Polyfonisen musiikin haku sisällön perusteella

Kjell Lemström
Helsingin yliopisto
Tietojenkäsittelytieteen laitos
klemstro@cs.helsinki.fi

1 Johdanto

Altavista, Google, Hotbot, ... — kaikki, joille hiirestä tulee mieleen muutakin kuin jyrsijä, tietävät näiden olevan Internet-hakukoneita tekstidokumenttien etsimiseksi. Samoja hakukoneita käytetään jatkuvasti myös muun muotoisten tiedostojen hakuun. Viime aikoina musiikkidokumenttien (esimerkiksi mp3-tiedostojen) etsintä näillä hakukoneilla on ylittänyt suosiollaan jopa perinteisesti suosituimmat "K18-kategorian" sivustojen haut.

Multimedian, kuten musiikki- ja kuvadokumenttien, hakemiseen tekstihakukoneilla liittyy kuitenkin selkeitä ongelmia: haut joudutaan kohdistamaan dokumentteihin liitettyihin teksteihin eikä itse sisältöön. Vaikka tällaisia hakuja varten multiediadokumentteihin voidaan liittää niitä kuvaavia avainsanoja, se vaatii liikaa ihmistyötä. Lisäksi täydellisen avainsanaluettelon laatiminen on mahdotonta; sanojen assosiointi kuviin tai musiikkiin on hyvin persoonallinen prosessi.

Kuvien hakua sisällön perusteella on tutkittu aktiivisesti jo vuosikymmenen ajan. Esimerkiksi IBM on kehittänyt kaupallisen QBIC-kuvanhakujärjestelmän (Query by Image Content). Järjestelmällä voidaan hakea kuvia tietokannasta esimerkiksi hahmottelemalla halutussa kuvassa esiintyvän objektin ääriviivoja tai antamalla arvio kuvan värijakaumasta. Musiikinhakuongelman nuorempi pioneerijärjestelmä on Uuden Seelannin Waikaton yliopiston digitaalikirjastoprojektin yhteydessä kehitetty MELDEX-musiikinhakujärjestelmä (The New Zealand Digital Library MELody inDEX)[1].

Musiikin sisältöön liittyvät kyselyongelmat voidaan karkeasti jakaa kahteen päätyyppiin. Ensimmäisessä, *tunnistusongelmassa*, on tarkoituksena automaattisesti tunnistaa soitetun audiotiedoston (kuten mp3- tai

[1] http://nzdl2.cs.waikato.ac.nz/cgi-bin/gwmm?c=meldex.

Kuva 1: Esimerkki länsimaisesta nuotinnoksesta polyfonisessa musiikissa.

wav-tiedoston) esittäjä, kappaleen nimi, musiikin tyylisuunta, jne. Tämän ongelman ratkaisut perustuvat pääosin signaalinkäsittelytekniikoihin. Tässä kirjoituksessa käsiteltävä ongelma on *hakuongelma*, jossa etsitään annetun musiikillisen (esim. hyräillyn) hahmon esiintymiä musiikkidokumenteissa. Tässä tapauksessa käsiteltävä musiikki on useimmiten symbolimuotoista ja ratkaisut perustuvat yleensä kombinatorisiin merkkijonoalgoritmeihin. Nyrkkisääntönä näiden kahden ongelman käsittelemille tiedostomuodoille voidaan pitää sitä, että symbolimuotoinen musiikki (tai tiedosto) on ohje soittajille akustisen lopputuloksen aikaansaamiseksi. Akustinen lopputulos voidaan lopuksi tallentaa audiotiedostoksi.

Symbolisessa muodossa esiintyvästä musiikista tunnetuinta on länsimainen nuottikirjoitus (kts. kuva 1). Se ei kuitenkaan ole digitaalista, eikä siksi sellaisenaan sovellu meidän tarkoitukseemme. Lähes kaikki symbolista lähestymistapaa käyttävät musiikinhakumenetelmät käsittelevät MIDI-muotoisia (tai niistä muokattuja) tiedostoja. MIDI (Musical Instrument Digital Interface) on itse asiassa edellä esitettyjen kahden tiedostomuodon välimuoto, sillä vaikka MIDI-tiedosto sisältää soitto-ohjeen (mutta ei audiotietoa), tiedosto kuitenkin yleensä tuotetaan soittamalla.

Musiikinhakumenetelmien sovellusmahdollisuudet lisääntyvät lähitulevaisuudessa selvästi, jos mm. laitevalmistajat hyväksyvät ja ottavat käyttöönsä parhaillaan kehitteillä olevan MPEG7-standardin[2]. Standardissa määritellään ns. metatason kuvauksia multimediadokumenttien sisällöille. Musiikin osalta ne tulevat olemaan hyvin samankaltaisia myöhemmin tässä kirjoituksessa esiteltävien kuvausten kanssa. Standardin myötä tulevaisuuden digitaaliset tallenteet tulevat mahdollisesti sisältämään audiodatan lisäksi sitä kuvaavaa metadataa ja näiden välisen linkityksen, jolloin esimerkiksi radioasemien ja levy-yhtiöiden musiikkiarkistojen penkominen helpottuisi nykyisestä huomattavasti. Tavallisen kuluttajan

[2] http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm.

WYHIWYG-sovelluksia ("What You Hum is What You Get") löytyisi esimerkiksi levykaupoista, missä asiakas voisi etsiä tai koostaa haluamansa levyn hyräilemällä pätkän musiikkikappaleesta, ja cd-soittimista, missä kymmeniä tai satoja cd-levyjä sisältävää soitinta voisi ohjata hyräilemällä muistamatta levyn ja sen raidan nimeä tai numeroa. Baarissa musiikin voisi tilata jukeboxista hyräilemällä matkapuhelimeensa. WYHIWYG-sovellusten lisäksi eräät musiikkitieteelliset analyysiongelmat tarjoavat samankaltaisia mutta vaikeampia haasteita.

Vaikka MELDEXillä ja muilla musiikinhakujärjestelmillä voidaan jo nyt hakea esimerkiksi hyräillyn hakuavaimen likimääräisiä esiintymiä suurista tietokannoista vieläpä suhteellisen tehokkaastikin, riittää alalla vielä paljon tutkittavaa ja kehitettävää. Symbolista lähestymistapaa käyttävät ohjelmat ja algoritmit perustuvat lähes poikkeuksetta ns. merkkijonoalgoritmeihin, jotka on alunperin kehitetty tekstimuotoisten hahmojen paikallistamiseksi laajoista tekstimassoista. Vaikka symbolimuotoinen musiikki muodostuukin merkkijonoista, monet musiikille tyypilliset piirteet eivät ole relevantteja tekstimerkkijonoille, eikä niitä siksi ole otettu huomioon merkkijonoalgoritmeja kehitettäessä. Tekstimerkkijonojen yhteydessä on vaikea nähdä mitään samankaltaisuutta esimerkiksi merkkijonojen $s_1 = (c, d, e)$ ja $s_2 = (f, g, a)$ välillä. Musiikkiyhteydessä nämä saavat kuitenkin uuden tulkinnan; molemmat merkkijonot koostuvat kolmesta sävelestä, joiden väliset intervallit muodostuvat kahdesta puolisävelaskelesta. Länsimaiselle ihmiselle on tyypillistä musiikin analyyttinen kuunteleminen, missä absoluuttisen sävelkorkeuden sijaan kuunnellaan sävelten korkeuksien välisiä erotuksia. Musiikkiyhteyksissä edellä olevat merkkijonot tulkitaan useissa tapauksissa yhtäpitäviksi. Tätä länsimaiselle musiikille ominaista piirrettä kutsutaan *transpositioinvarianssiksi*.

Toinen musiikille tyypillinen piirre, joka on relevantti musiikkihahmon mutta ei tekstihahmon etsinnässä, on musiikin moniäänisyys eli *polyfonisuus*. Kyselyn esittäjä saattaa etsiä hahmon "hajautettua esiintymää"[3]: kuvan 1 musiikkinäytteestä voitaisiin esimerkiksi etsiä esiintymää hajautetulle hahmolle p, jossa esiintyvät peräkkäin sävelet (b, g, d, d, c, a). Tehtävän selventämiseksi kuvassa 2 on annettu kuvan 1 musiikkinäytteestä yksinkertaistettu tekstiesitys, jossa on otettu huomioon ainoastaan oktaavin alueelle typistetyt sävelkorkeudet. Lisäksi ylin ääni on rytmitetty uudelleen alimpien äänten mukaisesti.

Merkkijonoalgoritmit soveltuvat sellaisinaan yksinkertaisimpiin *monofonisen* eli yksiäänisen musiikin ha-

[3] Esimerkiksi Edward Elgarin *Cockaignessa* melodia on hajautettu eri instrumenttien kesken.

f	$\boxed{g}$	b	b	c	d	c	b
d	$c\sharp$	$\boxed{d}$	$\boxed{d}$	$e\sharp$	e	d	d
$\boxed{b}$	g	b	b	b	$\boxed{a}$	b	b
b	$e\flat$	b	b	$\boxed{c}$	f	b	b

Kuva 2: Kuvan 1 musiikkinäyte yksinkertaistettuna tekstimerkkijonoksi. Kuvassa korostetaan hahmon $p = (b, g, d, d, c, a)$ hajautettua esiintymää.

kuongelmiin. Transpositioinvarianteiksi algoritmit saadaan yksinkertaisesti koodaamalla merkkijonot erotuksina eli *intervalleina* absoluuttisten sävelkorkeuksien sijaan (yllä esitetyt merkkijonot s_1 ja s_2 molemmat koodattaisiin tällöin intervallijonona $(2, 2)$). Merkkijonoalgoritmien muokkaaminen polyfoniseen tapaukseen on huomattavasti haastavampi tehtävä. Tämä on kuitenkin käytännön sovelluksissa välttämätöntä, sillä valtaosa länsimaisesta musiikista on polyfonista. Tällä hetkellä yksikään Internetistä löytyvä musiikinhakujärjestelmä ei pysty etsimään musiikkihahmoa polyfonisesta tietokannasta.

Tämän kirjoituksen tarkoituksena on esitellä, mihin musiikinhakualgoritmit perustuvat, ja antaa vertaileva katsaus kirjallisuudessa esitetyille toistaiseksi harvoille polyfonisen musiikin hakualgoritmeille.

2 Merkkijonoista

Polyfonisen musiikin hakualgoritmien toiminnan ymmärtämiseksi esitellään seuraavassa merkkijonoalgoritmien hahmontunnistukseen liittyvää käsitteistöä ja perustietoa.

2.1 Notaatiosta

Olkoon Σ äärellisen kokoinen merkkien joukko, jota kutsutaan *aakkostoksi*. Aakkoston Σ merkkijonot ovat muotoa $A = (a_1, a_2, \ldots, a_m)$, missä jokainen a_1 kuuluu aakkostoon Σ. Jatkossa tällaisesta merkkijonosta käytetään lyhyempää muotoa: $A = a_1 \cdots a_m$. Merkkijonon A pituus on $|A| = m$. Merkkijonoa, jonka pituus on 0, kutsutaan tyhjäksi merkkijonoksi ja sitä merkitään symbolilla λ. Σ^i tarkoittaa aakkoston Σ niiden merkkijonojen joukkoa, joiden pituus on i. Kaikkien aakkoston Σ merkkijonojen joukkoa merkitään symbolilla Σ^*. Olkoon merkkijono A muotoa $A = \alpha\beta\gamma$, missä $\alpha, \beta, \gamma \in \Sigma^*$. Tällöin α, β ja γ ovat A:n *tekijöitä*. Merkkijono α on myös A:n *alkuosa* ja γ sen *loppuosa*. Merkkijono A' on A:n *alisekvenssi*, jos $A' = A$ tai A' saadaan muodostetuksi poistamalla merkkijonon A merkkejä; toisin sanoen jos $A' = a_{i_1} a_{i_2} \cdots a_{i_w}$, missä $i_1 \ldots i_w$ on A:n indekseistä muodostuva kasvava jono.

2.2 Hahmontunnistusongelmia

Perinteisessä merkkijonojen hahmontunnistusongelmassa etsitään hahmon $p = p_1 \cdots p_m$ esiintymää tekstistä $t = t_1 \cdots t_n$. Hahmolla p on tarkka esiintymä tekstin kohjassa j, jos $p_i = t_{j+i-1}$ pätee arvoilla $i = 1, \ldots, m$. Hahmolle voidaan etsiä myös likimääräisiä esiintymiä ottamalla käyttöön sopiva etäisyysmitta, ns. *editointietäisyys* (kts. esimerkiksi [4]). Tällöin merkkijonoja voidaan korjailla käyttämällä ennalta määriteltyjä *editointioperaatioita*. Yleisimmin käytetyt editointioperaatiot ovat:

- *muutos*: kohdistetut merkit p_i ja t_j eivät ole samoja;
- *poisto*: merkillä t_j ei ole vastinetta hahmossa p;
- *lisäys*: merkillä p_i ei ole vastinetta tekstissä t.

Jos kaikkien edellä mainittujen operaatoiden käyttäminen on hahmontunnistuksessa sallittua, on kyseessä *k-eroavaisuuden ongelma*. Jos muutosoperaatio on ainoa sallittu editointioperaatio, on kyseessä *k-epätäsmäyksen ongelma*. Käyttämällä sallittuja operaatioita saadaan hahmo p muunnetuksi miksi tahansa tekstin tekijäksi p' (k-epätäsmäyksen tapauksessa vaaditaan kuitenkin, että $|p| = |p'|$). Koska k-eroavaisuuden tapauksessa muunnos voidaan tehdä usealla eri tavalla, määritellään p:n ja p':n editointietäisyydeksi $D(p, p')$ minimi tarvittavien editointioperaatioden määrästä[4].

2.3 Algoritmien toteutuksesta

Sekä tarkkaan että likimääräiseen hakuun on kehitetty runsaasti erilaisia algoritmeja. Nämä algoritmit voidaan jakaa *online*- ja *offline*-menetelmiin. Online-menetelmät lukevat syötettä järjestyksessä merkki kerrallaan ja pystyvät antamaan palautetta syötettä lukiessaan (esimerkiksi tulostamaan hahmon esiintymän heti esiintymän viimeisen merkin luettuaan). Offline-menetelmät toimivat vasta koko syötteen saatuaan: tyypillisimmillään ne rakentavat esiprosessina tekstiä indeksoivan rakenteen, jonka avulla hahmon etsintä voidaan myöhemmin toteuttaa vähintään kertaluokkaa online-menetelmiä nopeammin. Indeksi saattaa kuitenkin vaatia huomattavasti muistitilaa, joten usein on perustellumpaa rakentaa esiprosessissa vaatimattomampia apurakenteita, vaikka ne eivät mahdollistaisikaan alilineaarisia hakuaikoja.

[4]Tässä on yksinkertaisuuden vuoksi oletettu editointioperaatioille yksikkökustannukset. Tarvittaessa operaatioihin voidaan liittää mitkä tahansa positiiviset reaalilukuarvoiset kustannukset käyttämällä *painofunktioita*.

2.3.1 Online-menetelmiä

Hahmon $p = p_1 \cdots p_m$ likimääräisten esiintymien etsiminen tekstissä $t = t_1 \cdots t_n$ voidaan toteuttaa *dynaamisella ohjelmoinnilla*. Tällöin etsitään t:n sellaisia tekijöitä p', joilla pätee $D(p, p') \leq k$. Tässä kynnysparametrin k arvo säätelee haun tarkkuustasoa. Haku toteutetaan muodostamalla taulukko (d_{ij}), jossa lasketaan kaikkien hahmon p alkuosien ja tekstin t tekijöiden väliset etäisyydet $d_{ij} = D(p_1 \cdots p_i, t_{j'} \cdots t_j)$ $(j' \leq j)$. k-eroavaisuus-ongelman taulukko (d_{ij}) saadaan muodostetuksi ajassa, joka on verrannollinen tuloon nm (lyhyemmin merkkittynä ajassa $O(nm)$) iteroimalla palautuskaavaa:

$$\begin{aligned}
d_{0j} &= 0 \quad (0 \leq j \leq n); \\
d_{i0} &= i \quad (0 \leq i \leq m); \\
d_{ij} &= \min \begin{cases} d_{i,j-1} + 1 \\ d_{i-1,j} + 1 \\ d_{i-1,j-1} + (\text{jos } p_i = t_j \\ \qquad \text{niin } 0, \text{ muuten } 1). \end{cases}
\end{aligned} \tag{1}$$

Taulukossa elementin d_{mj} arvo ilmaisee hahmon pienimmän editointietöisyyden sellaiseen t:n tekijään, joka loppuu kohtaan j. Näin ollen jos $d_{mj} \leq k$, hahmolla on tekstissä t likimääräinen esiintymä, joka loppuu kohtaan t_j.

Tietyissä tapauksissa dynaamisen ohjelmoinnin taulukointia voidaan simuloida erittäin tehokkailla *bittirinnakkaisalgoritmeilla*. Siinä missä perinteisessä tapauksessa lasketaan yksi taulukon elementti d_{ij} kerrallaan, bittirinnakkaistoteutuksessa pyritään kerralla laskemaan taulukon koko sarake d_j. Ensimmäisiä bittirinnakkaisalgoritmejä oli tarkan hahmon etsintään kehitetty, ajassa $O(n\lceil \frac{m}{W} \rceil)$ toimiva ShiftOr-algoritmi [1] (edellä W on tietokonesanan koko bitteinä).

ShiftOr käyttää laskennassa apunaan $|\Sigma|$ bittivektorin taulukkoa T, jossa kukin bittivektori sisältää m bittiä. Jokaista hahmon p merkkiä c_l vastaa sarake $\mathtt{T}[l]$, jossa $\mathtt{T}[l].r = 0$ jos ja vain jos $p_r = c_l$ (muuten $\mathtt{T}[l].r = 1$). Toisin sanoen $\mathtt{T}[l]$ esittää, mitkä hahmon kohdat sisältävät merkin c_l. Simulaatiossa m bitin kokoinen bittivektori E esittää dynaamisen ohjelmoinnin laskennan kohteena olevaa taulukon (d_{ij}) saraketta. Olkoon $\mathtt{E}_j$ vektorin E arvo j tekstin symbolin käsittelyn jälkeen. $\mathtt{E}_j.i = 0$, jos ja vain jos $p_1 \cdots p_i = t_{j-i+1} \cdots t_j$. Toisin sanoen $\mathtt{E}_j$:stä nähdään kaikki kohtaan j loppuvat p:n alkuosat. Siirryttäessä laskennassa seuraavan sarakkeen käsittelyyn, $\mathtt{E}_{j+1}$:n bitit saadaan palautuskaavasta:

$$\mathtt{E}_{j+1}.i = \begin{cases} 0, \text{ jos } \mathtt{E}_j.(i-1) = 0 \\ \quad \text{ja } p_i = t_{j+1}, \\ 1, \text{ muuten.} \end{cases} \tag{2}$$

Sarakkeen laskenta on toteutettu käyttämällä bittisiirtoa (shift) ja loogista OR-operaatiota, mistä seuraa algoritmin nimi.

Jos $p_i = t_{j+1}$ pätee, palautuskaava (2) laajentaa kaikkia edellisessä askelessa löytyneitä hahmon alkuosan esiintymiä jatkamalla vastaavia diagonaaleissa esiintyviä 0-jonoja askelen pidemmälle. Hahmolla on kohtaan t_j loppuva esiintymä, jos $\mathtt{E}_j.m = 0$.

2.3.2 Offline-menetelmiä

Tarkkaan hahmonsovitukseen voidaan käyttää esimerkiksi *loppuosapuita* [4]. Tällöin kaikki tekstin loppuosat talletetaan puurakenteeseen, josta minkä tahansa hahmon esiintymät löydetään ajassa $O(m)$. Vaikka loppuosapuut vaativat vain lineaarisen $O(n)$ tilan, kasvavat puut käytännön sovelluksissa usein aivan liian suuriksi, kuten esimerkiksi yritettäessä indeksoida tällä tavalla kaikki Internetistä löytyvät MIDI-tiedostot [8].

Puiden kokoa voidaan tehokkaasti rajata tallettamalla ainoastaan tekstin kaikki q merkin mittaiset tekijät, ns. q-grammit. Tällöin tilatarve pienenee huomattavasti, tosin suoritusajan kustannuksella. Toinen saavutettu hyöty q-grammien käytöstä on likimääräisten hakujen mahdollistumisessa. Parametrin q arvo saadaan määritetyksi tarvittavan virhesietoisuuden perusteella (k:n arvo on kääntäen verrannollinen q-grammien pituuden suhteen).

Esiprosessoinnin tarkoituksena ei välttämättä ole indeksirakenteen muodostaminen. Monilla algoritmeilla on tarve muokata saamaansa syötettä toisenlaiseen, algoritmin toiminnan kannalta välttämättömään muotoon. Näissä tapauksissa rakennetaan usein myös joitakin apurakenteita, jotka eivät kuitenkaan yleensä vaadi niin paljon tilaa kuin indeksirakenteet.

3 Musiikki digisymmuotoiseksi

Musiikkitietokannan saa helposti koostetuksi "imuroimalla" digitaalisia, symbolimuotoisia musiikkidokumentteja Internetistä, mistä niitä löytyy tuhansittain. Näistä *digisym-tiedostoista* valtaosa on MIDI-tiedostoja. MIDI-tiedostojen huono puoli on se, että niiden laadusta ei ole takuita; tiedostot voi olla tuotettu soittamalla, jolloin ne voivat sisältää runsaasti virheitä. Varmempi, joskin huomattavasti työläämpi lähtökohta on koostaa tietokanta konvertoimalla nuottikirjoituksia MIDI-tiedostoiksi optisella nuotinlukuohjelmalla. Tällöin tietokanta koostuu juuri halutuista kappaleista, joiden laatu on tarkasti selvillä.

Vaikka joissakin digisym-koodeissa pyritään monikäyttöisyyden takia esittämään mahdollisimman tarkasti kaikki nuotinnoksen informaatio, nuottikirjoituksiin verrattuna digisymkoodaukset eivät ole yhtä ilmaisuvoimaisia. Tässä kirjoituksessa käsiteltävissä algoritmeissa käytetään hyvin yksinkertaista esitystä, jossa mukana ovat vain kaikkein oleellisimmat musiikilliset piirteet.

Mitkä piirteet ovat sitten hakujen kannalta kaikkein olennaisimmat? Luonnollisimmat vaihtoehdot ovat nuotin (äänen) korkeus, sen kesto tai ne molemmat. Länsimaisessa musiikissa nuotin korkeuksiin perustuvat intervallijonot ovat erottelukyvyiltään paremmat kuin kestoihin perustuvat rytmijonot. Paras erottelukyky edellä mainituista vaihtoehdoista saadaan luonnollisesti ottamalla huomioon sekä korkeus- että kestoinformaatio. Ongelmaksi tällöin kuitenkin muodostuu etenkin populaarimusiikille tyypillinen piirre: tunnetuimmista musiikkikappaleista on uusia versioita eri tyylisuunnille. Usein tyylisuunnan muutos vaikuttaa nimenomaan kappaleen rytmiin (siis nuottien kestoihin), ei niinkään intervallijonoihin (sävelkorkeuksiin). Tämän vuoksi piirteiden yhteinen erottelukyky on joskus liiankin tarkka, jolloin haettavaa dokumenttia ei välttämättä tietokannasta löydetä, vaikka se siellä olisikin.

Tarkasteltavien polyfonisen musiikin hakualgoritmien kannalta riittävä esitysmuoto nuotti-informaatiolle on kolmikko $\langle q, t, v\rangle$, jossa q (pitch) ilmaisee nuotin korkeuden, t (on-set time) sen syttymishetken ja v (voicing) mihin ääneen nuotti kuuluu ($q, t, v \in Z_+$). Kaikki algoritmit eivät kuitenkaan välttämättä käytä kaikkea tätä informaatiota (varsinkaan komponenttia v), tai ne saattavat prosessoida syöteinformaatiota haluamaansa muotoon. Erittäin tyypillistä on, että algoritmit prosessoivat komponenttien q ja t arvoalueita. Esimerkiksi typistämällä q:n arvoaluetta (MIDI:ssä 0–127) käytetään "yleistettyjä" sävelkorkeuksia, minkä seurauksena voidaan sallia virheitä hakuavaimessa, vaikka sovellettaisiin tehokkaita tarkkaan hakuun kehitettyjä algoritmeja. Peräkkäisten nuottien väliset intervalliarvot voidaan typistää esimerkiksi arvoalueelle $\{-1, 0, 1\}$, missä -1 edustaa negatiivisia, 0 *priimiä* (eli 0-intervallia) ja 1 kaikkia positiivisia intervalleja. Komponentin t arvoaluetta puolestaan pienennetään usein niin, että täsmällisten nuottien syttymishetkien sijaan käytössä on ainoastaan syttymishetkien välinen osittainen järjestys (samalla hetkellä syttyviä nuotteja ei järjestetä).

4 Polyfonisia hakuongelmia

Useimmiten musiikin merkkijonoesityksenä käytetään muotoa, jossa otetaan huomioon äänen korkeuden lisäksi vain syttymishetkien osittainen järjestys (kuvassa 2 tämä muoto esiintyy hieman yksinkertaistettuna). Koska musiikin yhteydessä on harhaanjohtavaa puhua hahmon etsimisestä tekstistä, käytetään jatkossa tekstin sijasta termiä *lähde*.

Polyfonisissa musiikinhakuongelmissa lähde $S = S_1 \cdots S_n$ muodostuu h rinnakkaisesta merkkijonosta, ns. *äänestä*: $s^i = s^i_1 \cdots s^i_n, i \in \{1, \ldots, h\}$, missä h on vakio (esimerkiksi kuvassa 2 $h = 4$)[5]. Polyfonisen musiikin

[5]Merkinnän S_j voisi ajatella esittävän lähteen "poikkileikkausta" kohdassa j ($1 \leq j \leq n$)

erikoistapuksessa, ns. *homofonisessa musiikissa*, kaikissa äänissä uusi nuotti syttyy samanaikaisesti. Kuva 2 esittää kuvan 1 "homofonisoitua" digisymversiota.

Yksinkertaisimmissa hakutapauksissa aiemmin esitetyt merkkijono-ongelmat ovat sellaisinaan relevantteja myös polyfonisen musiikin haussa. Näin on mm. sellaisissa tapuksissa, kun hahmon p esiintymiä etsitään minkä tahansa yhden äänen sisältä (tällöin äänet voidaan katenoida peräkkäin ja soveltaa merkkijonoalgoritmeja sellaisenaan) tai kun hahmokin on polyfoninen ja sen jokaisen polyfonisen elementin pitää täsmätä sellaisenaan lähteen polyfonisiin elementteihin. Tällöin tarvitaan isompaa aakkostoa, jossa kutakin mahdollista polyfonista elementtiä (sointua) vastaa yksi aakkoston merkki. Huomaa, että näin ei kuitenkaan löydetä polyfonisen hahmon esiintymiä, joissa hahmon soinnut ovat lähteen vastaavien sointujen osajoukkoja.

Edellä mainitut triviaalit erikoistapaukset eivät kata kaikkia relevantteja polyfonisen musiikin hakuongelmia. Perinteisillä merkkijonoalgoritmeilla ei voida ratkaista esimerkiksi kuvassa 2 esiintyvää *hajautetun hahmon etsintäongelmaa*. Hahmolla p on hajautettu esiintymä lähteen kohdassa j, jos $p_1 = s_j^{i_1}, p_2 = s_{j+1}^{i_2}, \ldots, p_m = s_{j+m-1}^{i_m}$ kun $\{i_1, \ldots, i_m\} \in \{1, \ldots, h\}$. Vielä relevantimpi ja samalla haastavampi ongelma on etsiä hahmon transpositioinvarianttia hajautettua esiintymää: $p_1 = s_j^{i_1} + c, p_2 = s_{j+1}^{i_2} + c, \ldots, p_m = s_{j+m-1}^{i_m} + c$, missä c on kokonaislukuarvoinen vakio.

Näistä tarkan haun ongelmista voidaan muodostaa likimääräisiä hakuongelmia, joihin voidaan soveltaa editointietäisyyttä. Esimerkiksi poisto-operaation yksikkökustannus ei välttämättä kuitenkaan ole musiikillisesti kovin mielekäs. Tietokannassa esiintyvä musiikkidokumentti saattaa sisältää runsaasti erilaisia musiikillisia "koristeita", kuten kadensseja, korusäveliä, jne. Sovellettaessa editointietäisyyttä parametrin k arvo voidaan helposti määrittää liian pieneksi (esimerkiksi haun tehokkuuden takia). Tällöin korukuvioiset esiintymät saattavat jäädä löytymättä. Usein onkin mielekkäämpää antaa hahmon komponenttien esiintyä lähteessä niin, että lähteen vastaavien komponenttien välillä saa esiintyä "hyppyjä" (editointietäisyyden kannalta "ilmaisia" poistoja). Hahmolla p on lähteessä S *t-hyppyesiintymä*, jos $p_1 = s_{j_1}^{i_1}, p_2 = s_{j_2}^{i_2}, \ldots, p_m = s_{j_m}^{i_m}$ ($\{i_1, \ldots, i_m\} \in \{1, \ldots, h\}$) ja $j_{l+1} - j_l \leq t + 1$ ja $1 \leq l \leq m - 1$. Vaikka parametrille t on helpompi löytää musiikillisesti mielekäs arvo kuin k:lle, sama ongelma saattaa toistua edelleen. Musiikillisesti mielekkäin vaihtoehto onkin usein määräämättömän mittaisten hyppyjen salliminen.

ja s^i lähteen i:nnettä "pitkittäisleikkausta" ($1 \leq i \leq h$).

5 Hakualgoritmeja

Seuraavaksi esitellään lyhyesti tähän mennessä kirjallisuudessa esitettyjä ratkaisuja polyfonisen musiikin hakemiseksi sisällön perusteella. Esiteltävistä seitsemästä menetelmästä kaksi käyttää indeksointirakenteita ja viisi perustuu merkkijonoalgoritmeihin.

Algoritmeille, jotka eivät käytä indeksirakenteita, vertailukohtana voidaan pitää "naiiviin merkkijonoalgoritmiin" [4, p. 34] perustuvaa algoritmia, joka löytää hajautetun hahmon transpositioinvariantit esiintymät ajassa $O(nh(h+m))$ [9]. Naiivi menetelmä indeksointiin olisi indeksoida kaikki lähteen mahdolliset monofoniset merkkijonokombinaatiot. Vaikka tämä johtaisikin ajassa $O(m)$ toimivaan hakumenetelmään, suurilla n:n tai h:n arvoilla toteutus tulee mahdottomaksi kombinaatioiden lukumäärän ollessa h^n.

5.1 BD-algoritmi

Bloch ja Dannenberg esittelivät algoritmin automaattisen säestejän tahdistamiseksi polyfonisen soolosoittimen kanssa [2]. Tässä myös polyfonisen musiikin hakuun soveltuvassa algoritmissa automaattisella säestäjällä on tiedossaan solististemman nuotinnoksesta tehty osittaisjärjestetty digisymesitys. BD-algoritmi etsii polyfonista (soitettua) hahmoa polyfonisesta digisym-lähteestä dynaamisen ohjelmoinnin keinoin (ilman transpositioinvarianssia): Soolosoittimen yksittäisiä ääniä etsitään yksitellen osittaisjärjestetystä lähteestä. Polyfonisesta hahmosta, jossa on m sointua, tehdään hakua varten täysin järjestetty monofoninen versio, jossa on m' säveltä. Varsinaisen taulukon (d_{ij}) lisäksi laskennassa käytetään aputaulukkoa $(\mathcal{D}_{ij})$, jossa alkio $\mathcal{D}_{ij}$ on niiden $p_1 \cdots p_i$ loppuosan merkkien joukko, jotka sisältyvät sointuun S_j. BD-algoritmissa d_{ij} edustaa symbolijonojen etäisyyksien sijasta niiden yhtäläisyyttä. Alustuksilla $d_{0j} = d_{i0} = 0$ saadaan loput arvot ratkaistuksi palautuskaavasta (vrt. yhtälöön (1)):

$$d_{ij} = \max \begin{cases} (i) \ d_{i,j-1} - |S_{j-1} - \mathcal{D}_{i,j-1}| \\ (ii) \ d_{i-1,j} + (\text{jos } p_i \in S_j \\ \quad \text{ja } p_i \notin \mathcal{D}_{i-1,j} \text{ niin } 1, \\ \quad \text{muuten } -\infty) \\ (iii) \ d_{i-1,j-1} + 1 - \\ \quad |S_{j-1} - \mathcal{D}_{i-1,j-1}|. \end{cases} \tag{3}$$

Taulukkoa $(\mathcal{D}_{ij})$ päivitetään taulukon (d_{ij}) kanssa synkronisesti. Seuraavassa palautuskaavassa sovelletaan jokaisessa päivitysaskelessa sitä vaihtoehtoa, jonka numerointi vastaa yhtälössä (3) maksimi-operaation valitseman vaihtoehdon numeroa (alustus: $\mathcal{D}_{0j} = \mathcal{D}_{i0} = \emptyset$):

$$\mathcal{D}_{ij} = \begin{cases} (i) & \{p_i\} \\ (ii) & \mathcal{D}_{i-1,j} \bigcup \{p_i\} \\ (iii) & \emptyset. \end{cases}$$

Jos palautuskaavassa (3) useampi rivi antaa saman tuloksen, valitaan maksimin antavista vaihtoehdoista se, jonka numerointi on pienin. Tätä preferenssiä tarvitaan, jotta $(\mathcal{D}_{ij})$:n elementit olisivat yksikäsitteisiä.

Sovellettaessa BD-algoritmia polyfonisen musiikin hakuun hahmon likimääräiset esiintymät löydetään tutkimalla taulukon (d_{ij}) rivin m' arvoja. Esiintymisille voidaan asettaa ehdoksi esimerkiksi se, että $d_{ij} \geq (m' - k')$ pätee jollakin sopivalla vakiolla k'[6].

5.2 HIM-algoritmi

ShiftOr-algoritmi voidaan hyvin yksinkertaisesti muuntaa etsimään monofonisen hahmon hajautettuja esiintymiä polyfonisessa lähteessä [7, 9]. ShiftOrAnd-algoritmi toteuttaa palutuskaavan (vrt. kaavaan (2)):

$$\mathtt{E}_{j+1}.i \quad = \quad \begin{cases} 0, & \text{jos } \mathtt{E}_j.(i-1) = 0 \\ & \text{ja } p_i \in S_{j+1}, \\ 1, & \text{muuten.} \end{cases} \tag{4}$$

Taulukko T muodostetaan analogisesti ShiftOr-algoritmin kanssa: jokaista hahmossa p esiintyvää merkkiä c_l vastaa sarake $\mathtt{T}[l]$, jossa $\mathtt{T}[l].r = 0$ jos ja vain jos $p_r = c_l$ (muuten $\mathtt{T}[l].r = 1$). Soinnulle S_i saadaan bittivektoriesitys yhdistelemällä sointuun kuuluvia komponentteja vastaavia T:n sarakkeita loogisella AND-operaattorilla.

ShiftOrAnd:in aikavaativuus on sama kuin ShiftOr:in, $O(n\lceil \frac{m}{W} \rceil)$ (W tietokonesanan koko). Olkoon P hahmo, jossa on h' rinnakkaista ääntä. Palautuskaavassa (4) esitetyn tapauksen duaalitapauksessa etsitään koostetun hahmon $P = P_1 \cdots P_m$ esiintymää monofonisessa lähteessä $s = s_1 \cdots s_n$. Jos jollakin j pätee:

$$s_j \in P_1, \ldots, s_{j+m-1} \in P_m, \tag{5}$$

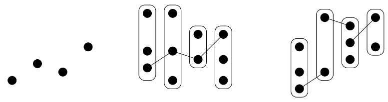
on koostetulla hahmolla P esiintymä kohdassa $s_j \cdots s_{j+m-1}$ [7] .

Edellisen tapauksen ratkaisun lisäksi Holub, Iliopoulos ja Mouchard esittivät myös, kuinka duaalitapaukset kombinoidaan etsimään koostetun hahmon hajautettuja esiintymiä polyfonisesta lähteestä [7]. Jos $\Sigma' \subseteq \Sigma$ esittää hahmossa esiintyvien merkkien joukkoa, HIM-algoritmi tarvitsee tämän ongelman ratkaisemiseksi ensin $O((h'm + |\Sigma'|)\lceil \frac{m}{W} \rceil)$ ajan esiprosessointiin, minkä jälkeen etsintä suoritetaan ajassa $O(nh\lceil \frac{m}{W} \rceil)$ ja tilassa $O(|\Sigma'|\lceil \frac{m}{W} \rceil)$. Artikkelissa esitetään myös, kuinka algoritmi on muokattavissa k-epätäsmäyksen tapaukseen (tämän tapuksen aika- ja tilavaativuus esitetään kirjoituksen lopussa taulukossa 1).

5.3 MonoPoly-algoritmi

MonoPoly-algoritmi [9] on ShiftOr-muunnelma hajautetun hahmon transpositioinvarianttien esiintymien löytämiseksi. Algoritmi toimii kolmessa vaiheessa, jotka ovat esiprosessointi-, suodatus- ja tarkistusvaihe.

[6]Koska Blochin ja Dannenbergin alkuperäinen sovellus oli erilainen, he eivät esittäneet hahmon esiintymiselle sopivaa ehtoa.



Kuva 3: Vasemmalla esitetyllä monofonisella hahmolla on hajautettu esiintymä keskellä esitetyssä lähteessä. MONOPOLY löytää kuitenkin mahdollisen esiintymän myös oikealla esitetyssä lähteessä.

Esiprosessoinnissa lähteestä $S = S_1 \cdots S_n$ muodostetaan bittivektorimerkkijono $\overline{S} = \overline{S}_1 \cdots \overline{S}_{n-1}$, missä kukin $\overline{S}_i$ koodaa sointujen S_i ja S_{i+1} väliset intervallit. Lähteen esiprosessointi kestää ajan $O(nh)$. Suodatusvaiheen alussa muodostetaan taulukko `T'`, joka on kuin SHIFTORin `T` taulukko, mutta siinä on sarake jokaista mahdollista $\overline{S}_i$:n arvoa kohti. Suodatusvaiheen lopussa esiintymäkandedaatteja etsitään SHIFTOR-algoritmilla. Suodatusvaiheen aikavaativuus on $O(n\lceil \frac{m}{W} \rceil + m + c)$ ja tilavaativuus $O(n+d)$, missä $c = |\Sigma| \cdot 2^{|\Sigma|}$ ja $d = 2|\Sigma| + 2^{|\Sigma|}$[7].

Suodatusvaihe saattaa löytää myös virheellisiä esiintymiä, koska bittivektoreihin $\overline{S}_i$ ei voida koodata tietoa siitä, mistä edellisen soinnun komponentista intervalli saavutettiin (kts. kuva 3). Tämän vuoksi suodatusvaiheen löytämät kandidaatit on vielä tarkastettava esimerkiksi luvun alussa esitetyllä naiivilla menetelmällä.

5.4 Doveyn algoritmi

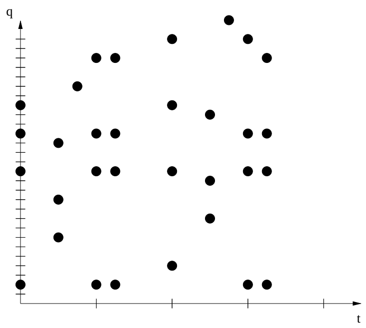
Musiikillisten koristeiden mahdollisesti aiheuttamien ongelmien vuoksi Dovey on kehitänyt dynaamisesta ohjelmoinnista t-hyppyesiintymät sallivan version [6]. Doveyn algoritmi toteuttaa ajassa $O(nm)$ palautuskaavaan (alustuksella $d_{00}, d_{0j}, d_{i0} = 0$):

$$d_{ij} = \begin{cases} d_{i,j-1} - 1, & \text{jos } ((p_i \notin S_j) \text{ ja} \\ & (d_{i,j-1} \neq 0)); \\ t+1, & \text{jos } ((p_i \in S_j) \text{ ja } (i=1 \\ & \text{tai } d_{i-1,j-1} \neq 0)); \\ 0, & \text{muuten,} \end{cases}$$

missä $1 \leq i \leq m$ ja $1 \leq j \leq n$. Palautuskaavan $\in$-operaatiot voidaan toteuttaa vakioajassa, koska soinnut S_i on toteutettu bittivektoreina (kuten edellisissä algoritmeissä).

Myös tässä tapauksessa haun tulokset luetaan taulukon (d_{ij}) alimmaiselta riviltä. Hahmolla on kohtaan j loppuva t-hyppyesiintymä, jos $d_{mj} = t + 1$. Algoritmi ei ole transpositioinvariantti; transponoitujen esiintymien löytämiseksi algoritmia iteroidaan hahmon eri transpooseilla.

[7] Suurten vakiotekijöiden vuoksi vektoreiden $\overline{S}_i$ arvoaluetta rajoitetaan, jotta algoritmi olisi käytännöllinen. Empiiristen kokeiden perusteella vektoreiden koko on 12 bittiä, joten taulukossa `T'` on 2^{12} saraketta.



Kuva 4: Musiikista muodostettu 2-ulotteinen pistejoukko.

5.5 SIA(M)ESE-algoritmi

Edellisistä poiketen SIA(M)ESE-algoritmi [12], joka perustuu musiikin analyysiin kehitettyihin SIA- ja SIATEC-algoritmeihin [11], käyttää geometrista lähestymistapaa. SIA(M)ESE pystyy ratkaisemaan kaikkien edellä esitettyjen algoritmien ratkaisemien ongelmien lisäksi myös määräämättömän pitkien hyppyesiintymien ongelman. Ajateltaessa asiaa editointietäisyyskäsitteen kannalta, algoritmi sallii haluttaessa myös määräämättömän mittaiset lisäykset, eli algoritmi löytää tarvittaessa myös hahmon alisekvenssien määräämättömän pitkät hyppyesiintymät. Näin hahmon esiintymiä voidaan hakea halutulla resoluutiolla: hahmosta voidaan poistaa epävarmat elementit.

Algoritmin syötteeksi (hahmoksi ja lähteeksi) käy mikä tahansa n-ulotteinen datajoukko. Tässä musiikkitapauksessa syöte voisi olla pareista $\langle q, t\rangle$ muodostuvia joukkoja (kuvassa 4 on tällä tavalla muodostettu pistejoukko kuvan 1 musiikkiesimerkistä). Olkoon $\overline{v} = (a, b)$ siirtovektori, joka siirtää esimerkiksi pisteen $\langle q, t\rangle$ pisteeseen $\overline{v}(\langle q, t\rangle) = \langle q+a, t+b\rangle$. SIA(M)ESE tutkii, löytyykö sellaisia siirtovektoreita $\overline{v}$, joilla hahmon elementit ovat siirrettävissä niin, että ne olisivat yhteneväisiä lähteen joidenkin elementtien kanssa. Tällaisilla siirtovektoreilla muodostetaan esiintymäjoukon $\mathcal{M}$ alkiot:

$$\mathcal{M} = \{ \langle \overline{v}, X\rangle \mid (x \in X) \Longleftrightarrow (x \in p) \wedge (\overline{v}(x) \in S)\}. \quad (6)$$

Hahmolla p on tarkka esiintymä lähteessä S, jos ja vain jos jollakin $\mathcal{M}$:n parilla $\langle \overline{v}, X\rangle$ pätee: $|X| = m$ $(X = p)$.

SIA(M)ESEn monipuolisuuden hintana on edellisiä suurempi aikavaativuus. Algoritmi toimii ajassa $O(n'm' \log(n'm'))$, missä n' ja m' esittävät tapahtumien (pisteiden) lukumäärää lähteessä ja hahmossa.

5.6 PROMSin algoritmi

Clausen, Engelbrecht, Meyer ja Schmitz ovat kehittäneet hajautettujen esiintymien etsintään PROMS-hakujärjestelmän [3]. PROMSin offline-hakumenetelmät perustuvat käänteisindeksiin (inverted index). PROMSissa kustakin lähteen tahdista muodostetaan indeksoitava "sormenjälki", jota edustaa pari (i, M). Tässä i kertoo dokumentin (kappaleen) ja M tahdin numeron. Sormenjälki on joukko kyseisen tahdin nuoteista muodostettuja pareja $\langle q, t\rangle$, jossa q määräytyy MIDI-korkeuden mukaan ja t on nuotin syttymishetki tahdin alusta laskien. Haussa etsitään hahmon elementtejä käänteisindeksistä yksitellen. Jos näiden yksittäishakujen leikkaus sisältää jonkin kappaleen i (tahdin M), hahmo esiintyy tässä kappaleessa (tahdissa).

PROMS sallii "ilmaiset poistot", mutta ei lisäyksiä. PROMS toteuttaa haut erittäin tehokkaasti alilineaarisessa ajassa. PROMSia voidaan käyttää myös likimääräisiin hakuihin käyttämällä koosteisia hahmoja (esiintymät määritellään kuten kaavassa (5)) tai sallimalla k epätäsmäystä. Molemmat likimääräiset tapaukset saadaan toteutetuksi hidastamalla algoritmin suoritusta vakiotekijän verran (vakiotekijähidastuksella voidaan etsiä myös hahmon eri tranpooseja).

PROMSin ongelmat liittyvät tarpeeseen rajoittaa käänteisindeksien kasvua: PROMS löytää vain hahmon esiintymät, jotka on tahditettu samalla lailla kuin hahmo. Vaikka lähteen voi olettaa useissa tapuksissa olevan (oikein) tahditettu, tämä on suuri rasite hahmolle. On erittäin epätodennäköistä, että WYHIWYG-käyttäjä osaisi tahdittaa antamansa hahmon oikein. Vaikka tahditus onnistuisikin, PROMS ei kykene löytämään hahmon kanssa eri tavalla tahditettuja esiintymiä, vaikka ne haun kannalta olisivatkin relevantteja.

5.7 DR-algoritmi

Doraisamyn ja Rügerin DR-algoritmi perustuu myös indeksointiin [5]. Algoritmissa käytetään q-grammeja likimääräisten hajautettujen transpositioinvarianttien esiintymien löytämiseksi. Kuten tekstimerkkijonojenkin tapauksessa, sallittaessa likimääräisyyttä hahmojen tulee olla pitkiä, jotta löydetään vain todellisia esiintymiä (eli saavutetaan riittävä erottelukyky). Menetelmän suurin ongelma liittyy kuitenkin indeksille tarvittavan muistin määrään. DR-algoritmissa indeksoidaan kaikki mahdolliset polyfonisessa lähteessä esiintyvät q-grammit. Vaikka polyfonisessa tapauksessa tällaisia q-grammeja on vähemmän kuin luvun alussa esitetyssä naiivissa indeksointitapauksessa, jossa indeksoitiin kaikki lähteen mahdolliset kombinaatiot (eli $(n - |q| + 1)(h^{|q|}) \leq h^n$, kun h > 1), isoilla musiikkitietokannoilla näidenkin indeksointi on käytännössä mahdotonta.

6 Yhteenveto

Musiikinhaku sisällön perusteella on nopeasti kasvava tutkimusala, jonka tuloksia esitellään mm. uudessa, kansainvälisessä vuosittain järjestettävässä konferenssissa (kts. http:\\www.music-ir.org). Hakuongelmaan, jossa tietokannan musiikki on pelkästään monofonista, löytyy kirjallisuudesta paljon ratkaisuja. Koska länsimainen musiikki on suurimmaksi osaksi polyfonista, ovat näiden ratkaisujen sovellusmahdollisuudet tulevaisuudessa turhan rajalliset.

Musiikinhaun perusteiden esittelyn lisäksi tässä kirjoituksessa on erityisesti perehdytty tapaukseen, jossa annettua hahmoa etsitään polyfonisista musiikkidokumenteista koostuvista tietokannoista. Taulukossa 1 (sivu 65) esitetään luodun katsauksen yhteenvetona kyseiseen ongelmaan kehitettyjen algoritmien ominaisuuksia. Vaikka kaikki nämä algoritmit pystyvät löytämään monofonisen hahmon tarkat, hajautetut esiintymät polyfonisesta tietokannasta, jokaisen algoritmin "optimaalinen ongelma" on sen verran toisista poikkeava, että yleispätevää paremmuusjärjestystä algoritmien välillä on mahdotonta asettaa.

Jos tehtävänä on hakea polyfonisen hakuavaimen esiintymiä polyfonisesta tietokannasta, vaihtoehtoja ovat BD-, Doveyn tai SIA(M)ESE-algoritmit. Etsittäessä monofonisen hahmon esiintymiä tietokannasta, jossa ei esiinny korukuvioita, nopeita algoritmeja ovat HIM ja MONOPOLY. Edellinen algoritmeista soveltuu likimääräisten ja jälkimmäinen transpositioinvarianttien esiintymien etsintään.

Mikäli tietokannassa esiintyy (kohtuullisesti) korukuvioita eikä tietokanta ole valtavan kokoinen, sopivia algoritmeja ovat PROMS, Doveyn algoritmi ja SIA(M)ESE. PROMS ei kuitenkaan kykene löytämään kaikkia hahmon esiintymiä. Doveyn algoritmi löytää kaikki esiintymät, jos korukuvioita tai muuta "kohinaa" ei ole liikaa. SIA(M)ESE löytää kaikki esiintymät, eikä transponoitujen esiintymien etsiminen aiheuta algoritmille lisätyötä.

Koska WYHIWYG-sovellukset ovat luonteeltaan interaktiivisia, käyttäjän tulee saada hakukoneelta palautetta ilman liian pitkää viivettä. Vaikka nykyisten tietokantojen läpikäyminen onnistuu lyhyessä ajassa myös nopeimmilla indeksointia käyttämättömillä algoritmeilla [8], tulevaisuudessa kaikkein laajimmat tietokannat tulevat vaatimaan indeksoinnin mahdollistamia alilineaaria suoritusaikoja. Liian suuren tilavaatimuksen takia aivan kaikkea ei voi indeksoida, minkä seurauksena kaikkia esiintymiä ei tällaisessa prosessissa pystytä löytämään. Paras tulos saatetaankin saavuttaa kombinoimalla eri menetelmiä. Esimerkiksi artikkelissa [10] esitetään ajatus 3 hakutasoa käyttävästä menetelmästä. Ensimmäinen taso indeksoi kaikki musiikillisesti merkittävimmät alisekvenssit, joten useimmin haetut hahmot löytyvät todennäköisesti tehokkaasti, alilineaarisessa ajassa. Jos käyttäjä haluaa/tarvitsee useampia esiintymiä, voi hän käynnistää toisessa vaiheessa (lähes) lineaarisessa ajassa toimivan MonoPoly-algoritmin. Tämän jälkeen käyttäjällä on halutessaan vielä mahdollisuus etsiä kaikki hahmon esiintymät Sia(m)ese-algoritmilla.

Viitteet

[1] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[2] J.J. Bloch and R.B. Dannenberg. Real-time accompaniment of polyphonic keyboard performance. In *Proceedings of the 1985 International Computer Music Conference*, pages 279–290, Vancouver, 1985.

[3] M. Clausen, D. Meyer R. Engelbrecht, and J. Schmitz. Proms: A web-based tool for searching in polyphonic music. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR'2000)*, Plymouth, MA, October 2000.

[4] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[5] S. Doraisamy and S.M. Rüger. An approach towards a polyphonic music retrieval system. In *the 2nd Annual International Symposium on Music Information Retrieval (ISMIR'2001)*, pages 187–193, Bloomington, IND, October 2001.

[6] M.J. Dovey. A technique for "regular expression" style searching in polyphonic music. In *the 2nd Annual International Symposium on Music Information Retrieval (ISMIR'2001)*, pages 179–185, Bloomington, IND, October 2001.

[7] J. Holub, C.S. Iliopoulos, and L. Mouchard. Distributed string matching using finite automata. *Journal of Automata, Languages and Combinatorics*, 6(2):191–204, 2001.

[8] K. Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Department of Computer Science, 2000. Report A-2000-4.

[9] K. Lemström and J. Tarhio. Detecting monophonic patterns within polyphonic sources. In *Content-Based Multimedia Information Access Conference Proceedings (RIAO'2000)*, pages 1261–1279, Paris, April 2000.

[10] K. Lemström, G.A. Wiggins, and D. Meredith. A three-layer approach for music retrieval in large databases. In *the 2nd Annual International Symposium on*

Music Information Retrieval (ISMIR'2001), pages 13–14, Bloomington, IND, October 2001.

[11] D. Meredith, K. Lemström, and G.A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*. (accepted).

[12] G.A. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. (submitted).

	BD [2]	PROMS [3]	DR [5]	Dovey [6]	HIM [7]	MONOPOLY [9]	SIA(M)ESE [12]
Hajautetun hahmon etsintä	x	x	x	x	x	x	x
Transpositio-invariantti	-	-	x	-	-	x	x
Polyfoniset hahmot	x	x	-	x	-	-	x
Sallii lisäykset	x k-eroa	-	x k-eroa	x*	-	-	x (tarvittaessa)
Sallii poistot	x	x ∞	x	x t-hypyt	-	-	x ∞
Sallii muutokset	x	x k-epät. tai ‡	x	-	x k-epät. ja ‡	-	x (lisäys + poisto)
oNline / oFfline	N	F	F	N	N	F	F
Perustuu indeksointiin	-	x	x	-	-	-	-
Aika-vaativuus	$O(m'n)$	$\ll O(n)$	?	$O(mn)$	$O((k+1)\ ns\lceil \frac{m}{w} \rceil)^{\star}$	$O(n\lceil \frac{m}{w} \rceil)^{\star,\diamond}$	$O(n'm' \log(n'm'))$
Tila-vaativuus	$O(m'h)$	?	?	$O(m)$	$O((k+1+ \lvert\Sigma'\rvert)\lceil \frac{m}{w} \rceil)$	$O(n)$	O(n'm'D)

*: Lisäominaisuus joka vaatii esitetyn palautuskaavan muokkaamista.
∞: Määräämättömän mittaiset.
‡: Koostettu hahmo.
*: Ilman esiprosessointia.
$^{\diamond}$: Parhaassa tapauksessa. Huonoimmassa tapauksessa tarkistusvaiheen aikavaativuus.
D: Dimensioiden lukumäärä.
w: Tietokonesanan koko bitteinä.
$m' \leq mh, n' \leq nh$.

Taulukko 1: Polyfonisen musiikin hakualgoritmien ominaisuudet pähkinänkuoressa. Huomaa, että algoritmi ei ole transpositioinvariantti, jos se ei kykene löytämään hahmon transponoituja versioita ilman ylimääräisiä iteraatioita.