hyväksymispäivä arvosana

arvostelija

# Symboliset musiikkihakumenetelmät

Mika Turkia

Helsinki 17. maaliskuuta 2003
Luonnontieteen kandidaatin tutkielma

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Symboliset musiikkihakumenetelmät

Mika Turkia

Luonnontieteen kandidaatin tutkielma

Tietojenkäsittelytieteen laitos

Helsingin yliopisto

17. maaliskuuta 2003, 21 sivua

Työssä esitetään yleiskatsaus symbolista musiikkitietoa käyttäviin musiikkihakualgoritmeihin. Perusongelmana on melodiaa kuvaavan hakuhahmon etsintä valitussa muodossa esitetyistä musiikkikappaleista. Perinteisesti algoritmit ovat pohjautuneet merkkijonomenetelmiin, jotka on tarkoitettu yksiulotteisen tekstihahmon etsintään yksiulotteisesta tekstiaineistosta. Ne toimivat hyvin haettaessa yksiäänistä melodiaa yksiäänisestä aineistosta, mutta suurin osa musiikista on moniäänistä. Merkkijonomenetelmiin perustuvia moniäänisen aineiston käsittelyyn kykeneviä algoritmeja ovat mm. ShiftOrAnd, IntervalMatching ja MonoPoly. Moniäänisyyden hallinta merkkijonopohjaisilla tekniikoilla on kuitenkin vaikeaa, koska moniääninen musiikkitieto on luontevimmin kuvattavissa moniulotteiseksi. Geometrisella lähestymistavalla useat ongelmat ratkeavat triviaalisti. Geometrisia algoritmeja ovat mm. SIA(M)E, P1, P2 ja P3. Työssä esitellään myös prototyyppiohjelmisto, johon edellä mainittuja algoritmeja on toteutettu.

Aiheluokat(Computing Reviews 1998): G.2.1, H.3.3, H.5.5

Avainsanat: musiikkihaku, musiikkialgoritmit, merkkijonomenetelmät, dynaaminen ohjelmointi, bittirinnakkaiset algoritmit, geometriset algoritmit, MIDI

# Sisältö

# 1 Johdanto

Hakujen tekemiseen tekstitietokannoista on 1990-luvun aikana rakennettu useita tehokkaita julkisia hakukoneita, kuten esimerkiksi Google [BrP98]. Laadullisesti vastaavia järjestelmiä melodiaperusteisten hakujen tekemiseen musiikkitietokannoista ei ole ollut. Tarvetta sellaisille kuitenkin on. Musiikkikokoelmat ja musiikin kulutus ovat jatkuvasti kasvaneet. Käsin tehtävät haut musiikkitietokannoista ovat hitaita, rajoitettuja, ja vaativat laajaa musiikin tuntemusta. Tässä tutkielmassa luodaan yleiskatsaus uusiin algoritmeihin [LeT03, ULM03], joita voidaan käyttää tällaisia järjestelmiä rakennettaessa.

Musiikin digitaaliset esitysmuodot voidaan jakaa symbolisiin ja ei-symbolisiin [Dan93]. Ei-symbolisissa esitysmuodoissa kuvataan äänenpaineen tasoja ajan suhteen (audiosignaali), kun taas symbolisissa esitysmuodoissa talletetaan äänenpaineen tasojen kuvaus korkeammalla abstraktiotasolla, esimerkiksi nuotteina.

Symbolisessa lähestymistavassa musiikkikappale esitetään yleensä nuoteista koostuvina tietorakenteina. Algoritmisena perusongelmana on musiikillisen melodian kuvaavan hakuhahmon esiintymien etsintä näistä tietorakenteista. Ei-symbolisen lähestymistavan perusongelmia ovat muun muassa luokittelu esimerkiksi musiikillisen tyylin mukaan [TzE01] ja aaltomuotoisen moniäänisen hahmon etsintä aaltomuotoisesta moniäänisestä lähteestä [CaB02]. Jälkimmäisessä tapauksessa nykyiset hakumenetelmät tunnistavat ainoastaan täsmälleen saman esityksen, joten esimerkiksi monofonisen hahmon tunnistaminen polyfonisesta musiikista ei onnistu. Symbolisia menetelmiä siis tarvitaan.

Vaikka viime aikoina erillisenkin melodian etsinnässä polyfonisesta aaltomuotoisesta tiedosta on edistytty [ShS02], tehokkuus- ja tilasyistä aaltomuotoisesta äänestä kannattaa silti muodostaa symbolinen kanta. Symbolisten algoritmien merkitys tulee siis jatkossakin olemaan huomattava.

Polyfonisia symbolisia algoritmeja on tutkittu laajemmin 1990-luvun lopulta lähtien. Alalla on oma, vuonna 2000 perustettu konferenssi ISMIR (International Conference on Music Information Retrieval) [WWW02].

Tässä tutkielmassa käsitellään moniäänisen symbolisen musiikkiaineiston käsittelyyn kykeneviä hakualgoritmeja. Luvussa 2 luodaan katsaus musiikin symbolisen esittämisen ongelmakohtiin. Luvussa 3 käsitellään itse algoritmeja. Luvussa 4 esitellään prototyyppijärjestelmää, johon algoritmeja on toteutettu [WWW03]. Luku 5 tarjoaa lyhyen yhteenvedon käsitellyistä aiheista.

# 2 Musiikkitiedon symbolinen esittäminen

Musiikkia voidaan käsitteellistää useilla erilaisilla kuvaustasoilla, joita ei voida yksikäsitteisesti muuntaa toisikseen. Tällaisia kuvaustasoja edustavat esimerkiksi kuvaukset ääniaaltoina, esityksenä tai nuottikuvana [Dan93]. Näistä ääniaaltoesitys voidaan ajatella ei-symbolisena, nuottikuva symbolisena, ja esitystaso näiden välisenä, "puolisymbolisena" kuvaustasona [Dan93]. Tässä luvussa käsitellään eri esitysmuotoja ja attribuuttien valintaa.

## 2.1 Musiikkitiedon rakenteellisen kuvaamisen ongelmia

Länsimaisen nuottikirjoituksen on sanottu olevan monimutkaisimpia ihmisen muodostamista graafisista kuvauksista, minkä vuoksi sen rakenteiden esittäminen koneelliseen käsittelyyn sopivassa eksplisiittisessä yksikäsitteisessä muodossa on osoittautunut vaikeaksi. Monet nuottikirjoituksen elementtien väliset suhteet ja attribuutit ovat kuvattavissa matemaattisesti [Dan93]. Musiikkiesityksiin sisältyy tulkinnallisia komponentteja, jotka eivät ole johdettavissa nuottikuvasta, ja joista osa selittyy vain tietyn tulkintaperinteen tai tyylisuunnan sisällä. Toisaalta nuottikuvaan voi sisältyä elementtejä, jotka eivät välttämättä ole havaittavissa kuulonvaraisesti [Dan93].

Esitystapoja voidaan luokitella muun muassa sen mukaan, millaisia nuottien välisiä relaatioita niiden avulla voidaan määritellä [Dan93]. Esimerkki tällaisesta suhteesta on kuuluminen samaan sointuun, joka yleensä määritellään alkuaikojen samanhetkisyytenä. Tämä relaatio voidaan esittää useimmissa esitystavoissa. Sen sijaan esimerkiksi kuulumista samaan tahtiin ei voida esittää yleisessä MIDI-esitystavassa (luku 2.2.1). Attribuuttien ja relaatioiden määrittely on osin mielivaltaista. Nuotille voidaan esimerkiksi antaa tahtinumeroattribuutti, jonka pohjalta määrittyy implisiittisesti samaan tahtiin kuulumisen suhde. Toisaalta nuotit voidaan ryhmitellä tahdeittain, jolloin kyseessä on eksplisiittinen relaatio. Valintojen paremmuus riippuu sovelluskohteesta.

Musiikkihakualgoritmeja suunniteltaessa suuri osa nuotinnustiedon esittämiseen liittyvästä moniselitteisyydestä voidaan sivuuttaa. Kymmenien tuhansien kappaleiden käsittelyyn sopivaa järjestelmää suunniteltaessa valittava attribuutti- ja relaatiojoukko on tilan säästämiseksi minimoitava. Joukon valinta on algoritmikohtaista: useille algoritmeille riittää nuottien järjestyksen ja sävelkorkeuksien kuvaaminen. Haluttaessa hakea rytmin tai kestojen perusteella tarvitaan myös nuottien alkuaika- tai aikaintervallitiedot ja kestotiedot. Erityyppisillä algoritmeilla on erilaisista lähestymistavoista johtuvia tarpeita: esimerkiksi MonoPoly-algoritmi (luku 3.2.4) on suunniteltu käyttämään soinnun käsitettä, joten toteutuksen yksinkertaistamiseksi esitystavan on hyvä eksplisiittisesti tukea sitä, kun taas geometrisille algoritmeille (luku 3.3) se on tarpeeton [1]. Käy-

[1]Sointurakennetta käyttämällä soinnun alkuaikaa ei tarvitse toistaa, jolloin saavutetaan polyfonian asteeseen suoraan

tännössä monta eri algoritmia sisältävässä hakujärjestelmässä attribuuttijoukoksi on siis valittava kaikkien sisällytettävien algoritmien tarvitsemien attribuuttien ja relaatioiden yhdiste.

## 2.2 Eri esitysmuotoja

Lähinnä nuotinnus- tai äänisynteesiohjelmistojen tarpeisiin on kehitetty kymmeniä symbolisia esitysmuotoja [WWW00]. XML-pohjaisia esitysmuotoja on listattu yli 20, samoin kuin ASCII-pohjaisia.

Binäärimuotoisia esitysmuotoja on kehitetty yli 10. Niistä tärkein on MIDI. Muut, kuten NIFF (Notation Interchange File Format), eivät ole vakiintuneet laajaan käyttöön.

Yleisesti hyväksyttyä standardia musiikin symboliseen esittämiseen ei ole saatu muodostettua [HRG01]. Julkaistuja yleisesityksiä joihinkin symbolisiin musiikkitiedostomuotoihin on saatavana [HeS01, Sel97].

### 2.2.1 MIDI

Yleisin ja eräs varhaisimmista digitaalisista symbolisista esitystavoista on soitinten väliseksi kommunikaatioprotokollaksi soitinvalmistajien toimesta 1980-luvun alussa määritelty MIDI-protokolla (Musical Instrument Digital Interface). Protokolla käsittelee ajastettujen tapahtumien siirtoa. Sen laajennus SMF (Standard MIDI File) on tarkoitettu näiden tapahtumien tallentamiseen tiedostoon [Sel97, Lem00].

SMF-muodossa on saatavilla suuri määrä musiikkia, mutta SMF on huonosti laajennettava ja rajoittunut [HRG01, Dan93]. Rakenteellisen informaation puutteen vuoksi nuottikuvan muodostaminen MIDI-tiedon pohjalta on mahdollista hyvin rajoitetusti: esimerkiksi tahtiviivojen (ja tämän seurauksena kaarien) sijoittelu on mahdollista vain likimääräisesti, eikä enharmonisten nuottien (esimerkiksi cis ja des) erottaminen ole mahdollista [HRG01].

Hakualgoritmien kannalta nuottikuvan muodostusvaikeudet eivät ole ongelma [2] , mutta MIDI-muotoon liittyy niidenkin kannalta olennaisia puutteita. Koska suuri osa SMF-muotoisesta musiikista on soitettua ja MIDI-protokolla mahdollistaa länsimaisesta nuottikirjoituksesta poikkeavien nuottikestojen esittämisen, kestoissa esiintyy vaihtelua, joka pitäisi kestojen tai rytmin mukaan haettaessa erikseen diskretoida [HRG01].

verrannollinen tilansäästö.

[2] Nuotinnustieto kuitenkin auttaa tulosten havainnollisessa esittämisessä, minkä vuoksi sen sisällyttäminen järjestelmiin on hyödyllistä.

### 2.2.2 ASCII- ja XML-pohjaisia esitysmuotoja

Nuotinnusohjelmia varten suunniteltuja ASCII-pohjaisia esitysmuotoja ovat mm. GUIDO [HRG01] ja LilyPond [WWW01]. Äänisynteesiohjelmistoja varten suunniteltuja muotoja ovat mm. Csound [Bou00] ja MPEG-4 [Scr00]. XML-pohjaisia esitysmuotoja ovat mm. NIFFML, MDL (Music Description Language) ja MusicXML [HeS01].

Mikäli musiikkiaineistoa olisi riittävästi saatavilla jossakin MIDI-muotoa joustavammassa muodossa, hakujärjestelmän kannattaisi käyttää sitä [HRG01]. XML-pohjaisten muotojen suuret tilavaatimukset vähentävät niiden käytettävyyttä. Tiiviimpiä ja luettavampia muotoja ovat esimerkiksi GUIDO ja LilyPond.

# 3 Hakualgoritmeja

## 3.1 Yhteisiä käsitteitä

### 3.1.1 Musiikillista käsitteistöä

*Nuotti* voidaan esittää monikkona, jonka komponentit ilmaisevat nuotin parametreja kuten korkeutta ja kestoa. Nuotit nimetään suomalaisen käytännön mukaan seuraavasti: c, cis/des, d, dis/es, e, f, fis/ges, g, gis/as, a, ais/b, h.

*Sävellaji* voidaan perinteisessä länsimaisessa taidemusiikissa määritellä tietystä nuotista alkavana 8 nuotin pituisena sarjana, jossa nuotinkorkeuksien väliset intervallit ovat vakioidut. Esimerkiksi duurisävellajeissa ne ovat 2, 2, 1, 2, 2, 2 ja 1 puolisävelaskelta, ja luonnollisissa mollisävellajeissa 2, 1, 2, 2, 1, 2 ja 2 puolisävelaskelta [Sal97]. MIDI-järjestelmässä (luku 2.2.1) nuotin korkeudet esitetään välillä 0-127 [Sel97]. Yksi puolisävelaskel vastaa yhtä MIDI-muodon nuotinkorkeusyksikköä. Ns. keski-C:n arvoksi määritetään yleensä 60.

*Transpositioinvarianssi* tarkoittaa sävellajiriippumattomuutta, joka saavutetaan vertaamalla haettavan hahmon ja lähdeaineiston nuottien korkeuksien sijaan niiden korkeuksien välisiä erotuksia (intervalleja) [Lem00]. Transpositioinvarianssin haluttavuus perustuu siihen, että länsimainen ihminen yleensä havaitsee eri korkeuksilla soitetut saman intervallijonon sisältävät esitykset samoiksi; esimerkiksi nuottijono “c d” samaistetaan nuottijonoon “d e”.

*Oktaaviekvivalenssi* tarkoittaa nuottien oktaavikerrannaisten samaistamista. Nuottien korkeuksien sijaan tarkastellaan ns. nuottiluokkia, jotka ilmaisevat nuotin nimen ilman oktaavimäärettä [Lem00]. Käytännössä nuottiluokka saadaan ottamalla nuotinkorkeuden MIDI-numeroarvon jakojäännös luvun 12 suhteen. Melodiahaun kannalta oktaaviekvivalenssilla ei ole musiikillista merkitystä kuin erityistarkoituksissa. Tavoitteena on mahdollistaa kaikkien mahdollisten intervallien esittäminen kahdella tavulla bittirinnakkaisissa algoritmeissa.

*Polyfonia* tarkoittaa moniäänisyyttä, joka yleensä ymmärretään nuottien sointiaikojen samanaikaisuudeksi. Vastaavasti *monofonia* tarkoittaa yksiäänisyyttä.

*Sointu* määritellään joukoksi nuotteja, joilla on sama alkuaika. Sointu voi siis olla monofoninen tai polyfoninen.

*Sanakirjajärjestys* eli ns. *leksikaalinen* järjestys määritellään näiden algoritmien yhteydessä järjestykseksi, jossa nuotit ovat ensisijaisesti alkuaikojensa ja toissijaisesti nuotinkorkeuksiensa suhteen nousevassa järjestyksessä. Toisin sanoen soinnut ovat alkuaikojensa mukaisessa järjestyksessä, ja kunkin soinnun nuotit ovat korkeutensa suhteen nousevassa järjestyksessä.

### 3.1.2 Online- ja offline-menetelmät

Hakumenetelmiä voidaan jaotella lähdeaineiston esitysmuodon mukaan *online-* ja *offline*-menetelmiin. Online-menetelmissä haku kohdistetaan suoraan alkuperäisen muotoiseen lähdeaineistoon. Offline-menetelmissä lähdemuotoisesta aineistosta muodostetaan haun kannalta tarpeettomista parametreista puhdistettu esitysmuoto, johon haku kohdistetaan (vrt. luku 2.1). Offline-menetelmiä käytettäessä lähdeaineiston on oltava kokonaisuudessaan saatavana ennen haun käynnistämistä; online-menetelmillä taas voidaan käsitellä jatkuvaa tietovirtaa.

Jatkossa online- ja offline-termejä käytetään kuitenkin vakiintuneen käytännön [Lem00] mukaisesti hieman eri merkityksessä siten, että online-menetelmiksi kutsutaan niitä menetelmiä, jotka käyttävät SMF-muodosta muodostettua erillistä esitysmuotoa, jota jatkossa kutsutaan kannaksi. Offline-menetelmiksi kutsutaan niitä menetelmiä, jotka käyttävät jotakin kannasta edelleen muodostettua algoritmikohtaista tietorakennetta.

### 3.1.3 Suodatus- ja tarkistusmenetelmät

Toinen tapa jaotella hakumenetelmiä on jakaa ne tulosten tyypin mukaan *suodatusmenetelmiin* ja *tarkistusmenetelmiin* [Lem00]. Osumassa on oltava hahmon määräämät intervallit, ja lisäksi peräkkäisten intervallien laskennassa käytetyn yhteisen nuotin on oltava sama. Kuvallisesti ajateltuna nuotinkorkeuksien alkuaika–nuotinkorkeus-avaruudessa muodostaman murtoviivan on oltava jatkuva.

Suodatusmenetelmät löytävät (esimerkiksi oktaaviekvivalenssin käytön vuoksi) vain ensimmäisen ehdon täyttäviä osumakandidaatteja. Tarkistusmenetelmät löytävät molemmat ehdot täyttäviä osumia, ja niitä käytetään myös suodatusmenetelmien tuottamien osumakandidaattien tarkistamiseen.

## 3.2 Merkkijonomenetelmiin perustuvat algoritmit

Suurin osa musiikinhakualgoritmeista on perustunut merkkijonomenetelmiin [Lem00]. Näistä tärkeimpiä ovat olleet *dynaamisen ohjelmoinnin* algoritmit, joiden erikoistapauksia ovat ns. *bittirinnakkaiset* algoritmit. Niissä pyrkimyksenä on käsitellä esimerkiksi kokonainen sointu yhdellä tai muutamalla prosessorin bitti- ja sanatason operaatiolla.

Kahden merkkijonon samankaltaisuutta mitataan yleensä *editointietäisyyden* avulla [Lem00]. Sen määrittämiseksi on ensin valittava sallittujen *muunnosoperaatioiden* joukko ja kustannuksen jokaiseen muunnosoperaatioon liittävä positiivisarvoinen *kustannusfunktio*. Esimerkiksi ns. *Levensh-*

*tein*-etäisyyden tapauksessa muunnosoperaatiot ovat merkin korvaaminen toisella, merkin poisto ja merkin lisäys, joiden kunkin kustannus on yksi. Esimerkiksi merkkijonon "cab" jonoksi "cde " muuntamisen kustannus on kaksi (kaksi korvausoperaatiota) ja jonon "ce" jonoksi "cde " muuntamisen kustannus yksi (yksi lisäysoperaatio) [CrR94].

Editointietäisyys voidaan laskea dynaamisen ohjelmoinnin avulla taulukoimalla toinen merkkijono pystyakselille ja toinen vaaka-akselille ns. etäisyystaulukoksi. Merkitään pystyakselin merkkijonon pituutta muuttujalla $M$ ja vaaka-akselin merkkijonon pituutta muuttujalla $N$. Taulukon alkioiden arvot lasketaan Levenshtein-etäisyyden tapauksessa seuraavalla kaavalla:

$$d_{0,0} = 0$$

$$d_{i,0} = i$$

$$d_{0,j} = j$$

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + 1, \\ d_{i,j-1} + 1, \\ d_{i-1,j-1} + (if\ a_i = b_j\ then\ 0\ else\ 1) \end{cases}$$

Kukin taulukon alkio $d_{i,j}$ ilmaisee $i$- ja $j$-koordinaattien rajaamien osajonojen editointietäisyyden. Koko merkkijonojen editointietäisyys on alkion $d_{M,N}$ arvo.

Vastaavalla periaatteella voidaan tehdä merkkijonon etsintää. Ylimmän rivin arvot alustetaan nolliksi ($d_{0,j} = 0$). Muut arvot lasketaan samoin kuin yllä. Alimman rivin alkioiden arvot ilmaisevat virheiden määrän, joten tarkkojen osumien loppukohdissa alimman rivin alkioiden arvo on nolla.

Jos osumasta riittää kertoa sen loppukohta, koko taulukkoa ei tarvitse varata. Tällöin riittää tallettaa vain yksi pystysarake, joka siirtyy oikealle siirryttäessä seuraavaan merkkiin.

Ns. *likimääräiset* hakualgoritmit perustuvat usein yllä esitettyyn menetelmään. Ne voivat sallia esimerkiksi hahmon "cde " muuntumisen muotoon "cdf ". Jos likimääräistä hakua käytettäessä halutaan selvittää myös osuman alkukohta, on talletettava koko taulukko, koska etsintä tehdään peruuttamalla taulukossa osuman loppukohdasta alkaen (ns. backtracking).

### 3.2.1 Naivi algoritmi

Merkitään lähteen pituutta muuttujalla $N$ ja hahmon pituutta muuttujalla $M$. Naivi algoritmi ei käytä hyväkseen bittirinnakkaisuutta vaan toimii niin sanotulla brute force-menetelmällä tarkastaen kaikki $N - M + 1$ mahdollista osuman alkukohtaa niin pitkälle kuin hahmo vastaa lähteessä esiintyviä merkkejä [LeT03, Lem00]. Algoritmin keskimääräinen aikavaatimus on $O(N)$ ja pahimman tapauksen aikavaatimus on $O(MN)$. Naivi algoritmi voidaan toteuttaa joko eksaktiksi tai transpositioinvariantiksi.

Edellä mainituissa lähteissä esitetyt algoritmit hakevat monofonista hahmoa polyfonisesta aineistosta transpositioinvariantisti. Listauksessa 1 esitetään muunnos, joka hakee polyfonista sanakirjajärjestyksessä olevaa hahmoa polyfonisesta sanakirjajärjestyksessä olevasta lähteestä ei-transpositioinvariantisti.

Kaksiulotteista lähdetaulukkoa on merkitty muuttujalla $source$, pääsilmukan sointuindeksiä muuttujalla $i$, väliaikaista sointuindeksiä muuttujalla $ci$, ja soinnun nuotti-indeksiä muuttujalla $ni$. Attribuutti $chords$ ilmaisee lähdetaulukon sointujen määrän. Soinnulla on alkuaikakenttä $start$ ja soinnun pituus -kenttä $length$. Nuotilla on nuotinkorkeuskenttä $pitch$.

Yksiulotteista hahmovektoria merkitään muuttujalla $pattern$ ja sen nuotti-indeksiä muuttujalla $pi$, Attribuutti $chords$ ilmaisee hahmon sointujen määrän ja attribuutti $notes$ nuottien määrän. Hahmon nuotit ovat $(start, pitch)$ -pareja. Samaan hahmon sointuun kuuluvilla nuoteilla on sama alkuaika.

Rivin 1 silmukka käy läpi kaikki mahdolliset osuman alkukohdat. Rivin 5 silmukka tarkistaa löytyvätkö hahmon nuotit lähteen soinnuista. Jos lähteen soinnun nuotin korkeus on suurempi kuin hahmon nuotin, sanakirjajärjestyksen takia haettavaa nuottia ei voi löytyä koko soinnusta.

Jos soinnun nuotin korkeus on pienempi, on käsiteltävä kaksi tapausta. Jos ollaan lähteen soinnun lopussa, haettavaa nuottia ei löytynyt, joten tarkistus epäonnistui. Muuten siirrytään lähteen soinnun seuraavaan nuottiin.

Jos soinnun ja hahmon nuotin korkeudet ovat yhtä suuret, on käsiteltävä neljä tapausta. Jos oltiin hahmon viimeisessä nuotissa, tarkistus onnistui. Jos oltiin hahmon soinnun lopussa, siirrytään hahmon seuraavaan nuottiin ja lähteen seuraavaan nuottiin. Jos oltiin lähteen soinnun lopussa mutta ei hahmon soinnun lopussa, tarkistus epäonnistui, koska jotakin hahmon soinnun nuottia ei löytynyt lähteen soinnusta. Muussa tapauksessa siirrytään eteenpäin sekä hahmon että lähteen soinnuissa.

**Listaus 1.** Sanakirjajärjestystä hyödyntävä polyfoninen hakualgoritmi.

```
1.  for (i = 0; i < source.chords - pattern.chords + 1; i++)
2.    pi = 0;
3.    ni = 0;
4.    ci = i;
5.    while (pi < pattern.notes)
6.      if (source[ci][ni].pitch > pattern[pi].pitch) break;
```

```
    elsif (source[ci][ni].pitch < pattern[pi].pitch)
      if (++ni == source[ci].length) break;
    else
       if (pi == pattern.notes - 1)
        output(i);
      elsif (pattern[pi].start != pattern[pi + 1].start)
        pi++;
        ci++;
        ni = 0;
      else
        if (++ni == source[ci].length) break;
        pi++;
      end
    end
  end
end
```

Poistamalla uloimmainen silmukka algoritmia voidaan käyttää tarkistusalgoritmina MonoPoly-algoritmin (luku 3.2.4) löytämille osumakandidaateille.

Transpositioinvarianssin toteuttaminen vaatii yksittäisten sointujen nuotinkorkeuksien tarkastelun sijaan peräkkäisten sointujen nuottien välisten intervallien tarkastelua.

### 3.2.2 ShiftOrAnd

ShiftOrAnd on ei-transpositioinvariantti online-tarkistusalgoritmi [LeT03]. Se on monofonisesta lähdeaineistosta tapahtuviin hakuihin tarkoitetun dynaamiseen ohjelmointiin perustuvan Shift-Or-algoritmin [BaG92] muunnelma. Periaatteena on simuloida aiemmin esitettyä etäisyystaulukointia siten, että tarkkojen osumien haussa vaadittava yksi taulukon rivi esitetään yhdellä sanalla. Tämän vuoksi hahmon pituus on rajoitettu sanan pituuden suuruiseksi.

ShiftOr-algoritmissa sanan bitit alustetaan ykkösiksi. Lisäksi on muodostettava aakkoston koon suuruinen aputaulukko, jonka alkioita ovat sanat. Kukin aputaulukon sana vastaa aakkoston merkkiä. Sanan bitit kuvaavat hahmon paikkoja. Nollabitti ilmaisee, että hahmon paikassa esiintyy kyseistä aputaulukon sanaa vastaava aakkoston merkki. Vastaavasti ykkösbitti ilmaisee, että hahmon paikassa ei esiinny kyseistä merkkiä.

Lähdettä läpikäytäessä sanan bittejä siirretään vasemmalle yhden askeleen verran jokaisen merkin kohdalla. Tällöin vähiten merkitseväksi bitiksi tulee nolla. Lisäksi siirretylle sanalle tehdään bittitason or-operaatio lähteen merkkiä vastaavan aputaulukon alkion kanssa. Jos siis lähteen merkki ja hahmon merkki ovat samat, bittivektorin siirrosta aiheutunut nolla säilyy ja siirtyy seuraavalla kierroksella edelleen vasemmalle. Muussa tapauksessa sen arvoksi tulee yksi. Jos nolla pääsee siirtymään vasemmalle hahmon pituuden verran, on löydetty osuma.

ShiftOrAnd-algoritmissa or-operaation tekijänä on sana, jossa niiden bittien joiden indeksiä vastaavissa hahmon kohdissa esiintyy käsiteltävänä olevassa soinnussa esiintyvä merkki arvona on nolla. Muiden bittien arvona on yksi. Jos siis soinnussa esiintyy hahmon nuotti, bittivektorin nolla säilyy.

Lähde käydään läpi vain kerran, joten algoritmin aikavaatimus on $O(N)$.

### 3.2.3 IntervalMatching

IntervalMatching on bittirinnakkainen online-suodatusalgoritmi [LeT03]. Se löytää transpositioinvariantteja ja oktaaviekvivalentteja osumakandidaatteja.

Transpositioinvarianssi edellyttää intervallien laskentaa. IntervalMatching-algoritmi laskee jokaiselle soinnulle sen ja seuraavan soinnun nuottien väliset intervallit, ja vertaa intervalleja sillä hetkellä käsiteltävän hahmon merkkiparin intervalliin. Transpositioinvarianssin ja oktaaviekvivalenssin vuoksi mahdollisia intervalleja on 12, ja sointuparin intervallien tilavaatimus on siten 12 bittiä. Varsinainen täsmäys tehdään ShiftOr-algoritmilla.

### 3.2.4 MonoPoly

MonoPoly [LeT03, Lem00] on bittirinnakkainen transpositioinvariantteja ja oktaaviekvivalentteja osumakandidaatteja löytävä algoritmi kuten IntervalMatching, mutta se on ns. offline-algoritmi. Sen käyttämä tietorakenne on sekvenssi, jossa kunkin sointuparin väliset oktaaviekvivalentit intervallit on laskettu etukäteen ja talletettu kahteen tavuun.

Offline-tietorakenteen muodostamisen aikavaatimus on $O(Nq)$, jossa $q$ on soinnun maksimikoko. Tässä ns. esikäsittelyvaiheessa lasketaan intervallit jokaisen soinnun alimman nuotin ja seuraavan soinnun kaikkien nuottien välillä. Intervallit soinnun muiden nuottien kanssa saadaan siirtämällä laskettuja intervalleja soinnun alimman nuotin ja verrattavan nuotin välisen erotuksen verran. Esikäsittely täytyy tehdä vain kerran, jos saatu tietorakenne tallennetaan esimerkiksi levylle.

Alustusvaiheessa hahmon tiedoista muodostetaan aputaulukko lineaarisessa ajassa. Aputaulukon rakentamisessa käytetään kahta ylimääräistä aputaulukkoa. ShiftOrAnd-algoritmin aputaulukossa oli oma sarake kaikille mahdollisille nuottikorkeuksille. MonoPoly-algoritmin aputaulukossa on sarake jokaiselle mahdolliselle intervalliyhdistelmälle. Kuten IntervalMatching-algoritmilla, varsinainen täsmäys tehdään ShiftOr-algoritmilla.

## 3.3 Geometriset algoritmit

Merkkijonomenetelmiin perustuvien algoritmien ongelmana on, että polyfoninen musiikkitieto ei ole luonteeltaan sarjallista vaan rinnakkaista. Samanaikaisesti esiintyvien nuottien ongelmaa ei voi luontevasti ratkaista merkkijonomenetelmäpohjaisesti.

Toinen ongelma on ylimääräisten nuottien tuottamat katkot melodiassa. Muun muassa ns. *koristenuottien* esiintyminen (grace notes) aiheuttaa sen, että edellä esitetyt merkkijonopohjaiset algoritmit eivät tunnista melodiaa, koska ne edellyttävät hahmon nuottien löytyvän lähteestä peräkkäin. On mahdollista toteuttaa merkkijonopohjaisia algoritmeja, joille koristenuotit eivät ole ongelma, mutta mikäli myös polyfonia ja transpositioinvarianssi halutaan mukaan, ongelmaa ei voida ratkaista kovin tehokkaasti.

Geometrisesti musiikkitieto on esitettävissä täysin luontevasti niin, että molemmat ongelmat ratkeavat triviaalisti. Nuottien alkuajat voidaan esittää pisteinä aika–nuotinkorkeus-avaruudessa. Jos halutaan esittää myös kestot, nuotti kuvautuu janaksi samaan avaruuteen. Muita parametreja voidaan lisätä kasvattamalla avaruuden dimensiota. Algoritmien toteuttaminen käyttämään polyfonisia hahmoja ei ole ongelma. Alkuaikoja ja nuotinkorkeuksia ei tarvitse diskretisoida eikä muutenkaan rajoittaa. Lähteen on kuitenkin oltava järjestetty sanakirjajärjestykseen.

Yleisenä periaatteena on laskea lähteen ja hahmon jokaisen nuotin välinen erotusvektori (siirtovektori), jonka x-komponentti ilmaisee aikasiirtymän kappaleen alusta ja y-komponentti transposition määrän. Usea lähteen nuotti voi kuvautua samalle siirtovektorille. Jos yhdelle vektorille kuvautuvia nuotteja on hahmon pituuden verran, hahmo voidaan kokonaisuudessaan kuvata lähteeseen ko. vektorilla, eli on löytynyt osuma. Vastaavasti löytyvät osittaiset osumat. Algoritmit eroavat toisistaan sen suhteen, missä järjestyksessä vektorit lasketaan, miten ne talletetaan, ja etsitäänkö vain täysiä vai vain osittaisia osumia.

Geometristen algoritmien käyttö ei ole rajattu musiikkitietoon, vaan ne soveltuvat etsintään missä tahansa monidimensionaalisissa datajoukoissa.

### 3.3.1 SIA ja SIATEC

SIA-perheen algoritmit [MWL01] eivät ole varsinaisia hakualgoritmeja, vaan ne on tarkoitettu toistuvien hahmojen etsintään. Niitä voidaan käyttää musiikillisessa analyysissa. Ne tuottavat suuren määrän toistuvia hahmoja. Ongelmana on musiikillisesti merkityksellisten hahmojen erottaminen ei-merkityksellisistä hahmoista.

SIA (Structure Induction Algorithm) etsii annetun musiikkikappaleen kaikki maksimaaliset transpositioinvariantit toistuvat hahmot. Aikavaatimus on $O(kN^2 \log N)$, jossa $k$ on avaruuden dimensio. SIATEC (Structure Induction Algorithm with Translational Equivalence Classes) ottaa syötteekseen SIA-algoritmin tulosteen ja etsii maksimaalisten toistuvien hahmojen esiintymät. Aikavaatimus on $O(kN^3)$.

SIATEC-algoritmin tulosteen osajoukosta voidaan muodostaa loppuosapuu, josta haut voidaan tehdä hahmon pituuden suhteen lineaarisessa ajassa. Osajoukon valinnassa voidaan käyttää jonkinlaisia musiikillista merkityksellisyyttä kuvaavia heuristiikkoja. Loppuosapuu voidaan muodostaa lähteen pituuden suhteen lineaarisessa ajassa, ja sen tilavaatimus on lähteen pituuden suhteen lineaarinen.

Edellä kuvattu loppuosapuutekniikka olisi ominaisuuksiensa vuoksi ideaalinen ensisijaiseksi hakumenetelmäksi, jos käytettäisiin indeksointia. Ongelmana on kuitenkin liiallinen keskusmuistitilavaatimus suuria kantoja käytettäessä. Tilavaatimus kasvaisi entisestään, jos loppuosapuuhun pitäisi tallettaa myös kestot. Menetelmää on esitetty käytettäväksi osaksi kolmivaiheista hakumenetelmää, jossa ensimmäisenä vaiheena olisi haku suppeahkosta loppuosapuusta, toisena vaiheena MonoPoly-haku ja kolmantena vaiheena jäljempänä kuvattava SIA(M)E- tai P2-haku [LWM01].

### 3.3.2 SIA(M)E

SIA(M)E-algoritmi (Structure Induction Algorithm (Matching) Express) hyväksyy osumat, jotka sisältävät vain osajoukon hahmon nuoteista [WLM02]. Se perustuu SIA-perheen algoritmeja vastaavaan tekniikkaan. Algoritmista on kaksi versiota, joista ensimmäisen tavoitteena on saavuttaa keskimääräinen aikavaatimus on $O(MN)$ ja toisen pahimman tapauksen aikavaatimus $O(MN \log MN)$ ja pienempi tilavaatimus.

Sovelluskohtaisilla valintaheuristiikoilla laajennettu SIA(M)E-algoritmi on nimetty SIA(M)ESE-algoritmiksi, mutta sopivia valintaheuristiikkoja ei toistaiseksi ole tarkemmin määritelty.

Algoritmin tavoitteena on laskea kaikki siirtovektorit ja sijoittaa kullakin vektorilla siirrettävissä olevat lähteen pisteet listaan kyseisen siirtovektorin yhteyteen. Ensimmäinen versio sijoittaa siirtovektorit hajautustauluun, jolloin kunkin siirtovektorin yhteyteen sijoitettavat pisteet voidaan tallettaa keskimäärin vakioajassa.

Algoritmin toinen versio SIA(M)$E_2$ tallentaa jokaisen siirtovektorin taulukkoon laskentajärjestyksessä. Laskennan ulompi silmukka käy läpi hahmon merkit ja sisempi lähteen, molemmat nousevassa järjestyksessä. Lopuksi siirtovektorit järjestetään lomittelujärjestelyä käyttäen, ja saman siirtovektorin omaavat lähteen pisteet yhdistetään listaksi.

Jos hahmo ja lähde ovat valmiiksi sanakirjajärjestyksessä, hahmon koon ja soinnun koon tulon suuruiset lohkot saadaan valmiiksi nousevaan järjestykseen muuttamalla suoritusjärjestystä niin, että ulompi silmukka käy lähteen läpi nousevassa järjestyksessä ja sisempi silmukka hahmon merkit laskevassa järjestyksessä. Kokonaisjärjestys saavutetaan liittämällä lohkot yhteen. Toteutusta vaikeuttaa se, että lohkot ovat vaihtuvapituisia.

Listaus 2 esittää muokatun algoritmin. Muuttujanimien käyttö vastaa listausta 1. Parametri $errors$ ilmoittaa, kuinka monta hahmon nuottia osumasta saa enintään puuttua. Riveillä 1–13 vektorit lasketaan taulukkoon $S$, jonka alkiot sisältävät kentät siirtovektorille ja listalle johon lisätään kyseiselle siirtovektorille kuvautuvat pisteet. Riveillä 14–31 liitetään parillinen määrä vaihtuvapituisia lohkoja lyhdeksi järjestetyksi jonoksi. Mahdollinen pariton lohko liitetään mukaan riveillä 32–43. Riveillä 44–54 yhdistetään saman siirtovektorin omaavat listat yhdeksi listaksi ja tulostetaan ne sointuindeksit, joista alkaa vähintään halutun minimipituuden mittainen osuma.

**Listaus 2.** Geometrinen algoritmi SIA(M)$E_2$.

```
1.  k = 0;
2.  for (ci = 0; ci < source.chords; ci++)
3.    for (pi = pattern.notes - 1; pi >= 0; pi--)
4.      for (ni = 0; ni < source[ci].length; ni++)
5.        S[k].vector = source[ci][ni] - pattern[pi];
6.        v = new listnode;
7.        v.chordindex = ci
8.        v.noteindex = ni
9.        v.patternindex = pi
10.       S[k].list = v;
11.     end
12.   end
13. end

14. if (source.chords % 2 != 0) odd = 1;
15. for (chunksize = 1; chunksize < source.chords - odd; chunksize = 2 * chunksize)
16.   for (ci = 0; ci < source.chords - chunksize - odd; ci += 2 * chunksize)
17.     lpos = tmppos = notes before the first chunk * pattern.notes;
18.     rpos = tmppos = notes before and in the first chunk * pattern.notes;
```

```
    leftend = rpos - 1;
    if (ci + 2 * chunksize >= source.chords - odd) rightend = S.length - odd - 1;
    else rightend = notes before and in the second chunk * pattern.notes;
    elements = rightend - lpos + 1;
    while (lpos <= leftend && rpos <= rightend)
      if (S[lpos] < S[rpos]) T[tmppos++] = S[lpos++];
      else T[tmppos++] = S[rpos++];
    end
    while (lpos <= leftend) T[tmppos++] = S[lpos++];
    while (rpos <= rightend) T[tmppos++] = S[rpos++];
    for (k = 0; k < elements; k++, rightend--) S[rightend] = T[rightend];
  end
end

if (odd)
  lpos = tmppos = 0;
  rpos = notes before the last chunk;
  leftend = rpos - 1;
  rightend = S_len - 1;
  while (lpos <= leftend && rpos <= rightend)
    if (S[lpos] < S[rpos]) T[tmppos++] = S[lpos++];
    else T[tmppos++] = S[rpos++];
  end
  while (lpos <= leftend) T[tmppos++] = S[lpos++];
  while (rpos <= rightend) T[tmppos++] = S[rpos++];
  for (k = 0; k < S.length; k++, rightend--) S[rightend] = T[rightend];

while (k < S.length)
  j = 1;
  while (k + j < S.length & S[k].vector = S[k + j].vector)
    S[k + j].list.next = S[k].list;
    S[k].list = S[k + j].list;
    j++;
  end
  if (j >= pattern.notes - errors)
    output(S[k].list.last.chordindex);
  k = k + j;
end
```

### 3.3.3 Pyyhkäisyviiva-algoritmit

Algoritmit P1, P2 ja P3 perustuvat ns. *pyyhkäisyviivatekniikkaan* (sweepline) [ULM03], jossa lähde pyritään käymään läpi vain kerran. Algoritmi P1 etsii lähteen kohdat, jossa hahmon nuottien korkeuksien ja alkuaikojen erotukset vastaavat lähteen nuottien korkeuksien ja alkuaikojen erotuksia [ULM03]. Se on siis transpositioinvariantti, ja osumien rytmi vastaa hahmon rytmiä. Väliin mahdollisesti jäävät koriste- tai muut nuotit eivät haittaa, ja aikavaatimus on $O(MN)$. P1 on merkkijonomenetelmien yhteydessä esitetyn naivin algoritmin (luku 3.2.1) geometrinen vastine.

Algoritmi P2 ratkaisee saman ongelman kuin aiemmin esitetyt SIA(M)E-algoritmit, mutta keskimääräinen aikavaatimus on vain $O(N)$, pahimman tapauksen aikavaatimus $O(NM \log M)$ [ULM03] ja tilavaatimus $O(M)$. Sen sijaan että kaikki erotusvektorit laskettaisiin taulukkoon ja järjestettäisiin laskennan jälkeen kuten SIA(M)E-algoritmeissa, P2-algoritmi liittää ne oikeaan järjestykseen prioriteettijonon avulla jo laskennan yhteydessä.

Algoritmin toiminta on seuraava. Hahmo pyyhkäisee lähteen yli epäsynkronisesti, toisin sanoen hahmon merkkejä ei välttämättä verrata lähteeseen järjestyksessä. Jokaista hahmon merkkiä vastaa osoitin, joka ilmaisee kyseistä hahmon merkkiä sillä hetkellä vastaavan lähteen merkin. Jokaista hahmon merkkiä vastaa myös siirtovektori, joka ilmaisee hahmon merkin ja vastaavan lähteen merkin erotuksen.

Alussa prioriteettijonoon lisätään lähteen ensimmäisen merkin ja hahmon kaikkien merkkien suhteen lasketut siirtovektorit. Samoin kaikki osoittimet asetetaan osoittamaan lähteen ensimmäiseen merkkiin.

Tämän jälkeen siirrytään pääsilmukkaan, jonka toiminta on seuraava. Prioriteettijonosta poistetaan pienin siirtovektori. Saatua siirtovektoria vastaava osoitin päivitetään osoittamaan lähteen seuraavaan merkkiin. Tämän jälkeen lasketaan uuden merkin ja siirtovektoria vastaavan hahmon merkin välinen siirtovektori ja lisätään se prioriteettijonoon. Jos laskettu uusi siirtovektori on sama kuin edellisellä kierroksella saatu, päivitetään laskuria, joka osoittaa kuinka monta samaa siirtovektoria on esiintynyt peräkkäin. Jos uusi siirtovektori ei ole sama, merkitään tulostaulukkoon siirtovektori ja sillä siirrettävissä olevien hahmon merkkien määrä (laskurin arvo). Jos siis siirtovektorilla voidaan kuvata koko hahmo lähteeseen, on löydetty täydellinen osuma. Muussa tapauksessa osuma on osittainen. Silmukkaa toistetaan lähteen viimeiseen merkkiin saakka.

Algoritmi P3 löytää siirtovektorit, joilla hahmon ja lähteen viivasegmenttien (janojen) leikkauksen pituus eli yhteinen kestoaika on suurin [ULM03]. Aikavaatimus on $O(MN \log MN)$. P3 on ainoa kirjallisuudessa esitetty ratkaisu tähän ongelmaan.

# 4 Toteutetun prototyyppijärjestelmän esittely

## 4.1 Olemassaolevia musiikkihakujärjestelmiä

Yleiskatsaus muihin olemassaoleviin järjestelmiin on esitetty mm. Kjell Lemströmin väitöskirjassa [Lem00]. Yksi tunnetuimmista on Uudessa Seelannissa toteutettu MELDEX, joka tukee hakuja vain monofonisesta aineistosta [Bai98]. MELDEX-järjestelmän uusi versio on valmisteilla. Helsingin Yliopistossa on aiemmin toteutettu polyfoniseen hakuun kykenevä järjestelmä SEMEX, joka käyttää mm. MonoPoly-algoritmia [LeP00]. Se ei kuitenkaan ole julkisesti saatavilla.

Uudempi GUIDO/MIR-prototyyppi käyttää esitysmuotonaan GUIDO-muotoa [HRG01]. Lähtökohta on hieman eri tyyppinen kuin tässä työssä esiteltävässä prototyypissä. Muunnosta MIDI-muotosta GUIDO-muotoon ei voida tehdä automaattisesti, joten prototyyppi sisältää vain 150 kappaletta, ja kokoelman kasvattaminen vaatii paljon käsityötä.

Guidossa hakuja voidaan tehdä absoluuttisten nuotinkorkeuksien, intervallien, intervallityyppien (esimerkiksi terssi), intervalliluokkien (sama, pieni, keskisuuri, suuri) ja melodisen trendin (sama, ylos, alas) perusteella. Rytmisiä hakuparametreja ovat absoluuttiset ja suhteelliset kestot sekä rytminen trendi (sama, lyhyempi, pitempi). Haku voi perustua vain nuotinkorkeusparametreihin tai vain rytmisiin parametreihin, tai näiden yhdistelmään. Lisäksi sekä eksaktit että likimääräiset haut ovat mahdollisia [HRG01].

## 4.2 Toteutusteknologiat ja nuottikokoelma

Ohjelmiston toteutuskieleksi on valittu uudehko tulkittava oliokieli Ruby, jota voidaan suurempaa tehokkuutta tarvittaessa laajentaa C-kielisillä moduleilla. Algoritmit ja HTML-sivujen generointi on toteutettu C-kielellä, muut osat Ruby-kielellä. Lähestymistapa mahdollistaa suorituskyvyn kannalta epäolennaisten kohtien nopean ja joustavan toteutuksen ja muokkauksen turvaten kuitenkin riittävän suorituskyvyn. Koodirivimäärän pieneneminen parantaa myös ylläpidettävyyttä.

Hakutietojen syöttö tapahtuu HTML-lomakkeella tai Java-sovelmalla. Ne kutsuvat skriptejä, jotka puolestaan ottavat yhteyden palvelinprosessiin etäoliokutsurajapinnan avulla.

Järjestelmään on tällä hetkellä toteutettu algoritmit ShiftOrAnd, IntervalMatching, MonoPoly, SIA(M)$E_2$ ja P1. MonoPoly- ja IntervalMatching-algoritmeille on toteutettu kaksi tarkistusalgoritmia, joista toinen on tavanomainen naivi algoritmi ja toinen luvun 3.2.1 listauksessa 1 kuvattu muunnettu algoritmi. Suodatusalgoritmeilla hakuaika on kääntäen verrannollinen tarkistettavien osumakandidaattien määrään, koska tarkistusalgoritmi on suodatusalgoritmia hitaampi. Online-algoritmeilla suoritusaika on likimain vakio, poikkeuksena SIA(M)E-algoritmi, joilla hahmon pidentäminen pidentää hakuaikaa.

Esittelyjärjestelmässä käytetään tekijänoikeusvapaata Mutopia-nuottikokoelmaa, joka sisältää noin 250 klassista musiikkiteosta Lilypond-lähdekoodeineen, joista on generoitu MIDI-tiedostot sekä PS- ja PDF-muotoiset nuottikuvatiedostot. Lisäksi jokaisen musiikkiteoksen yhteyteen kuuluu kappalenimen, tekijän ym. määrittelevää metatietoa. Järjestelmässä voidaan käyttää mitä tahansa MIDI-kappalekokoelmaa, mutta tällöin nuottikuva ja metatieto eivät ole käytettävissä tulosten esittämisen yhteydessä.

## 4.3 Luokkarakenne

Järjestelmän sisältämä musiikkitieto on jaettu kokoelmiin, jotka sisältävät kappaleita. Kappaleet sisältävät nuottitiedon, varsinaisesta kannasta MonoPoly-algoritmille muodostetun offline- eli esikäsitellyn muodon, sekä kappaletta kuvailevan tekstitiedon kuten kappalenimen ja säveltäjän.

SEMEX-järjestelmässä kappaleiden nuottitiedot oli sijoitettu yhtenäiseksi sekvenssiksi. Tällöin osumista oli erikseen poistettava kappalerajan ylittävät osumakandidaatit. Uudessa järjestelmässä haku kohdistetaan jokaiseen kappaleeseen erikseen, mikä poistaa tarkistustarpeen mutta toisaalta aiheuttaa ylimääräisiä funktiokutsuja. Uusi rakenne helpottaa kuitenkin kappaleiden osajoukkoihin kohdistettuja hakuja ja on intuitiivisempi.

Lähinnä hahmon oliopohjaisen käsittelyn helpottamiseksi mukana ovat luokat nuotille ja soinnulle. Algoritmien alustusparametrien ja osumien talletukseen on muodostettu omat luokkansa.

MIDI-tiedostojen luku tapahtuu erillisen SMF-tiedostojen käsittelyyn tarkoitetun kirjaston avulla, ja sen vaatimat callback-funktiot on sijoitettu omaan luokkaansa.

## 4.4 Tietorakenteet

SMF-tiedostoista on luettava Note On ja Note Off -kanavaviestit nuottitietoja varten sekä Time-Signature-viesti tahdin laskentaa varten. Viestitapahtumiin reagointi määritetään callback-funktioiden avulla.

MIDI-tapahtumista muodostetaan tietorakenne, jossa samanaikaisesti alkavat nuotit on sijoitettu sointuihin. Soinnun parametreja ovat alkuaika, soinnun nuottien määrä ja lista soinnun nuoteista.

Nuotin parametrimäärää voidaan vaihdella. Tällä hetkellä nuotin parametreina käytetään äänenkorkeutta, ääntä (soitin) ja kestoa. Kesto on merkitty MIDI-tapahtumien loppu- ja alkuaikojen erotuksena. Yksikkö riippuu SMF-muotoon talletetusta resoluutiosta, joka myös talletetaan kappaleobjektiin. Symboliset kestot voidaan laskea likimääräisesti resoluution ja absoluuttisen keston avulla.

Soinnut ja nuotit on talletettu tavusekvenssiksi, jossa soinnun pituudelle on varattu 1 tavu, alkuajalle 4 tavua, nuotin korkeudelle 1 tavu, kestolle 2 tavua ja äänelle 1 tavu. Monopoly-algoritmin vaatima offline-muoto muodostuu sekvenssistä, jossa jokaiselle soinnulle on varattu 2 tavua (12 bitin tallettamista varten). Tämä muoto muodostetaan varsinaisen kannan muodostamisen yhteydessä.

SMF-tiedostomuoto ei tue tahtiviivojen tallentamista. Tahtinumero voidaan kuitenkin laskea neljäsosanuotin keston, tahtilajimerkinnän ja osuman ensimmäisen nuotin (soinnun) alkuajan avulla. Tahtilaji voi vaihdella kappaleen aikana, joten tahtinumeroa on kasvatettava paloittain. Näin saatu tahtinumero on likimääräinen, koska laskennassa ei voi ottaa huomioon esim. ensimmäisen tahdin mahdollista vajautta.

## 4.5 Suorituksen kulku

### 4.5.1 Asiakas

HTML-sivulta kutsutaan asiakasskriptiä, joka tarkistaa joitakin haun oikeellisuuteen liittyviä tekijöitä kuten hahmon pituuden, muodostaa tekstinä annetusta hahmosta sisäisen esityksen, ja asettaa algoritmiparametrit. Tämän jälkeen se luo palvelimesta etäobjektin, ja kutsuu tämän objektin hakumetodia. Haun tuloksena saadaan palvelimelta valmis HTML-taulukko, joka liitetään osaksi generoitavaa HTML-sivua. Sivulle liitetään myös mitatut suoritusajat.

### 4.5.2 Palvelin

Käynnistettäessä palvelin lukee halutut kappaletietokannat keskusmuistiin, luo etäobjektin ja jää kuuntelemaan hakupyyntöjä. Hakupyynnön yhteydessä skripti välittää hahmon, algoritmin nimen, hakutulosten maksimimäärätiedon, tiedon siitä näytetäänkö yhdestä kappaleesta vain yksi osuma vai kaikki osumat, järjestämiskriteerin ja tarkistusfunktion jos kyseessä on MonoPoly-algoritmi. Tämän jälkeen kutsutaan algoritmin alustusfunktiota, ja siltä saatu alustetut parametrit sisältävä objekti välitetään kutsuttavalle varsinaiselle hakufunktiolle.

Hakufunktiota kutsutaan jokaiselle kokoelmalle erikseen. Jokainen kokoelma kutsuu puolestaan hakufunktiota erikseen jokaiselle kappaleelle. Hakutulokset liitetään yhteiseen hakutuloslistaan.

Hakutuloksen löytymisen yhteydessä sille lasketaan relevanssikriteeri oktaavisiirtymien ja transponoinnin määrän perusteella.

Haun valmistuttua listasta poistetaan ensin useat samaan kappaleeseen kohdistuvat osumat, jos tätä on pyydetty. Tämän jälkeen osumat järjestetään relevanssikriteerin mukaan nousevaan järjestykseen. Lopuksi tuloslistan koko rajataan halutun maksimimäärän pituiseksi, listasta muodostetaan HTML-taulukko ja taulukko palautetaan kutsuvalle skriptille.

# 5 Yhteenveto

Musiikillista melodiaa voidaan etsiä moniäänisestä symbolisesta musiikkitiedosta merkkijonomenetelmiin perustuvilla algoritmeilla. ShiftOrAnd-algoritmilla voidaan hakea tarkkoja osumia, ja IntervalMatching- ja MonoPoly-algoritmeilla sävellajiriippumattomia osumia.

Geometriset algoritmit SIA(M)E, P1 ja P2 soveltuvat kuitenkin paremmin moniäänisen aineiston käsittelyyn. Niillä voidaan etsiä moniäänistä hahmoa moniäänisestä aineistosta. P1-algoritmi löytää rytmin säilyttäviä sävellajiriippumattomia osumia, ja ja SIA(M)E- ja P2-algoritmit rytmin säilyttäviä sävellajiriippumattomia osittaisia osumia.

Algoritmeja on toteutettu julkisesti käytettävissä olevaan prototyyppiohjelmistoon [WWW03], joka käyttää lähdeaineistonaan SMF-muotoisia musiikkitiedostoja.

# Lähteet


BaG92 Baeza-Yates, R., Gonnet, G. H., Fast text searching allowing errors. *Communications of the ACM*, 35(10), 1992, 74–82.

Bai98 Bainbridge, D., MELDEX: a web-based melodic index service. Teoksessa [HeS98], 223–230.

CaB02 Cano, P. et al., Robust sound modeling for song detection in broadcast audio. *Proc. 112th AES Convention*, München, Germany, 2002.

Bou00 Boulanger, R. (toim.), *The Csound Book*. MIT Press, 2000.

BrP98 Brin, S., Page, L., The anatomy of a large-scale hypertextual web search engine. *Proc. 7th WWW Conference*, Brisbane, Australia, April 1998, 107–117.

CrR94 Crochemore, M., Rytter, W., *Text Algorithms*. Oxford University Press, 1994.

Dan93 Dannenberg, R. B., A brief survey of music representation issues, Techniques and Systems. *Computer Music Journal*, 17(3), 1993, 20–30.

HeS98 Hewlett, W., Selfridge-Field, E. (toim.), Melodic Similarity: Concepts, Procedures, and Applications. *Computing in Musicology 11*, MIT Press, 1998.

HeS01 Hewlett, W. B., Selfridge-Field, E. (toim.), The Virtual Score: Representation, Retrieval, Restoration. *Computing in Musicology 12*, MIT Press, 2001.

HRG01 Hoos, H., Renz, K., Görg, M., GUIDO/MIR - an experimental musical information retrieval system based on GUIDO music notation. *Proc. 2nd International Symposium on Music Information Retrieval*, Bloomington, USA, October 15–17, 2001, 41–50.

LeL98 Lemström, K., Laine, P., Musical information retrieval using musical parameters. *Proc. 1998 International Computer Music Conference*, University of Michigan, Ann Arbor, USA, October 1–6, 1998, 341–348.

Lem00 Lemström, K., *String Matching Techniques for Music Retrieval*. Helsingin yliopisto, Tietojenkäsittelytieteen laitoksen julkaisusarja A, raportti A-2000-4.

LeP00 Lemström, K., Perttu, S., SEMEX - an efficient music retrieval prototype. *Proc. 1st International Symposium on Music Information Retrieval*, Plymouth, Massachusetts, USA, October 23–25, 2000.

LeT03 Lemström, K., Tarhio, J., Transposition invariant pattern matching for multi-track strings. Toimitettu julkaistavaksi.

LWM01 Lemström, K., Wiggins, G .A., Meredith, D., A three-layer approach for music retrieval in large databases. *Proc. 2nd International Symposium on Music Information Retrieval*, Bloomington, USA, October 15–17, 2001, 13–14.

MWL01 Meredith, D., Wiggins, G. A., Lemström K., Pattern induction and matching in polyphonic music and other multidimensional datasets. *Proc. 5th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA, July 22–25, 2001, 61–66.

Sal97 Salmenhaara, E., *Sointuanalyysi*. Otava, 1997.

Scr00 Schreier, E. D., The MPEG-4 structured audio standard. *Proc. IEEE ICASSP*, 1998.

Sel97 Selfridge-Field, E. (toim.), *Beyond MIDI*. MIT Press, 1997.

ShS02 Shalev-Shwartz S. et al., Robust temporal and spectral modeling for query by melody. *Proc. 25th Conf. Research and Development in Information Retrieval*, Tampere, August 2002.

TzE01 Tzanetakis, G., Essl, G., Cook, P., Automatic musical genre classification of audio signals. *Proc. 2nd International Symposium on Music Information Retrieval*, Bloomington, USA, October 15–17, 2001, 205–210.

ULM03 Ukkonen E., Lemström K., Mäkinen V., Sweepline the music! Teoksessa Klein, R., Six, H.-W., Wegner, L. (toim.), *Computer Science in Perspective (LNCS 2598)*, 330–342, 2003.

WLM02 Wiggins, G., Lemström, K., Meredith, D., SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. *Proc. 3rd International Symposium on Music Information Retrieval*, Paris, France, October 13–17, 2002, 283–284.

WWW00 Musical notation codes. `http://www.music-notation.info/en/compmus/notationformats.html`. [15.03.2003]

WWW01 LilyPond-ohjelmiston kotisivu. `http://www.lilypond.org/`. [15.03.2003]

WWW02 ISMIR-konferenssin kotisivu. `http://www.ismir.net/`. [15.03.2003]

WWW03 C-Brahms -tutkimusryhmän kotisivu. `http://www.cs.helsinki.fi/group/cbrahms/`. [15.03.2003]