-of select="$litemz"/>
                        </litemset>
            </xsl:if>
            </xsl:for-each>
      </litemsets>
</xsl:for-each>
</data>
</xsl:variable>

<xsl:choose>
      <xsl:when test="$maximal &gt; 1">
            <xsl:choose>
            <xsl:when test="count($data/data/litemsets/litemset
                   [(count(item) = $maximal)
                  and (position() = $position + 1)])
                  &gt; 0">
                  <xsl:call-template name="do">
                  <xsl:with-param name="maximal"
                              select="$maximal"/>
                  <xsl:with-param name="position"
                              select="$position + 1"/>
                  <xsl:with-param name="data"
                              select="$output"/>
                  </xsl:call-template>
                  </xsl:when>
                  <xsl:otherwise>
                        <xsl:call-template name="do">
                              <xsl:with-param name="maximal"
                                          select="$maximal - 1"/>
                              <xsl:with-param name="position"
                                          select="1"/>
                              <xsl:with-param name="data"
                                          select="$output"/>
                        </xsl:call-template>
                  </xsl:otherwise>
                  </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
                  <xsl:copy-of select="$output"/>
```

```
            </xsl:otherwise>
      </xsl:choose>
 </xsl:template>



<xsl:template name="loop">
<xsl:param name="maximal"/>
<xsl:param name="subseqs"/>

<xsl:variable name="data" select="/"/>
      <xsl:for-each select="$subseqs/subseqs/subseq
                        [count(item) &gt; 1]">
            <xsl:variable name="subseq" select="node()"/>
            <xsl:for-each
                        select="$data/data/litemsets/litemset
                              [(count(item) &lt; $maximal)
                              and (count(item) &gt; 1)]">


            <!--

            Toistuvien rakenteiden etsinnässä
            käytetään tätä kommentoitua lohkoa

            <xsl:variable name="temp1">
            <temp>
            <xsl:for-each select="node()">
            <item><xsl:value-of select="node()"/></item>
            </xsl:for-each>
            </temp>
            </xsl:variable>

            <xsl:variable name="temp2">
            <temp>
            <xsl:for-each select="$subseq">
            <item><xsl:value-of select="node()"/></item>
            </xsl:for-each>
            </temp>
            </xsl:variable>

            <xsl:if test="deep-equal($temp2, $temp1)">
            <litemset><xsl:copy-of select="$temp1"/></litemset>
            </xsl:if>
            -->

            <!--
            Toistuvien rakenteiden etsinnässä
            seuraavaa if-lohkoa ei käytetä,
            vaan asetetaan se kommentteihin
                  -->
            <xsl:if test="deep-equal($subseq, item)">
                  <litemset>
                  <xsl:copy-of select="item"/>
                  </litemset>
            </xsl:if>
```

```
      </xsl:for-each>
</xsl:for-each>
</xsl:template>


<xsl:template name="counter">
<xsl:param name="i"/>
<xsl:param name="p"/>
<xsl:param name="k"/>

<xsl:if test="$i &lt; ($p - 1)">
      <subseq>
      <xsl:call-template name="subsets">
            <xsl:with-param name="i" select="$i"/>
            <xsl:with-param name="j" select="1"/>
            <xsl:with-param name="k" select="$k"/>
      </xsl:call-template>
      </subseq>
      <xsl:call-template name="counter">
            <xsl:with-param name="i" select="$i + 1"/>
            <xsl:with-param name="p" select="$p"/>
            <xsl:with-param name="k" select="$k"/>
      </xsl:call-template>
</xsl:if>
</xsl:template>

<xsl:template name="subsets">
<xsl:param name="i"/>
<xsl:param name="j"/>
<xsl:param name="k"/>

<xsl:if test="$j &lt;= $k">
      <xsl:choose>
            <xsl:when test="not($i &lt; 1)">
                  <xsl:call-template name="subsets">
                        <xsl:with-param name="i" select="$i div 2"/>
                        <xsl:with-param name="j" select="$j + 1"/>
                        <xsl:with-param name="k" select="$k"/>
                  </xsl:call-template>
                  <xsl:if test="($i mod 2) &gt;= 1">
                        <item>
                        <xsl:value-of
                              select="item
                              [position() = (($k + 1) - $j)]"/>
                        </item>
                  </xsl:if>
            </xsl:when>

            <xsl:otherwise>
                  <xsl:call-template name="subsets">
                        <xsl:with-param name="i" select="0"/>
                        <xsl:with-param name="j" select="$j + 1"/>
                        <xsl:with-param name="k" select="$k"/>
                  </xsl:call-template>
            </xsl:otherwise>

      </xsl:choose>
```

```
</xsl:if>
</xsl:template>

<xsl:template name="pow">
      <xsl:param name="n"/>
      <xsl:param name="base"/>
      <xsl:if test="$n &gt; 0">
            <xsl:call-template name="pow">
                  <xsl:with-param name="n"
                        select="$n - 1"/>
                  <xsl:with-param name="base"
                        select="$base * 2"/>
            </xsl:call-template>
      </xsl:if>
      <xsl:if test="$n = 0">
            <xsl:value-of select="$base"/>
      </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
treetraversal.xml

KUVAUS:
Puun läpikäynti

SYÖTE:
XML-dokumentti
-->

<xsl:output method="xml"/>

<xsl:template match="/">
       <customer>
                  <transaction>
                        <xsl:apply-templates/>
                  </transaction>
            </customer>
</xsl:template>


<xsl:template match="*">
<item>
<xsl:attribute name="name">
<xsl:value-of select="name(.)"/>
</xsl:attribute>
<xsl:attribute name="depth">
<xsl:value-of select="count(ancestor::*)"/>
</xsl:attribute>
<xsl:attribute name="path">
<xsl:for-each select="ancestor::*">
      <xsl:value-of select="name()"/>/
</xsl:for-each>
<xsl:value-of select="name()"/>
</xsl:attribute>
<xsl:value-of select="name(.)"/>
<xsl:value-of select="count(ancestor::*)"/>
</item>

<xsl:apply-templates select="*"/>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/TR/xpath-functions"
      xmlns:xdt="http://www.w3.org/TR/xpath-datatypes">

<!--
TIEDOSTO:
maketree.xml

KUVAUS:
XML-puun muodostaminen

SYÖTE:
Rakenteen louhinnassa
tuotetut enimmäissekvenssit
-->


<xsl:output method="xml"/>

 <xsl:template match="data/litemsets/litemset">
     <tree>
         <xsl:call-template name="process">
             <xsl:with-param name="depth" select="1"/>
             <xsl:with-param name="seq" select="item/@path"/>
         </xsl:call-template>
     </tree>
 </xsl:template>

 <xsl:template name="process">
     <xsl:param name="depth"/>
     <xsl:param name="seq"/>
     <xsl:for-each-group select="$seq"
            group-by="tokenize(.,'/')[$depth]">
         <xsl:variable name="part" select="tokenize(., '/')[$depth]"/>
         <xsl:element name="{$part}">
              <xsl:call-template name="process">
                  <xsl:with-param name="depth"
                        select="$depth + 1"/>
                  <xsl:with-param name="seq"
                        select="
                        $seq[tokenize(., '/')[$depth] = $part]
                        "/>
              </xsl:call-template>
         </xsl:element>
     </xsl:for-each-group>
 </xsl:template>

 </xsl:stylesheet>
```