VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Muzikinių kūrinių indeksacija ir greita paieška

Atliko: 2 kurso, 2 grupės studentas
Viktoras Žalpys (parašas)

Darbo vadovas:
doc. dr. Algirdas Bastys (parašas)

Vilnius
2012

Turinys

Anotacija 4
Summary 5
Įvadas 6
1 Muzikos kūrinių indeksacija ir paieška 7
2 Egzistuojantys muzikos paieškos produktai ir naudojami algoritmai 9
2.1 Algoritmų grupės 9
2.1.1 Query by humming 9
2.1.2 Acoustic fingerprint 9
2.2 Algoritmai 10
2.2.1 Shazam 10
2.2.2 Shazam algoritmas 10
2.2.3 Midomi 12
2.2.4 Midomi algoritmas 13
2.2.5 Soundhound 13
2.2.6 Last.fm 13
2.2.7 Last.fm algoritmas 13
2.2.8 ECE algoritmas 14
2.2.9 Ivonos Šurpickos algoritmas 16
2.3 Algoritmų palyginimas 18
3 Praktinis muzikos paieškos įgyvendinimas 19
3.1 Muzikos kūrinio nuskaitymas 19
3.2 Spektogramos generavimas 20
3.3 Požymių išrinkimas iš dainos 21
3.4 Atspaudo generavimas 23
3.5 Kūrinių lyginimas 24
3.6 Greitasis algoritmas 25
3.7 Greitųjų raktų sudarymas 25
3.8 Greitasis kūrinių lyginimas 26
3.9 Algoritmų techninis palyginimas 27
4 Eksperimentai 28
4.1 Algoritmo nustatymų reikšmė atpažinimui 28
4.2 Atpažinimo kokybės priklausomybė nuo kūrinio trukmės 30
4.3 Jautrumas triukšmui 32
4.4 Algoritmų apjungimo galimybė 37
4.5 Algoritmų techniniai duomenys 38

4.6 Palyginimas su kitais algoritmais 38
Išvados 40
Literatūros sąrašas 41

Anotacija

Šio darbo tikslas – pasiūlyti naują algoritmą muzikos kūrinių indeksacijai ir paieškai. Tikslui pasiekti formuluojami uždaviniai ir reikalavimai naujai pasiūlytam algoritmui. Taip pat darbe išnagrinėjami šiuo metu naudojami algoritmai muzikos indeksacijai ir paieškai.

Kitoje darbo dalyje pateikiamas algoritmas, kuriam naudojami Teiloro koeficientai padeda išskirti muzikos požymius. Išskirtų muzikos požymių palyginimui pateikiamos dvi algoritmo versijos: greitoji versija, kuri naudojasi *hash* raktais, ir lėtoji versija, naudojanti daugiau duomenų muzikos palyginimui.

Rasti algoritmai testuojami eksperimentinėje darbo dalyje – tikrinamas algoritmų atsparumas triukšmui, jų priklausomybė nuo užklausos trukmės. Taip pat algoritmų rezultatai lyginami ir su kitais algoritmais. Gauti rezultatai parodo, kad algoritmai geba atpažinti muzikos kūrinį esant trisdešimt penkių decibelų triukšmui tik iš trisdešimties sekundžių įrašo.

Summary

Indexation and fast Searching of Music Composition.

The goal of this work is to propose a new algorithm for music indexing and searching. To achieve this, objectives and requirements were formulated for the newly proposed algorithm. State of the art algorithms for music indexing and searching were also examined.

Following that, an algorithm that uses Taylor coefficients to distinguish music features was suggested. To compare music features, two algorithm versions were suggested: a quick version that uses hash keys, and a slow version, using more data to compare the music.

The suggested algorithms are tested in the experimental part. Noise immunity and their dependence on the length of the query are checked. The results are compared with those of the state of the art algorithms. They show that the algorithm is able to recognize a music that has thirty-five decibel noise and only from a thirty seconds query.

Įvadas

Meniškas garsų komponavimas - muzikos kūriniai - žmogaus gyvenimą lydi nuo neatmenamų laikų. Tobulėjant žmogaus naudojamoms technologijoms, tobulėjo ir muzika, atsirado galimybė ją užrašyti popieriaus lape, po to įrašyti į plokštelę, magnetinę kasetę. Vėliau analoginį formatą pakeitė skaitmeninis formatas, taip suteikdamas galimybę lengvai apdoroti muziką kompiuteriais.

Muziką perkėlus į kompiuterius, buvo galima paprasčiau išspręsti su muzika iškilusius uždavinius, kurie žmogui nebūdavo trivialūs. Vienas iš šių uždavinių - grojamo kūrinio atpažinimas. Šiam uždaviniui išspręsti galime pasitelkti ne tik kompiuterius, bet ir sparčiai tobulėjančius išmaniuosius telefonus, kurie savyje geba ne tik įrašyti aplinkos grasus, bet ir atlikti sudėtingus skaičiavimus.

Muzikos atpažinimas yra sudėtingas procesas, kurio problematika yra nagrinėjama šiame darbe. Išanalizavau skirtingus algoritmus, sukurtus spręsti muzikos atpažinimo uždaviniui. Pasiūliau savo algoritmą spręsti šiam uždaviniui.

Norėdamas pasiekti darbo tikslą, iškėliau sau tokius uždavinius:

1. Išanalizuoti esamus muzikos indeksacijos ir paieškos algoritmus bei nustatyti jų trūkumus ir privalumus;
2. Įgyvendinti ir optimizuoti muzikos indeksacijos algoritmą;
3. Atlikti tyrimus, rodančius algoritmo veikimo kokybę.

Darbas susideda iš teorinės ir praktinės dalių. Teorinėje dalyje detaliai išanalizavau muzikos atpažinimo užduoties sprendimo būdus. Praktinėje dalyje, pasitelkęs žinias iš pirmosios dalies, įgyvendinau du algoritmus, viename iš jų optimizuodamas paieškos laiką.

1 Muzikos kūrinių indeksacija ir paieška

Norint suprasti muzikos indeksacijos uždavinį ir jo iškeliamus sunkumus bei problemas, reikia jį apibendrinti. Mūsų atveju muzikos kūrinių indeksacija laikome muzikos kūrinio požymių išskyrimą ir jų pritaikymą siekiant identifikuoti kūrinį sukuriant tik jam unikalų indeksą. Šis indeksas turėtų užimti mažiau atminties vietos nei indeksuojamas muzikos kūrinys. Saugant tik kūrinio indeksą, sutaupoma kompiuterio atmintis.

Muzikos kūrinio paieška suteikia galimybę identifikuoti kūrinį, esantį duomenų bazėje. Prieš muzikos paiešką kūrinys turi būti nuskaitomas ir indeksuojamas, o tik tada atliekama jo atitikmens paieška duomenų bazėje. Be to, suformuluojame tikslą, kad muzikos paiešką būtų galima atlikti ne tik geros kokybės kūriniams, bet ir kūriniams, kurie nėra pilnos trukmės arba kūriniams su triukšmu. Muzikos atpažinimo procesas savo kokybe turėtų prilygti žmogaus klausai. Kūrinio užklausa gali būti įrašyta diktofonu ir pateikta paieškai.

Pirmoji problema, su kuria susiduriama atliekant muzikos paieškos užduotį, yra muzikos kūrinio trukmė. Užklausos trukmė nėra nurodyta. Kūrinys gali būti ne pilnos trukmės ir kūrinį tenka atpažinti tik iš penkiolikos sekundžių įrašo.

Antroji problema yra taip pat susijusi su užklausos įrašo trukme. Be papildomų skaičiavimų negalime nusakyti, kuri muzikos kūrinio dalis patenka į užklausą. Todėl negalime tiksliai nusakyti pateiktos užklausos vietos, nes tai gali būti ir pradžia, ir pabaiga.

Trečioji problema yra pateikto muzikos kūrinio užklausos kokybė. Kaip ir minėjau, muzikos paieška turi atpažinti kūrinį, kurio įrašas turi pašalinių garsų. Kaip pavyzdį galima paminėti diktofonu įrašytus fragmentus. Jie daugeliu atvejų turės pašalinių garsų, taip sukeldami problemų identifikuojant kūrinį.

Ketvirtoji problema - duomenų bazės surinkimas ir saugojimas. Norint užtikrinti gerą muzikos kūrinių paiešką, reikia sukaupti duomenų bazę, turinčią daug muzikos kūrinių įrašų. Čia susiduriame su šiais uždaviniais: pirmasis - muzikos kūrinių paieška ir jų nuskaitymas duomenų bazėje, antrasis - užtikrinti, kad kūrinių saugojimas neužimtų daug atminties vietos duomenų bazėje. Šiems uždaviniams pasiekti, kaip ir minėta, bus saugoma tik muzikos kūrinių indeksai, o ne realūs kūriniai.

Penktoji problema - tai kūrinio paieškos greitis. Pateikę užklausą, atsakymą gauti tikimės akimirksniu. Pateiktos užklausos paieška yra atliekama didelėje duomenų bazėje, todėl rezultato

gavimo laikas priklauso nuo atliekamų palyginimų skaičiaus. Vieno palyginimo greitis tampa svarbiu faktoriumi.

2 Egzistuojantys muzikos paieškos produktai ir naudojami algoritmai

Šiame skyriuje nagrinėsiu rinkoje egzistuojančius produktus ir jų taikomus algoritmus, kurie atlieka muzikos paiešką ir indeksaciją. Muzikos indeksacijos ir paieškos algoritmai skirstomi į dvi grupes: *Query by humming* ir *Acoustic fingerprint*. Pirmiausia apžvelgsiu šias grupes. Po to išnagrinėsiu keturias populiariausias muzikos atpažinimo sistemas. Darbe bandysiu analizuoti ir jų veikimo principus. Nors ne visi produktai atskleidžia savo algoritmus muzikai atpažinti.

2.1 Algoritmų grupės

2.1.1 Query by humming

Query by humming muzikos indeksavimo ir paieškos grupė ypatinga tuo, kad gali atpažinti muzikos kūrinį iš vartotojo sudainuoto kūrinio dalies, parašyto ritmo ar dainos žodžių. Algoritmams, priklausantiems šiai grupei, muzikos kūrinio atpažinimui nereikia turėti net jo originalaus įrašo. Šio tipo paieškos sistemos dažniausiai kuriamos naudojantis *MPEG-7* muzikos formatu. *Query by humming* algoritmai dažniausiai susiduria su šiomis problemomis: dainos žodžių praleidimas, ritmo sumaišymas. Čia naudojami algoritmai, išskiriantys tokias kūrinio savybes, kaip garso ritmas, intervalai ir trukmė. [VAL10]

2.1.2 Acoustic fingerprint

Kita muzikos paieškos ir indeksavimo algoritmų grupė - *Acoustic fingerprint*. Šios algoritmų grupės uždaviniai skiriasi tuo, kad atpažįsta muzikos kūrinį tik iš įrašytos kūrinio dalies. Įrašo kokybei nėra keliami griežti reikalavimai. Algoritmo tikslas – atpažinti muzikos kūrinį tik iš gauto įrašo diktofonu. Algoritmai vykdydami užduotį susiduria su tuo, jog įrašo kokybė ne visada yra gera. Taip pat nėra žinomas ir laikas, kada įrašinėjama pateikta užklausa. Be to, įrašomų kūrinių atspaudai negali būti lyginami tiesiogiai, nes originalus kūrinys gali skirtis bitų lygmenyje, nors žmogaus ausis girdi tą patį. Kita bitų lyginimo problema – muzikos formatai iškraipo kūrinio bitus.

Šių algoritmų principai pagrįsti muzikos kūrinio atspaudo (*fingerprint*) išskyrimu[HAI02].

2.2 Algoritmai

2.2.1 Shazam

Pirmasis produktas, kurį apžvelgsiu, - tai *Shazam*. Šis produktas pasirodė 2002 metais Jungtinėje karalystėje. Pirmosios versijos veikimo principas buvo paprastas. Vartotojas surinkdavo trumpąjį numerį ir skambučio metu pridėdavo telefoną kuo arčiau garso šaltinio, dažniausiai radijo garsiakalbio. Įrašui pasibaigus, sulaukdavo trumposios žinutės su dainos pavadinimu. Jau nuo pirmosios versijos *Shazam* sugebėdavo atpažinti muzikos kūrinį iš tokios kokybės įrašo, kaip telefono skambutis.

Vėlyvesnės *Shazam* versijos tobulėjo kartu su mobiliaisiais telefonais. Labiausiai buvo tobulinamas įrašomo garso pateikimas sistemai. Šiuo metu sistema yra prieinama populiariausiose išmaniųjų telefonų operacinėse sistemose. Deja, ši programa vis dar nesugeba atpažinti dainuojamų įrašų iš koncertų. Ji priskiriama *Acoustic fingerprint* algoritmų grupei.[WAN03]

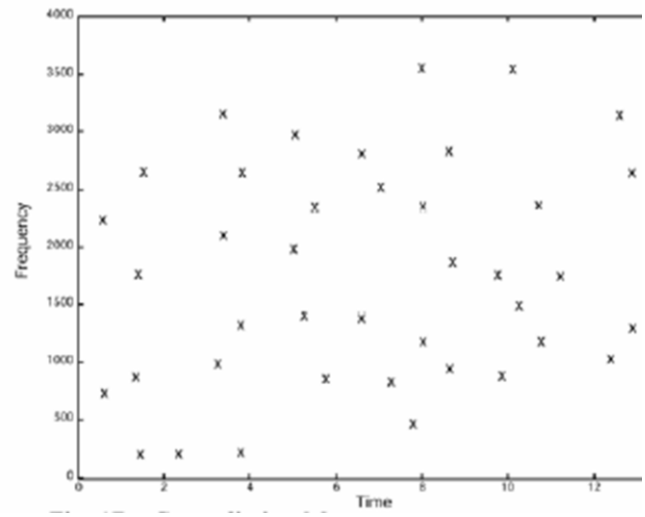
2.2.2 Shazam algoritmas

Shazam algoritmas parašytas ir patentuotas *Avery Li-Chun Wang*. Šio algoritmo idėja - muzikos kūrinių paiešką atlikti išskiriant muzikos požymius (*fingerprints*) visiems kūriniams, esantiems duomenų bazėje, ir taip pat pateiktai užklausai. Algoritmas padalintas į keletą žingsnių, kuriuos išnagrinėsiu detaliau.

Pirmiausia, algoritmas nuskaito muzikos kūrinius ir konvertuoja garso formatą į PCM [WAN03]. Tada naudodamas konvertuotus duomenis generuoja spektogramą.

Apibrėžimas. Spektograma - tai 3 matmenų diagrama, kurioje abscisių ašis vaizduoja laiką, ordinačių ašis vaizduoja dažnį, o taško spalva - garso intensyvumą.[SPE11]

Toliau algoritmas analizuoją spektogramą ir ieško ekstremumų taškų (*peaks*). Šie taškai randami ieškant tamsiausių spektogramos taškų ten, kur garso energija yra didžiausia, ir ten, kur jie išsiskiria iš aplinkinių taškų. Šie taškai pasirenkami todėl, kad čia yra didžiausia galimybė, jog jie išliks paveikus muzikos kūrinį triukšmu. Radus visus svarbiuosius taškus spektogramoje, sudaromas *konsteliacijos* žemėlapis (*constellation map*). Šis žemėlapis vaizduoja tik stipriausius taškus, jų laiką abscisių ašyje, o dažnį ordinačių ašyje. [WAN03]

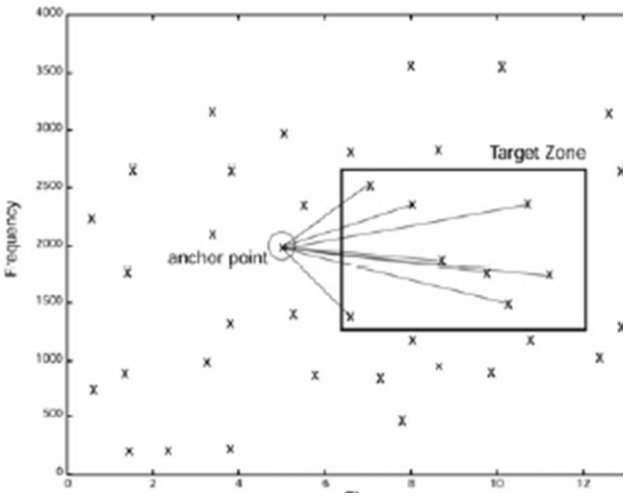


1 paveikslėlis. [WAN03] Constellation Map

Kadangi algoritmui veikti funkcionaliai reikalingas didelis kiekis palyginamųjų kūrinių, saugoti kiekvieną kūrinį ar jo spektogramą neprasminga, tai užimtų didžiulę atminties vietą. Todėl kitu žingsniu yra suformuojami duomenys, kurie bus saugomi domenų bazėje. Duomenų bazėje bus saugomi *hash* kodai. Jie sudaryti iš svarbiųjų taškų naudojantis formule:

$$hash = [f1:f2:\Delta t].$$

Šioje formulėje f1 atitinka pirmąjį stiprųjį tašką, o f2 – antrąjį. Δt – tai f1 ir f2 taškų laiko skirtumas. Saugomi du taškai todėl, kad norima išvengti *hash* kodų persidengimų. F1 ir f2 taškų poros nėra sudaromos atsitiktinai. Visi taškai yra padalyti į sritis (*target zone*) ir kiekvienai iš jų priskiriamas pagrindinis taškas (*anchor point*). Tada visi srityje esantys taškai sudaro porą su pagrindiniu tašku.[WAN03]



2 paveikslėlis. [WAN03] Spektogramos taškų išskyrimas

Algoritmo aprašyme teigiama, kad šis *hash* įrašas turi užimti 32 bitus. [WAN03]

Sudarius visas poras kartu su *hash* įrašu, saugomas ir laikas taško, kuriam sudarytas *hash* įrašas, todėl duomenų bazėje vieną svarbųjį tašką atitiks 64bitų struktūros. [WAN03]

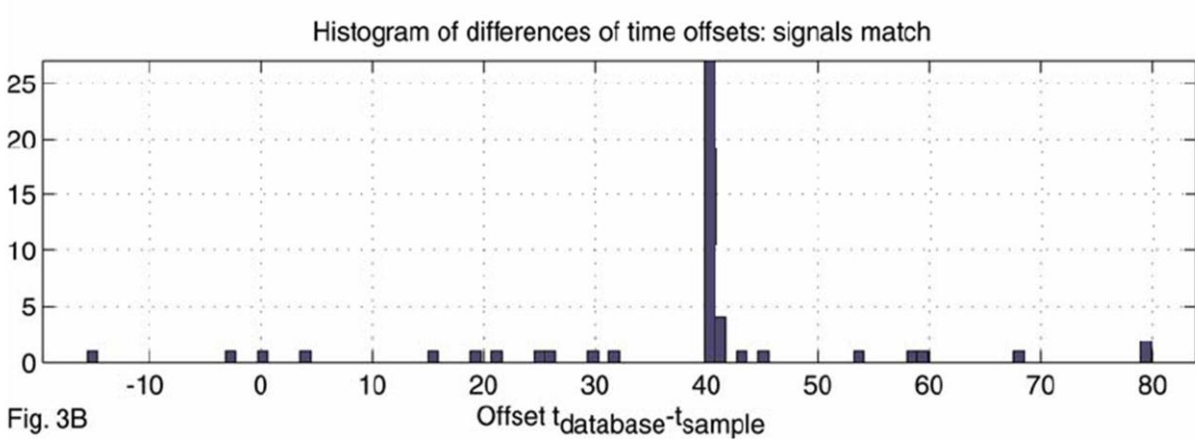
hash kodas : pirmojo taško laikas : kūrinio indeksas.

Šios struktūros duomenų bazėje surikiuotos pagal *hash* kodą.

Tada algoritmas atlieką lyginimą. Lyginimui atlikti pasitelkiamas maišos algoritmas (*combinational hash*). Lyginimo algoritmo idėja – tų pačių muzikinių kūrinių svarbieji taškai pasikartos tokiu pačiu laiko skirtumu. Kadangi užklausos įrašymo momentas nėra žinomas, panašūs kūriniai atitiks tokią formulę:

$$t_{ieškomas} - t_{orginalus} = konstanta.$$

Akivaizdu, kad tokia formulė bus tinkama daugeliui svarbių įrašų. *Shazam* algoritmas pasiūlė būdą, kaip greitai patikrinti, ar kūriniai sutampa, ar ne. Algoritmas ieškos *hash* sutapimų duomenų bazėje. Sudaroma histograma visiems kūriniams, kurių sutapimai yra aptikti. Jei histogramoje didžiausia reikšmė išsiskiria, lyginant su kitomis, laikoma kad yra sutapimas.



3 paveikslėlis. [WAN03] Shazam muzikos kūrinių lyginimo histograma

2.2.3 Midomi

Kitas produktas, kurį aptarsiu - *Midomi* sistema. Ši sistema taip pat atpažįsta muziką, tik tai daro kitu būdu, nei prieš tai aptartoji. *Midomi* programa sugeba atpažinti dainą ne vien tik iš įrašyto, bet ir įdainuoto dainos fragmento. Be to, kiekviena tokia užklausa tampa *Midomi* duomenų šaltiniu. Taigi ši sistema priklauso *Query by humming* algoritmų grupei.[MID07]

2.2.4 Midomi algoritmas

Midomi naudoja *Multimodal Adaptive Recognition System* algoritmą [MID07]. Deja, detalus šio algoritmo veikimo principas nėra atskleistas. Žinomos tik kai kurios detalės. Programa yra dinamiška – atpažįstanti įrašo kokybę (aiškus žmogaus balsas ar įrašas iš koncerto). Šios sistemos algoritmas reaguoja į garso tono aukštumą, tarpus tarp garsų bei ritmą ir kalbos fonetinį turinį.

2.2.5 Soundhound

Soundhound taip pat, kaip ir *Shazam* muzikos kūrinio paiešką atlieka naudojantis mobiliuoju telefonu. Be to, ji sugeba atpažinti ne tik įrašytus originalius kūrinius, bet ir žmogaus įdainuotus kūrinius ar net pasakytus dainos žodžius. *Soundhound* paieškai naudoja *Sound2Sound* technologiją, kurią sukūrė *Soundhound* komanda [SOU11]. Deja, daug detalių apie šią technologiją nėra atskleidžiama. Nurodoma, kad, kaip ir prieš tai minėtos dvi technologijos, ji gautą dainos fragmento užklausą „kristalizuoja“ ir ieško panašaus fragmento sukurtoje duomenų bazėje. Be to, *Sound2Sound* duomenų bazę sudaro ne tik gauta garso informacija, bet ir kita papildoma informacija, teksto duomenys, informacija apie atlikėją ir kita.

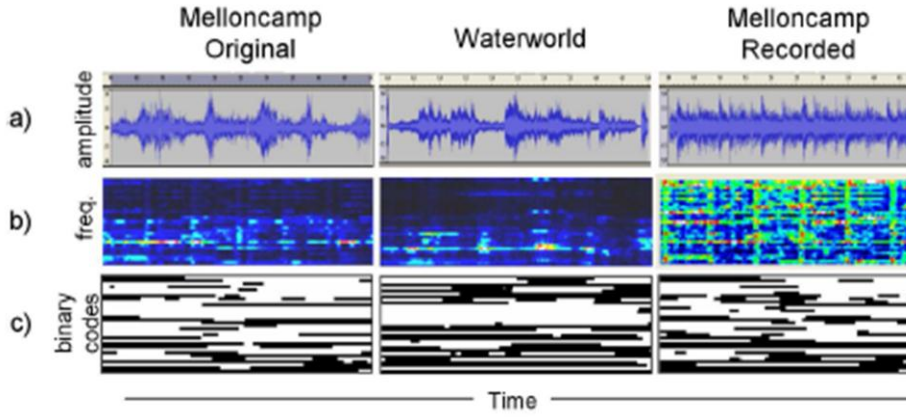
2.2.6 Last.fm

Šios programinės įrangos paslauga skiriasi nuo prieš tai pristatytų. Jos pagrindinis tikslas nėra atpažinti muziką iš mažos muzikos trukmės. Ji naudojama siekiant vartotojui pateikti klausomos muzikos išsamiausias rekomendacijas. Nors ji ir nėra panaši į prieš tai apžvelgtas sistemas, ši sistema taip pat pasitelkia muzikos atpažinimo algoritmus. *Last.fm* sugeba atpažinti dainą tik iš trumpo jos įrašo. Toliau apžvelgsiu kaip tai veikia.

2.2.7 Last.fm algoritmas

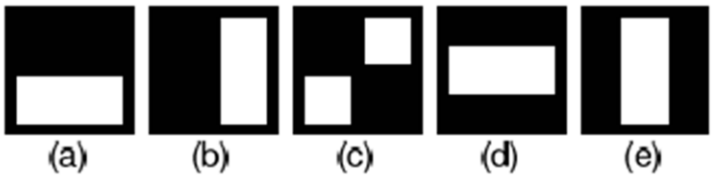
Last.fm naudoja *Computer Vision for Music Identification* algoritmą atpažinti muzikinį kūrinį[LAS11]. Šis algoritmas susideda iš keleto žingsnių. Pirma, jis kaip ir *Shazham* algoritmas, vieno matmens garso signalą paverčia į trijų matmenų paveikslėlį, tai yra spektogramą, taip išvengiama mažų nuokrypių, susidariusių dėl muzikos įrašymo skirtumo. Spektograma generuojama pasirenkant SFFT algoritmą[LAS11].

Toliau algoritmas turi penkis šablonus - filtrus. Juos pritaikydamas keičia spektogramą. Šie šablonai, gali keisti savo plotį ir aukštį atsižvelgiant į analizuojamą spektogramą.



4 paveikslėlis. [KEL05] a) natūralus garsas b) garsas apdorotas SFFT c) garsas apdorotas algoritmo

Algoritmo autoriai pažymi, kad egzistuoja apie dvidešimt penki tūkstančiai galimų nedalomų šablonų. Šie šablonai sudaro vektorių, vadinamą *deskriptoriumi*, jis pagreitina ir palengvina dainos radimą. Žinoma, vieno *deskriptoriaus* informacijos neužtenka kūrinio atpažinimui, todėl per kiekvieną vienuolika tūkstantųjų sekundės dalių sugeneruojamas naujas *deskriptorius*. Šie *deskriptoriai* yra generuojami *Adaboost* algoritmu, kuris pritaikomas garso informacijai[KEL05].



5 paveikslėlis. [KEL05] Deskriptoriai pritaikomi spektogramai

Visi *deskriptoriai* saugomi duomenų bazėje. Nors algoritmo kūrėjai iš pradžių manė, kad paieškai optimizuoti geriausiai tinkamas *Locality-Sensitive Hashing*, tolimesni bandymai parodė, kad sukurtiems *deskriptoriams* užtenka paprastos *hash* lentelės.

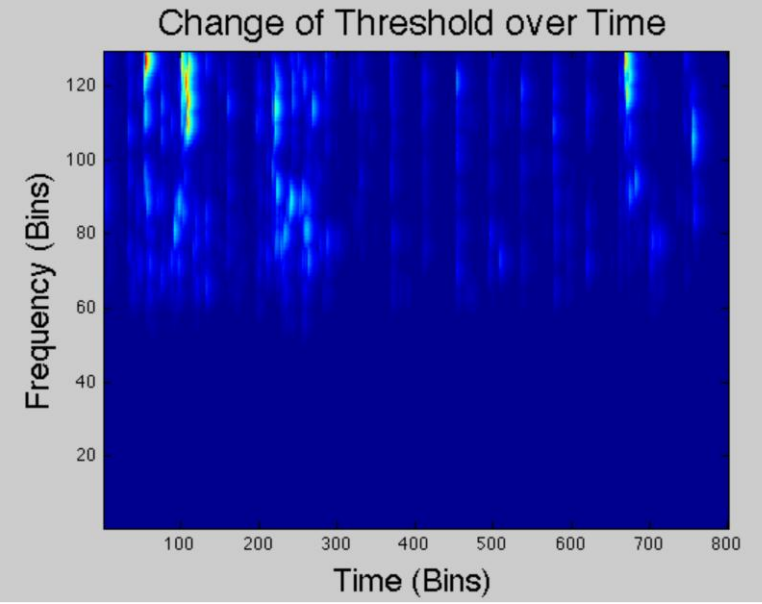
2.2.8 ECE algoritmas

2009 m. *ECE* komanda pasiūlė naują algoritmą muzikos kūrinių paieškai. Šis algoritmas buvo įkvėptas *Shazam* algoritmo pavyzdžiu. Algoritmo kūrėjai pažymi, kad jie naudojasi *Shazam* idėja muzikos indeksavimą vykdyti pasitelkiant svarbiuosius taškus, tik jų lyginimą ir paiešką atlieka kitokiu būdu nei *Shazam*.[RIC09]

Kaip ir *Shazam* algoritmas, *ECE* algoritmas pirmu žingsniu nuskaito muzikos kūrinio failą. Šis failas konvertuojamas į 8000 garso vienetų (*samples*) per sekundę formatą. Tai paspartina tolesnes

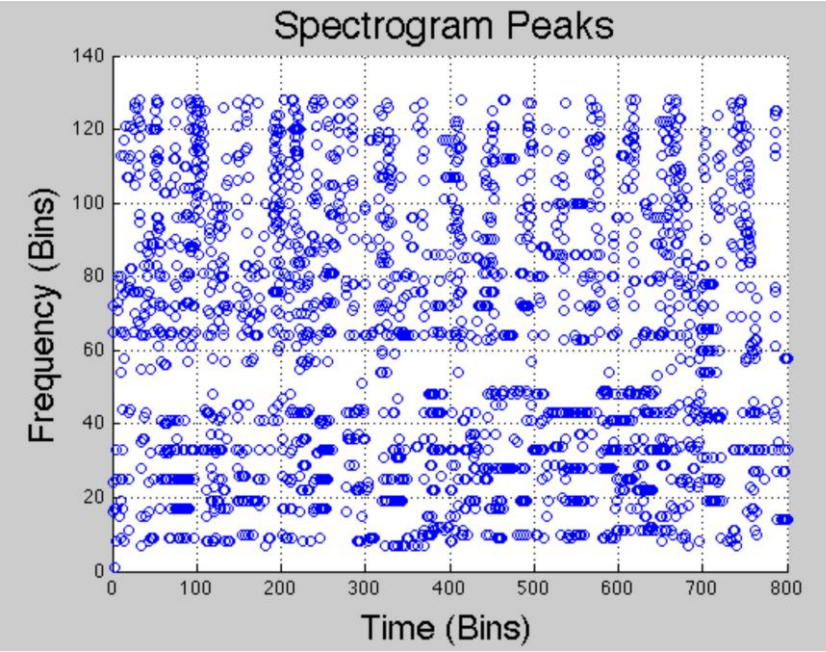
operacijas su muzikos kūrinio duomenimis. Taip pat algoritmas pritaiko aukštų dažnių (high-pass) filtrą. Autoriai pažymi, kad didesnė tikimybė, jog aukšti dažniai išliks įrašytoje užklausoje. [RIC09]

Filtru apdoroti duomenys skaidomi į atkarpas ir transformuojami – taip gaunama spektograma. Kiekvienoje tokioje atkarpoje yra ieškoma lokalių maksimumų. Pirmojoje atkarpoje surandami 5 lokalūs maksimumai. Kitoms atkarpoms pritaikomas Gauso filtras (*Gaussian curve*). Visiems atkarpų dažniams pritaikomas skirtingas Gauso filtras, skirtas atskirti taškus, esančius arti vienas kito.[RIC09] Taip iš kiekvienos atkarpos pasirenkami lokalieji maksimumai ir pridedami prie muzikos kūrinio atspaudo (*fingerprint*).



6 paveikslėlis. Filtro kitimas laike[RIC09]

Šio filtro pagalba randami tie svarbieji taškai, kurie ir sudaro dainos atspaudą (*fingerprint*)



7 paveikslėlis. Ypatingieji taškai [RIC09]

Ši informacija yra saugoma duomenų bazėje. Kitas žingsnis, kurį atlieka algoritmas, tai svarbiųjų taškų informacijos perkėlimas į matricą, kurios dydis toks pat, kaip ir viso muzikos kūrinio spektogramos. Laukuose, kur egzistuoja svarbusis taškas, įrašomas 1, o visuose kituose įrašomas 0.[RIC09] Taigi, lyginant muzikos kūrinį su pateikta užklausa, turime dvi matricas su reikšmėmis 0 ir 1. Kitu žingsniu šių matricų dydis yra suvienodinamas išplečiant matricą nuliais. Toliau šioms matricoms pritaikoma greitoji Furie transformacija (FFT). Transformacijos panaudojimas leidžia surasti lyginamųjų kūrinių maksimalią koreliaciją. Pabaigoje algoritmas suranda rezultato maksimumą ir laiko jį galutiniu panašumo požymiu.

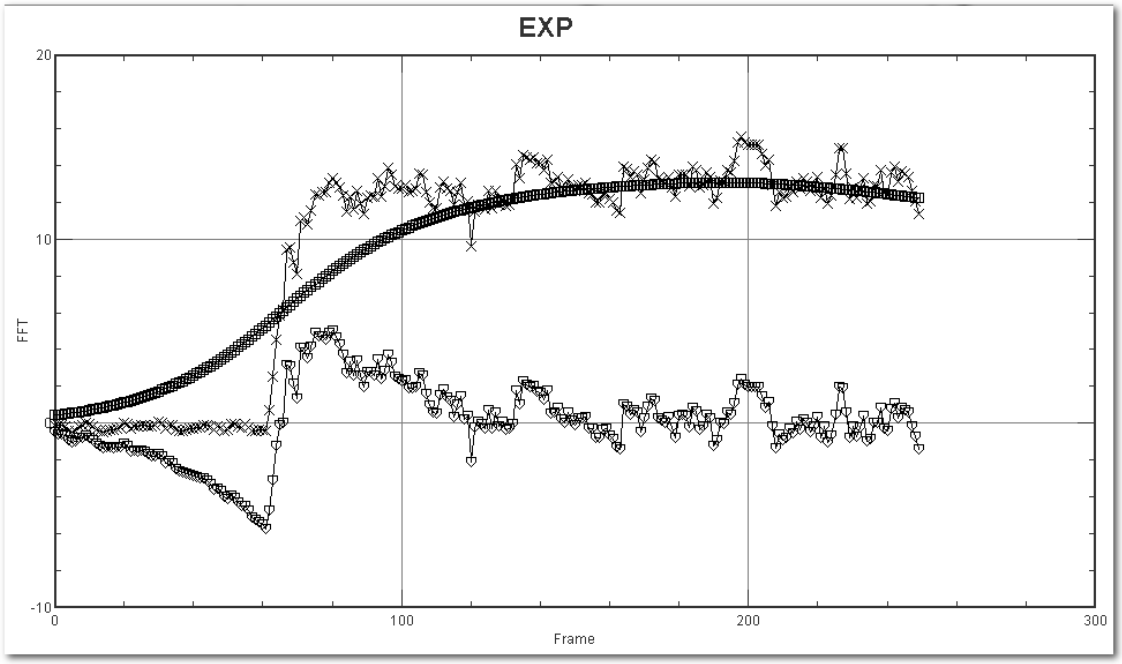
2.2.9 Ivonos Šurpickos algoritmas

Kitas algoritmas, kurį nagrinėsiu, yra Ivonos Šurpickos algoritmas, pasiūlytas bakalauriniame darbe [IVO11]. Šį algoritmą apžvelgsiu, nes jo rezultatus lyginu su savojo algoritmo rezultatais, be to, šio algoritmo principai panašūs į mano pasiūlytą algoritmą.

Pasiūlytas algoritmas, kaip ir kiti *fingerprint* algoritmai, muzikos kūrinių paiešką atlieka naudojant muzikos atspaudus. Toliau išnagrinėsiu algoritmo žingsnius detaliau.

Pirmuoju žingsniu, kaip ir kiti algoritmai, pasiūlytas algoritmas nuskaito muzikos įrašo duomenis. Šie duomenys konvertuojami į 11025 kadrus per sekundę (FPS) [IVO11]. Kitu žingsniu naudojantis *FFT* generuojama spektograma. Tolesni žingsniai skiriasi nuo standartinių. Kad būtų sumažintas operuojamų duomenų kiekis ir pasiektas didesnis tikslumas, algoritmas spektogramą

padalina į vienodo pločio dažnio juostas ir kiekvienai jų apskaičiuoja reikšmių vidurkius kiekvienam laiko momentui. Galiausiai kiekviena tokia dažnių juosta filtruojama glodžiu eksponentiniu filtru. Filtras naudojamas norint normalizuoti reikšmes. Toliau algoritmas apskaičiuoja skirtumą tarp suvidurkintos reikšmės ir reikšmės gautos filtruojant eksponentiniu filtru. [IVO11]



8 paveikslėlis. Grafikas vaizduoja gautas reikšmes. X – signalas, kvadratas – filtruota reikšmė, trikampis – skirtumas tarp reikšmės ir filtruotos reikšmės. [IVO11]

Apdorojus duomenis, algoritmas kuria muzikos kūrinio atspaudą. Algoritmas naudojasi supaprastintu AHS (*arithmetic harmonic sphericity measure*) metrikos skaičiavimu. Tai funkcija, kuri įvertiną ryšį tarp duomenų bazėje esančios koreliacinės matricos ir užklausos koreliacinės matricos. Muzikos kūrinys yra padalijamas į segmentus ir kiekvienam segmentui apskaičiuojama koreliacinė matrica. Kūrinio atspaudą sudaro segmentams apskaičiuotos normalizuotos koreliacinės matricos.

Palyginti normalizuotas matricas algoritmas naudoja L_1 metrikos atstumo tarp matricos elementų reikšmių suma. Toliau algoritmas naudoja *brute force* techniką ir lygina kiekvieną užklausos koreliacinę matricą su duomenų bazėje esančia matrica. [IVO11]

Paskutinis algoritmo žingsnis – tai panašumo tarp užklausos ir duomenų bazėje esančio kūrinio įvertinimas. Kiekvienam kūriniui apskaičiuojama poatspaudžių atstumų matrica.

Panašumo įvertis yra iš vieneto atimtas mažiausia panašumų matricos įstrižainės elementų suma. [IVO11]

2.3 Algoritmų palyginimas

Daugumos nagrinėtų algoritmų detalus palyginimas negali būti atliktas dėl informacijos stokos. Nagrinėti algoritmai naudojami komerciniams tikslams, todėl jų aprašymuose nėra detaliai atskleisti visi veikimo principai. Shazam algoritmas ieško lokalių maksimumų, bet nenurodo, kiek jų prireikia algoritmui ir kur jie ieškomi.

Kita problema iškyla lyginant algoritmus. Lyginamų algoritmų duomenų bazės nėra vienodos, todėl negalima tiksliai nustatyti jų efektyvumo vienodomis sąlygomis. Nagrinėtas Ivonos Šurpickos algoritmas bus palygintas su mano įgyvendintu algoritmu, nes pavyko gauti tokia pat duomenų bazę, kokia buvo naudota Ivonos Šurpickos algoritmo testavimui.

3 Praktinis muzikos paieškos įgyvendinimas

Kitoje tiriamojo mokslinio darbo dalyje sieksiu įgyvendinti savo algoritmą muzikos kūriniams indeksuoti ir jų paieškai atlikti. Sukursiu du atskirus algoritmus, siekdamas suderinti atpažinimo greitį ir atpažinimo kokybę.

Algoritmui įgyvendinti pasirinkau *Java* programavimo kalbą dėl jos universalumo.– šia programavimo kalba parašytas programas galima naudoti ne tik Windows operacinės sistemos aplinkoje. Taip pat *Java* leidžia rašyti programas, kurios gali būti naudojamos mobiliuosiuose telefonuose ar planšetiniuose kompiuteriuose. Be to, *Java* programavimo kalbos bendruomenė yra labai plati. Tai leidžia greitai spręsti iškilusias problemas, susirasti papildomą biblioteką, kai reikia atlikti nestandartinius veiksmus išvengiant papildomo darbo.

Išanalizavus surinktus duomenis apie kitus algoritmus, abiejų algoritmų įgyvendinimą suskirsčiau į dažniausiai pasikartojančias dalis:

- Muzikos kūrinio nuskaitymas
- Spektogramos generavimas
- Muzikos kūrinio atspaudo radimas
- *Atspaudų* lyginimas
- Parametrų tikslinimas geresniems rezultatams gauti

Pirmiausia, darbo eigoje aprašysiu pirmąjį algoritmą, o po to, pritaikius pakeitimus, bus gautas antrasis algoritmas.

3.1 Muzikos kūrinio nuskaitymas

Pirmasis žingsnis įgyvendinant algoritmą - muzikos failo nuskaitymas. Algoritmas nuskaito *WAV* formato failus. Šių failų formatą konvertuoju į PCM (*Pulse-code modulation)* formatą. Konvertavimui naudoju *WawReader Java* klasę iš A. Basčio paskaitų konspektų.[Bio12] Šis formatas pavaizduoja muzikos kūrinį sveikųjų skaičių eilute, kur kiekvienas skaičius rodo garso stiprumą.[PCM10]

Apibrėžimas. FPS (*Frames per second*) – kadrai per sekundę.

Įgyvendinta programa nuskaito kūrinius tik 8000 *FPS*. Ši reikšmė pasirinkta ne atsitiktinai. Tokia reikšmę siūloma naudoti *ECE* algoritme [RIC09]. Pirmoji problema, su kuria susiduriama,

yra *stereo* garsas. Šiame darbe muzikos kūriniams, kurie turėjo *stereo* garsą, buvo taikoma strategija imti tik vieną iš garso takelių. Ši strategija pasirinkta todėl, kad dauguma užklausų, pateikiamų algoritmui, įrašytos diktofonu ir yra įrašytos *mono* formatu. Tolesniame darbe įgyvendintas algoritmas naudoja nuskaitytus *PCM* duomenis.

3.2 Spektogramos generavimas

Kitas algoritmo žingsnis – spektogramos generavimas. Prieš spektogramos generavimą algoritmas apdoroja duomenis.

Pirma, duomenys nuskaitomi ne tiesiogiai, bet naudojantis *Hanning* langu[HAN12]. Šis metodas pasirinktas norint sumažinti duomenų praradimą skaičiuojant *Fourier* transformacijas. *Hanning* lango formulė naudojama:

$$s_n = 0,5 * (1 – \cos((2 * Pi * n)/N-1).$$

Nuskaitytiems muzikos duomenims, kitame žingsnyje pritaikiau *High Preemphasis* filtrą.

$$s_n = s_{n-1} – s_{n-2} .$$

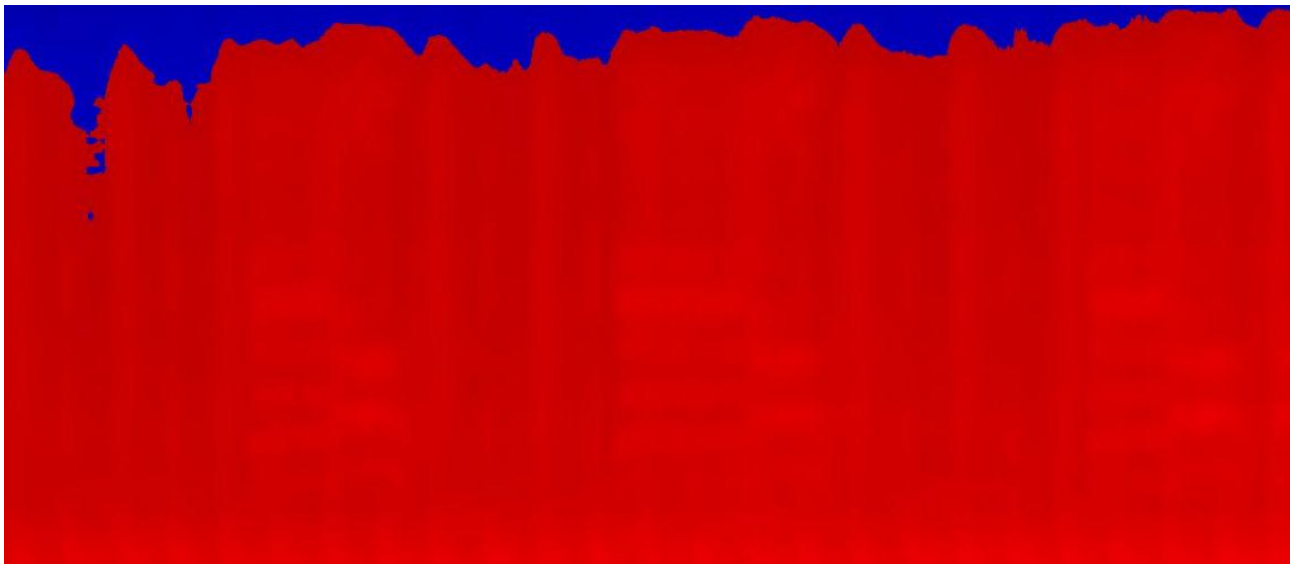
[SPE11]. S žymi nuskaitomą muzikos garso vienetą. Šis filtras išryškina aukštuosius garso dažnius. *ECE* algoritme [RIC09] patariama išryškinti aukštuosius garso dažnius, nes yra didesnė tikimybė, kad jie išliks nepakitę paveikus garso duomenis triukšmu. Šį filtrą pasirinkau dėl jo greito veikimo ir mažo skaičiaus aritmetinių veiksmų. Filtrui pritaikyti nereikia jokių papildomų veiksmų, duomenys filtruojami tiesiogiai juos skaitant.

Toliau algoritmas generuoja spektogramą. Kadangi muzikos failas nėra mažas (dažnai užima nuo trijų iki aštuonių megabaitų), duomenis suskaldau po *FPS* šimtąją dalį, tai suteikia geresnę skiriamąją gebą spektogramai, vaizdas matomas aiškesnis. Norint pavaizduoti spektogramą, šias duomenų dalis reikia apdoroti naudojant *Fourier* transformaciją. Ji suteiks informacijos apie kiekvieno garso dažnio stiprumą tam tikru momentu. Savo įgyvendintoje programoje naudoju greitąją *Fourier* transformaciją (*FFT).* Pati *Fourier* transformacija nėra įgyvendinta algoritme. Aš naudoju *Apache Math* biblioteką[APC12]. Tolesniam darbui algoritmas naudoja tik pusę gautų duomenų, kiti duomenys tėra veidrodinis atvaizdis. Gautas reikšmes algoritmas apdoroja vidurkinimo operacija.

Apibrėžimas. Vidurkinimas – tai matricos taško reikšmė, apskaičiavus sumos vidurkį taško kaimynų matricoje.

Ši operacija skirta sušvelninti atsiradusį triukšmą įrašant kūrinį. Vidurkinimas atliekamas naudojantis taško aplinkos vidurkiu, tai yra, sumuojamos visos reikšmės matricoje esančios arti

taško ir randamas sumos vidurkis, kuris ir priskiriamas taško reikšmei. Algoritme pasirinkta naudoti penkių kaimynų į visas taško puses aplinką. Atlikus šią transformaciją, belieka pavaizduoti informaciją grafiko pavidalu. Šiam tikslui pasinaudojau *Java AWT* biblioteka.



9 paveikslėlis. Sugeneruota ir suvidurkinta spektograma.

3.3 Požymių išrinkimas iš dainos

Trečiojoje darbo dalyje įgyvendinsiu muzikos kūrinio indeksavimą - požymių išskyrimą. Taip pat sudarysiu muzikos kūrinio raktą.

Algoritmo kūrinio požymius sudaro keturios matricos. Šios matricos paeiliui pavaizduoja pirmuosius *Teiloro* eilutės [TEI12] narius. Pirmoji matrica gaunama iš spektogramos matricos pasirenkant tik kas antrą dažnio eilutę. Taip bus sumažintas tolesnių skaičiavimų skaičius. Spektogramos matrica retinama tik dabar todėl, kad tai galėjo pakenkti vidurkinimo metu. Toliau rekursyviai apskaičiuoju tris matricas visiems matricos nariams pritaikydamas algoritmą:

$$m_n[t][f] = m_{n-1}[t + langas][f] - m_{n-1}[t - langas][f],$$

m - žymi matrica, t – laiko kintamasis, f - dažnio kintamasis. Kiekvienai matricai gauti langas pasirenkamas tokiu būdu: 1, 2 ir 3. Po atliktų skaičiavimų, turime keturias matricas.

Kitu žingsniu algoritmas skaičiuoja rangą šioms keturioms matricoms.

Apibrėžimas. Rangas – skaičius, kuris parodo, kiek yra didesnių taškų, esančių šalia pasirinkto matricos taško.

Rangui apskaičiuot naudoju algoritmą:

```
for (daznis = 0; daznis < dazniuSkaicius; daznis++)
        for (laikas = rank; laikas < maksimalusLaikas - rank; laikas++)
```

```
            rangas = 0;
        for (langoLaikas = laikas - rank; langoLaikas < (laikas + rank); langoLaikas++)
            if (matrica[langoLaikas][daznis] > matrica[laikas][daznis])
                rangas ++;
                ranguMatrica[laikas-rank][daznis] = rangas.
```

Šis algoritmas paeiliui perbėga visus apskaičiuojamus matricos elementus. Kiekvienam elementui laiko atžvilgiu apskaičiuojama, kiek elementų yra mažesnių už pasirinktą elementą rango kintamojo aplinkoje. Įgyvendintoje aplikacijoje naudojau rango kintamojo reikšmę – 50.

Atlikus rango skaičiavimą visoms matricoms, turime keturias naujas matricas, saugančias rangus. Kitame žingsnyje šias matricas *binarizuosime*.

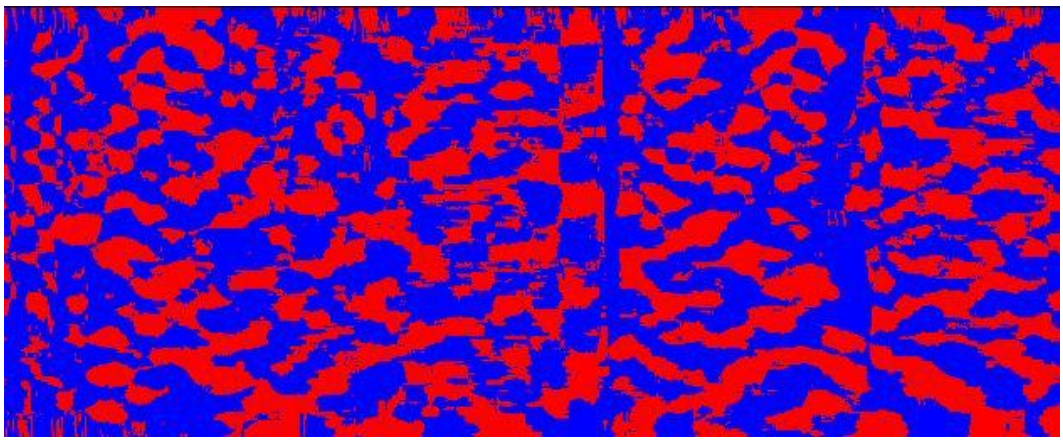
Apibrėžimas. *Binarizavimas* - matricų narių konvertavimas į 0 ir 1. Kaip ir rango skaičiavimas, *binarizavimas* atliekamas pasitelkus taško aplinką laiko atžvilgiu.

Binarizavimui naudoju algoritmą:

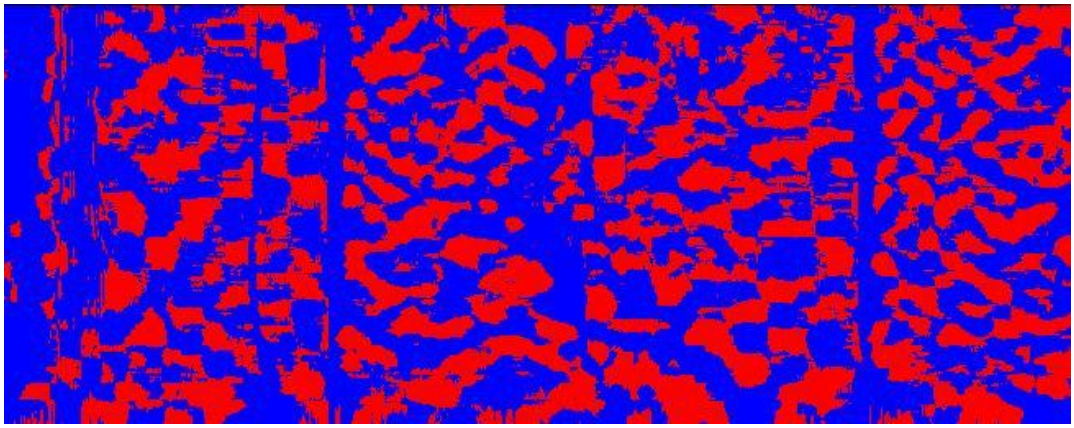
```
for (laikas = 0; laikas < maksimalusLaikas; laikas++)
    for (daznis = 0; daznis < maksimalusDaznis; daznis++)
            skaicius = 0;
    for (daznioKintamasis = dažnis - aplinka; daznioKintamasis < (daznis + aplinka);
            daznioKintamasis++)
  if (matrica[laikas][daznioKintamasis]> matrica[laikas][daznis])
    skaicius ++;
    if (skaicius > aplinka)
            binarizuotaMatrica[laikas][daznis] = 1.
```

Šis algoritmas – tai supaprastinta versija algoritmo, naudojamo įgyvendintoje aplikacijoje. Čia nėra pavaizduoti atvejai, kada skaičiavimai atsiduria matricų pabaigose. Algoritmas perbėga visas rangų matricos reikšmes. Kiekvienai reikšmei apskaičiuojame, kiek yra didesnių reikšmių už pasirinktą tašką. Jei tų reikšmių yra mažiau už naudojamą aplinką, *binarizuotai* matricai priskiriamas vienetas, kitu atveju - nulis. Įgyvendintoje programoje aplinkos kintamąjį pasirinkau – 32.

Derinimo režimu paleista programa atspausdina kiekvieną gautą *binarizuotą* matricą



10 paveikslėlis. Originalaus kūrinio pirmoji binarizuota matrica.



11 paveikslėlis. Užklausos pirmoji binarizuota matrica.

Pavaizduotos pirmosios *binarizuotos* matricos atkarpos originaliam kūriniui ir įrašui. Raudona spalva žymi vienetus, o mėlyna – nulius. Šiuo atveju galime pastebėti jų panašumą. Apskaičiavus kiekvienai rangų matricai *binarizuotą* matricą, turime keturias matricas, kurios ir sudaro dainos požymius.

3.4 Atspaudo generavimas

Iš keturių *binarizuotų* matricų algoritmas generuoja atspaudą (*fingerprint*). Algoritme muzikos kūrinio atspaudas susideda iš 16 bitų įrašų, gautų iš *binarizuotų* matricų. Kiekvienas atspaudo įrašas gaunamas pritaikius funkciją:

raktas[t][f] = bin0[t][f-4] | (bin1[t][f-4] << 1) | (bin2[t][f-4] << 2) | (bin3[t][f-4] << 3) | (bin0[t][f - 3] << 4) | (bin1[t][f -3] << 5) | (bin2[t][f -3] << 6) | (bin3[t][f -3] << 7) | (bin0[t][f - 2] << 8) | (bin1[t][f -2] << 9) | (bin2[t][f -2] << 10) | (bin3[t][f - 2] << 11) | bin0[t][f - 1] << 12) | (bin1[t][f - 1] << 13) | (bin2[t][f - 1] << 14)| (bin3[t][f - 1] << 15)).

Funkcija pritaikoma visiems laiko momentams ir kas ketvirtam dažniui. Kas ketvirtam dažniui pritaikoma todėl, kad sudaromas raktas savyje laiko keturių dažnių informaciją. Galiausiai kiekvieną muzikos kūrinį atvaizduoja matrica, kuri savyje turi 16 bitų įrašus.

3.5 Kūrinių lyginimas

Paskutinis algoritmo žingsnis – kūrinių lyginimas. Algoritmas skaičiuoja dviejų matricų panašumą (koreliaciją) laiko atžvilgiu. Panašumo skaičiavimas skirstytas į du žingsnius. Pirmuoju žingsniu apskaičiuojama panašumų eilutė.

Apibrėžimas. Panašumų eilutė – sveikųjų skaičių masyvas, kurio kiekvienas narys nurodo panašumą atitinkamu laiko momentu.

Šios eilutės skaičiavimui taikau *brute force* metodą, kurio pagalba bandoma atrasti laiko poslinkį įrašyto kūrinio. Tai yra, bandoma atrasti, kuriuo laiko momentu kūrinys buvo pradėtas įrašyti – tuo metu panašumas bus didžiausias.

Panašumų eilutės skaičiavimas pradedamas skaičiuojant duomenų bazėje esančio kūrinio kiekvienos laiko eilutės panašumą su užklausos laiko eilutėmis.

```
for (orginalusLaikas = orginalausYrasoIlgis-1; orginalusLaikas >= 0; orginalusLaikas--)
for (uzklausosLaikas = uzklausoYrasoIlgis-1; uzklausosLaikas >= 0; uzklausosLaikas--)
    panasumas = laikoPanasumas(uzklausa[uzklausosLaikas],orginalus[orginalusLaikas),
```

laikoPanasumas funkcijai paduodami du masyvai - laiko eilutės. Kadangi abiejų kūrinių dažnių skaičius vienodas, abu masyvai yra vienodo ilgio. Kiekvienas masyvo elementas atitinka vieną sugeneruotą raktą. Tada algoritmas paeiliui lygina abiejų masyvų raktus. Raktų panašumas nėra skaičiuojamas tiesiogiai lyginant gautą šešiolikos bitų skaičių. Atliekamas tikslesnis skaičiavimas – skaičiuojama, kiek bendrų skaitmenų lyginamuose skaičiuose sutampa. Kadangi iš viso egzistuoja 2^{16} variantų, galima iš anksto sudaryti lentelę, kuri iš pateiktų dviejų skaičių pasakytų bendrų bitų skaičių.

Pavyzdys: Lyginami du raktai 16 ir 1. Šiuos skaičius pavertus į dvejetainį formatą gauname 0000 0000 0001 0000 ir 0000 0000 0000 0001, todėl lyginat šių skaičių panašumą bitų lygmenyje, jis bus lygus keturiolikai. Lyginant 55 ir 110 atitinkamai 0000 0000 0011 0111 ir 0000 0000 0110 1110, šių skaičių panašumas bus devyni.

Kiekvieno laiko masyvo palyginimo rezultatas sumuojamas į panašumų eilutę šiuo algoritmu:

```
panasumuEilute[ ( orginalusLaikas + uzklausoYrasoIlgis – 1 – uzklausosLaikas )mod orginalausYrasoIlgis ] += panašumas.
```

Suskaičiavus visus panašumus gaunama užpildoma panašumų eilė.

12 paveikslėlis. Panašumų eilutė esant sutapimui

13 paveikslėlis. Panašumų eilutė, kai nėra sutapimo

Paskutinis algoritmo žingsnis. Eilės didžiausio elemento radimas. Kaip matome iš grafikų pavyzdžių, didžiausias elementas išskiria iš visų aplinkiniu. Prieš elemento paiešką visus eilės elementus padalinu iš užklausos laiko. Taip sumažėja skaičiavimams naudojami skaičiai. Rastas didžiausias elementas ir bus laikomas kūrinių panašumu.

3.6 Greitasis algoritmas

Aprašytas algoritmas, kaip vėliau parodė tyrimų rezultatai, atpažįsta muzikos kūrinius gerai. Deja, didžiausias jo trūkumas – veikimo greitis. Kadangi atliekami *brute force* perrinkimo metodai, algoritmo skaičiavimas užtrunka gana ilgai. Siekdamas geresnių rezultatų, sukūriau antrąjį algoritmą, kuris išsprendžia greičio problemą, bet kartu nepraranda palyginimo tikslumo. *Greitasis* algoritmas išsiskiria tik dviem paskutiniais etapais, kuriuos ir aprašysiu.

3.7 Greitųjų raktų sudarymas

Pirmoji problema, kurią sprendžiame greitajame algoritme, yra didelis raktų skaičius ir jų užimama vieta atmintyje. Šešiolikos bitų raktus pakeičiame aštuonių bitų raktais – tokiu būdu du kartus sumažinami duomenys reikalingi kūrinio atspaudui. Žinoma, sumažinus raktus, pasikeičia ir raktų sudarymo algoritmas, bet požymių išrinkimas iš dainos išlieka toks pats. Algoritmas keičiasi prasidėjus raktų sudarymui. Prarasta informacija kompensuojama sudarant raktus. Raktų sudarymui pasirenkami nariai išsidėstę įstrižaine. Tikimasi, kad pasirinkus ne iš eilės einančius narius, bus gautos unikalios reikšmės. Toliau pateikiu pakeistą raktų sudarymo algoritmą:

```
raktas[t][f] = bin0[t][f] | (bin1[t][f + lenght/8] << 1) | (bin2[t][f*2 + lenght/8] << 2)| (bin3[t][f*3 + lenght/8] << 3)| (bin0[t][f*4 + lenght/8] << 4) | (bin1[t][f*5 + lenght/8] << 5) | (bin2[t][f*6 + lenght/8] << 6)| (bin3[t][f*7 + lenght/8] << 7).
```

Sudaryta saugojimo struktūra skiriasi nuo senojo algoritmo. Sudaryti raktai saugojami trijų matmenų masyve.

```
muzikos_atspaudas [pozicijaDaznyje][raktas][laikas] .
```

Pirmasis matmuo – dažnių skaičius padalintas iš aštuonių. Skaičius padalintas iš aštuonių, nes raktai sudaromi tik kas aštuntą dažnį. Antrasis matmuo – 256 reikšmių masyvas, kuris atitinka gautąjį raktą. Paskutinis masyvas – laikas, kada esamu dažniu aptiktas esamas raktas.

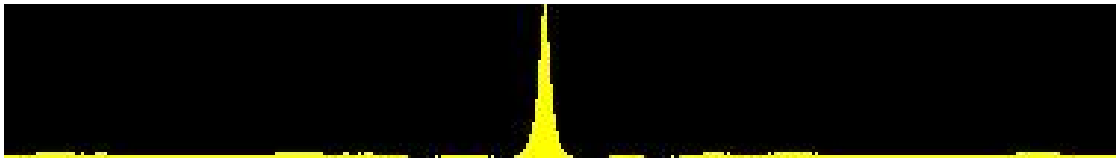
3.8 Greitasis kūrinių lyginimas

Kadangi pasikeitė kūrinio atspaudas (*fingerprint*), pasikeičia ir jų lyginimo technika. Lyginimo technika senajame algoritme atlikdavo daug veiksmų. Todėl ji taip pat yra optimizuota greitajame algoritme.

```
for ( vieta = 0; vieta < uzklausoMatrica.ilgis; vieta++ )
    for ( raktas = 0; raktas < 256; raktas++ )
for (laikasOrginalo = 0; laikasOrginalo < orginaloMatrica[vieta][raktas].ilgis; laikasOrginalo ++ )
for (laikasUzklausos = 0; laikasUzklausos < uzklausosIlgis[vieta][raktas]. ilgis; laikasUzklausos ++ )
similaraty[ ( orginaloIlgis - 1 - pilnas[vieta][raktas][laikasOrginalo] + fragmentas[vieta][raktas][laikasUzklausos] ) mod orginaloIlgis]++.
```

Iš algoritmo matome, kad *brute force* algoritmo naudojimas yra sumažintas. Algoritmas perbėga tik tas vietas, kurių skaičius nėra didesnis už 512 ir raktų variantus, kurie, kaip matome iš algoritmo, teturi 256 reikšmes. Toliau gaunamos tik tos reikšmės, kada pasirodė atitinkamas raktas. Čia jau nėra naudojamas *brute force* metodas. Galima pastebėti, kad raktų lyginimas nebevyksta bitų lygmenyje, kadangi tiesiogiai kreipiamasi į masyvo elementą, esantį atitinkamame rakte. Tokiu būdu išvengiama papildomų operacijų raktų lyginimui, kartu galima pastebėti, jog tai panašu į *hash* tipo lenteles.

Galiausiai panašumų eilutės sumavimas išlieka toks pats, kaip ir senajame algoritme, tik dabar mes žinome laiką, kuriuo reikia sumuoti elementus.



14 paveikslėlis. Panašumų eilutė esant sutapimui

15 paveikslėlis. Panašumų eilutė, kai nėra sutapimo

Pavaizduotose panašumo eilutėse galima pastebėti, kad netikrų rezultatų dydis ženkliai sumažėjo, o teigiamas rezultatas išryškėjo labiau. Tai lėmė griežtesnis atrankos kriterijus – nėra lyginami raktai, pasikliaujama tik tiksliu sutapimu, sumuojant panašumus. Kaip parodys eksperimentai, tai pablogina atpažinimą, kai užklausa yra prastos kokybės, bet pagerina, kai užklausa yra geros kokybės.

Galiausiai, kaip ir senojo algoritmo atveju, iš panašumų eilutės išrenkamas didžiausias skaičius, kuris ir yra laikomas kūrinio panašumu.

3.9 Algoritmų techninis palyginimas

Toliau pateiksiu abiejų algoritmų techninių skirtumų lentelę.

	Senasis algoritmas	Greitasis algoritmas
Vieno požymio dydis (rakto)	16 bitų	8 bitai
Rakto struktūra	raktas [laikas][daznis]	raktas [pozicija][raktas][laikas]
Požymių lyginimas	Lyginamas kiekvienas bitas	Lyginamas skaičius
Atspaudų lyginimas	*Brute force*	*Brute force – hash* algoritmas
Vienai paieškai atlikti reikalingas palyginimų skaičius	Originalaus kūrinio laiko eilučių kiekis * užklausos laiko kiekis * kūrinio dažnis	256 * kūrinio garso dažnis /16

1 lentelė. Lėtojo ir greitojo algoritmo palyginimas.

Iš lentelės matome, kad didžiausias greitojo algoritmo pranašumas prieš senąjį algoritmą yra perrinkimų skaičius reikalingas vienam raktui palyginti.

4 Eksperimentai

Šioje dalyje atliksiu eksperimentus, kuriuose nagrinėsiu abiejų algoritmų atpažinimo galimybes, palyginsiu jų greitį ir raktų užimamą vietą. Algoritmui įvertinti naudosiu šiuos rodiklius:

FAR (*false acceptance rate*) – žymi tikimybė, kad bus nustatyta, jog kūrinys yra tas kūrinys, kuris iš tiesų nėra.

FRR (*false rejection rate*) - žymi tikimybę, kad bus nustatyta, jog kūrinys nėra tas kūrinys, kuris iš tiesų yra.

EER (*Equal Error Rate*) – žymi skaičių, kai FAR = FRR.

Šiuos įvertinimus grafiškai pavaizduosiu DET kreivėmis (*Detection error trade-off*)[Mar97]. Kreivės abscisių ašis žymi FAR reikšmes, o ordinačių ašis – FRR.

Vienus iš tyrimo duomenų paėmiau iš Ivonos Šurpickos bakalauro darbo „Muzikos kūrinių atpažinimas“.[IVO11] Iš 30 originalių kūrinių bus lyginami 9 įrašyti kūriniai.

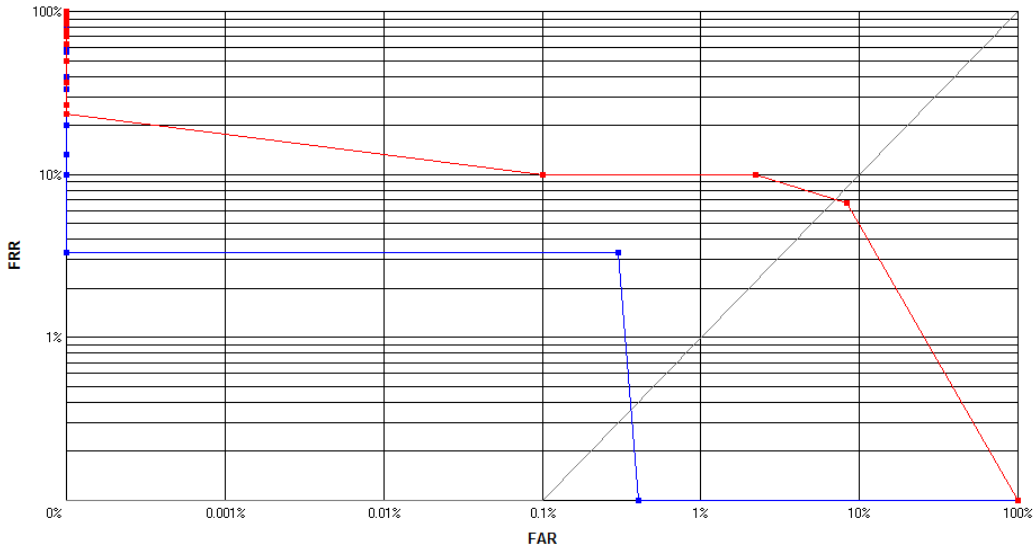
4.1 Algoritmo nustatymų reikšmė atpažinimui

Šiame eksperimente nagrinėsiu, kaip algoritmo parametrai veikia greitąjį algoritmą. Kaip ir ankstesniuose tyrimuose, naudosiu Ivonos Šurpickos duomenų bazę. Kūrinius dalinsiu trisdešimties sekundžių atkarpomis.

Pirmuoju bandymu patikrinsiu aukštų dažnių filtro naudojimo efektyvumą. Skaidysiu dainas 30 sekundžių atkarpomis ir lyginsiu su rezultatu, kuris gautas algoritmu be pakeitimų.

Algoritmas	**EER**	**0 FAR**	**0 FRR**
Su filtru	7,27 %	23,3 %	8,4%
Be filtro	0,39 %	3,33 %	0,3%

2 lentelė. Algoritmo rezultatai su ir be filtro.



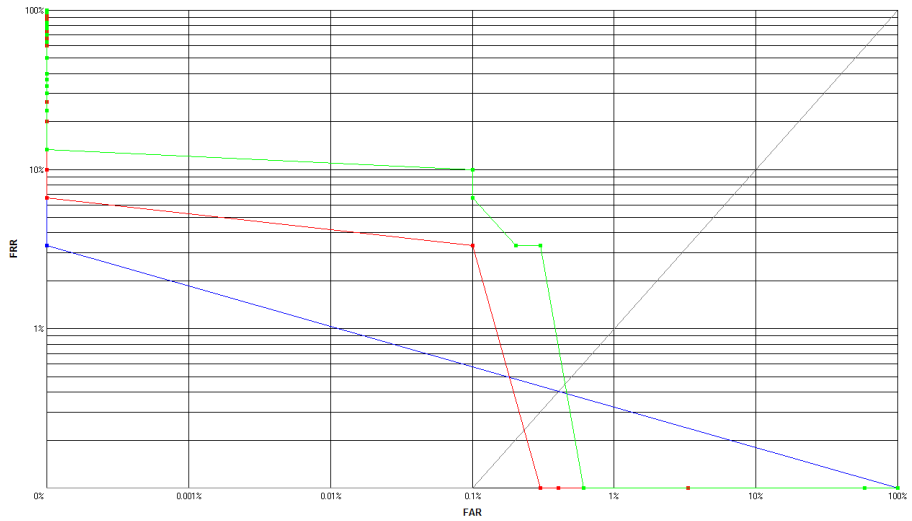
16 paveikslėlis. DET kreivė. Raudona spalva - su filtru, mėlyna - be filtro.

Iš atlikto tyrimo matome, kad aukštų dažnių filtras suprastina rezultatus eksperimento metu, todėl tolesniuose eksperimentuose nebebus naudojamas.

Kitas eksperimentas tiria spektogramos generavimo metu pasirenkamos taško aplinkos dydžio priklausomybę atpažinimui. Eksperimentui naudojami tie patys duomenys.

Taško aplinka	**EER**	**0 FAR**	**0 FRR**
5	0,39 %	3,33 %	0,3%
10	0,29 %	6,67 %	0,1%
15	0,55 %	13,3 %	0,3%

3 lentelė. Taško aplinkos priklausomybė algoritmui.



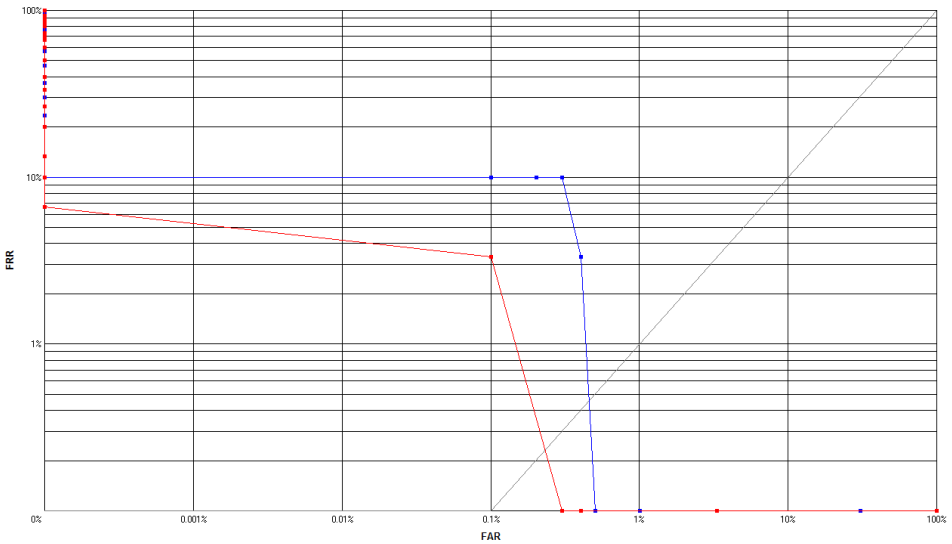
17 paveikslėlis. DET kreivė. Mėlyna - 5 taško aplinka, žalia - 15, raudona - 10.

Šis eksperimentas parodo, kad optimali taško aplinka yra 10, kuri ir bus naudojama tolesniuose eksperimentuose. Taip pat galime pastebėti, kad didesnis suvidurkinimas panaikina išskirtinius kūrinio požymius.

Kitas eksperimentas – tai naudojamų matricų pakeitimas raktų generavimui. Šiame eksperimente pamėginsiu nenaudoti keturių matricų, o pasiremti tik viena.

Algoritmas	**EER**	**0 FAR**	**0 FRR**
Keturios matricos	0,29 %	6,67 %	0,1%
Viena matrica	0,49 %	10,00 %	2%

4 lentelė.Raktų generavimo algoritmų rezultatai.



18 paveikslėlis. DET kreivė. Raudona - 4 matricos, mėlyna- viena.

Šis algoritmas parodo, kad papildomos trys matricos pagerina skaičiavimą.

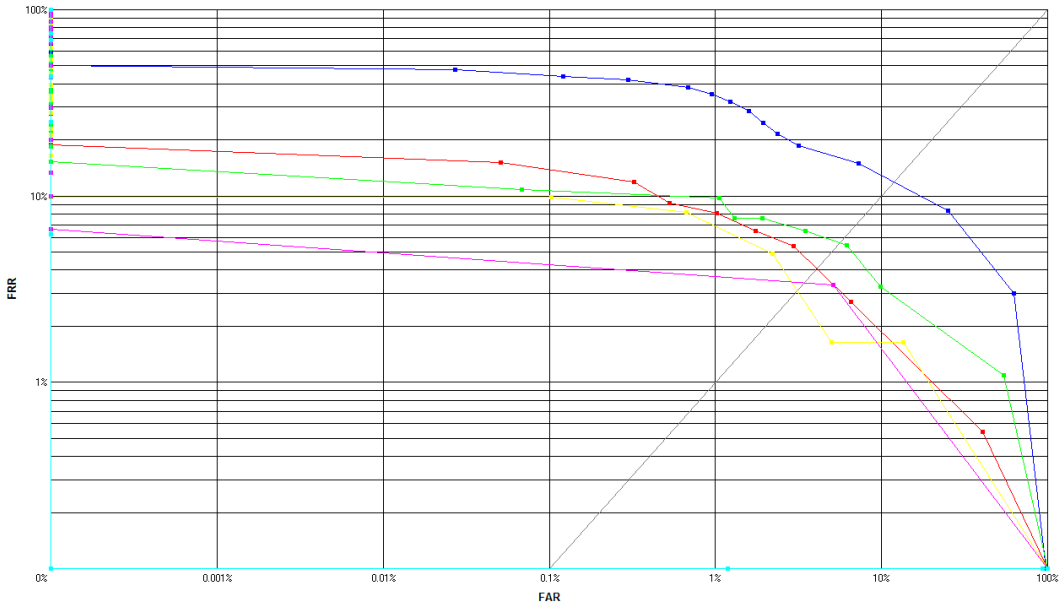
4.2 Atpažinimo kokybės priklausomybė nuo kūrinio trukmės

Pirmajame eksperimente tikrinsiu, kokia yra abiejų algoritmų atpažinimo priklausomybė nuo užklausos trukmės. Šiam tikslui naudosiu telefonu įrašytus muzikos kūrinius iš Ivonos Šurpickos duomenų bazės. Šiuos kūrinius dalinsiu į 2, 5, 10, 15, 30 ir 45s. Kuo mažesnėmis atkarpomis imamas kūrinys, tuo daugiau atliekama palyginimų. Pirmieji rezultatai naudojant greitąjį algoritmą:

Fragmento ilgis	**EER**	**0 FAR**	**0 FRR**
2s	12,9 %	50 %	62,8%
5s	4,35 %	18,9 %	40,7%

10s	5,64 %	15,2 %	54,7%
15s	3,46 %	9,84 %	13,5%
30s	4,04 %	6,67 %	5,1%
45s	0,00 %	0,00 %	0%

5 lentelė. Greitojo algoritmo priklausomybė nuo laiko.

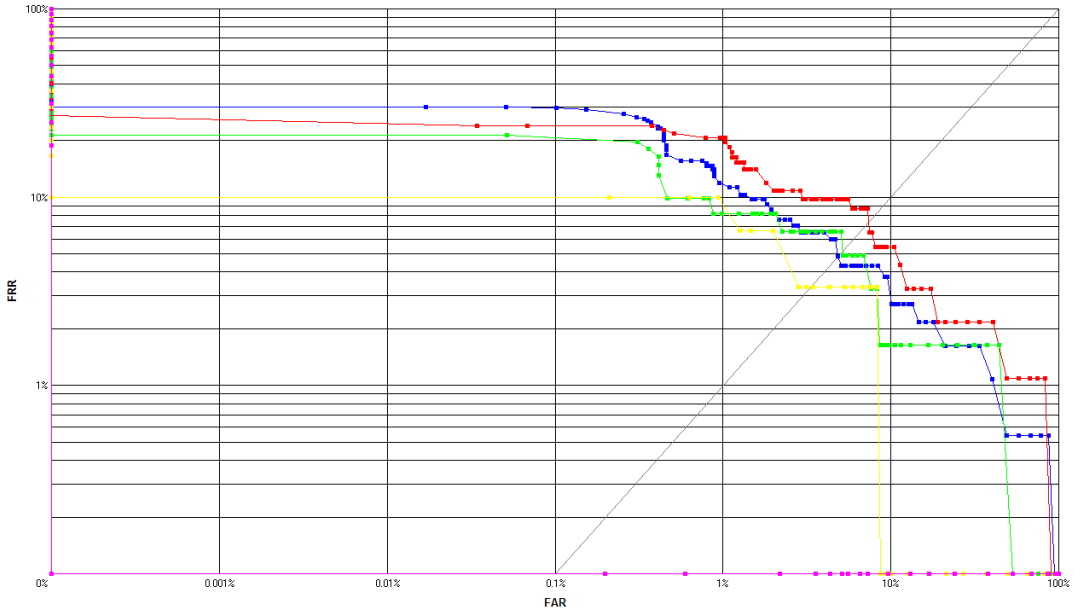


19 paveikslėlis. DET kreivė. Mėlyna - 2s, žalia - 10, raudona - 5, geltona -15, rožinė - 30, žydra -45 sekundės.

Iš pirmosios lentelės matome, kad ir penkių sekundžių įrašo užtenka atpažinti kūrinį. Taip pat, matome, kad 45 sekundės eksperimento atveju garantuoja teisingą rezultatą. Kitoje lentelėje pavaizduosiu lėtojo algoritmo rezultatus. Dėl algoritmo įgyvendinimo aplinkybių nėra palygintos dviejų sekundžių atkarpos.

Fragmento ilgis	**EER**	**0 FAR**	**0 FRR**
5s	4,48 %	30,3 %	87%
10s	7,38 %	27,2 %	82%
15s	5,14 %	21,3 %	44%
30s	3,33 %	10 %	8,3%
45s	0,00 %	0,00 %	0%

6 lentelė. Lėtojo algoritmo priklausomybė nuo laiko.



20 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona -30, rožinė - 45 sekundės.

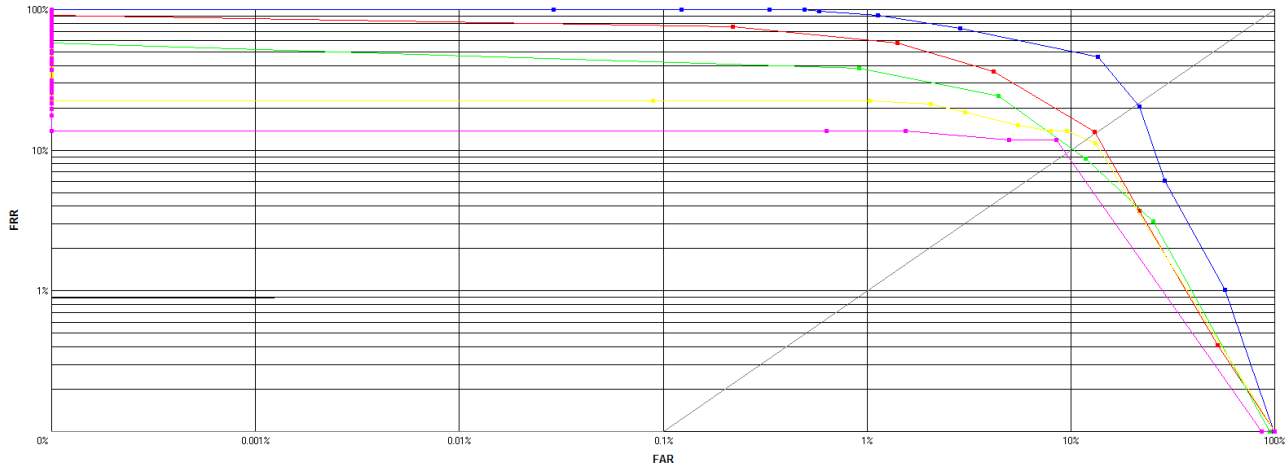
Kaip matome iš šeštosios lentelės, greitasis algoritmas nėra blogesnis už pirmąjį algoritmą. Abiems algoritmams užtenka keturiasdešimt penkių sekundžių trukmės kūrinio, kad jis būtų atpažintas teisingai.

4.3 Jautrumas triukšmui

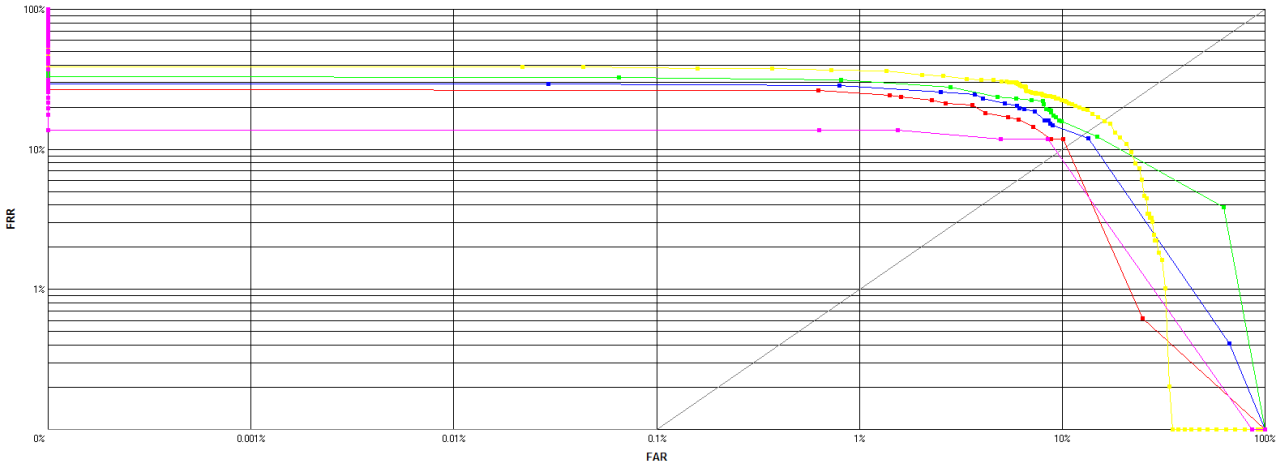
Kito eksperimento tikslas – patikrinti algoritmo atsparumą triukšmui. Šiam tikslui bus naudojama 40 dainų duomenų bazė. Šios dainos pasitelkus *DJ Audio Editor* [DJE12] paverčiamos į sugadintas dainas – dainoms pridedamas triukšmas. Tyrime naudosiu trijų lygių: 25, 35 ir 45 decibelų triukšmo paveikimą. Triukšmas bus generuojamas Gauso metodu. Bus lyginami devyni kūriniai, juos, kaip ir pirmajame eksperimente dalinsiu atkarpomis, kurios bus lyginamos. Pirmiausia eksperimentas bus atliktas su 25 decibelų triukšmo lygiu ir greituoju algoritmu. Lentelėje pavaizduosiu dvi reikšmes, vieną – su pritaikomu aukštų dažnių filtru, kitą – be filtro.

Fragmento ilgis	**EER**		**0 FAR**		**0 FRR**	
5s	13,4%	13,3 %	32,9 %	74,8%	62,5 %	66,2%
10s	12,5 %	13,3 %	29,2 %	90,5 %	67%	52,4%
15s	11,1 %	10,80 %	26,9 %	57,50 %	24,8%	24,5%
30s	12%	11,2%	22,5%	25%	13,2%	79,8%
45s	11,3 %	11,8%	13,7%	13,7%	8,5%	85%

7 lentelė. Greitojo algoritmo priklausomybė nuo 25dB triukšmo.



21 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas naudojant filtrą.



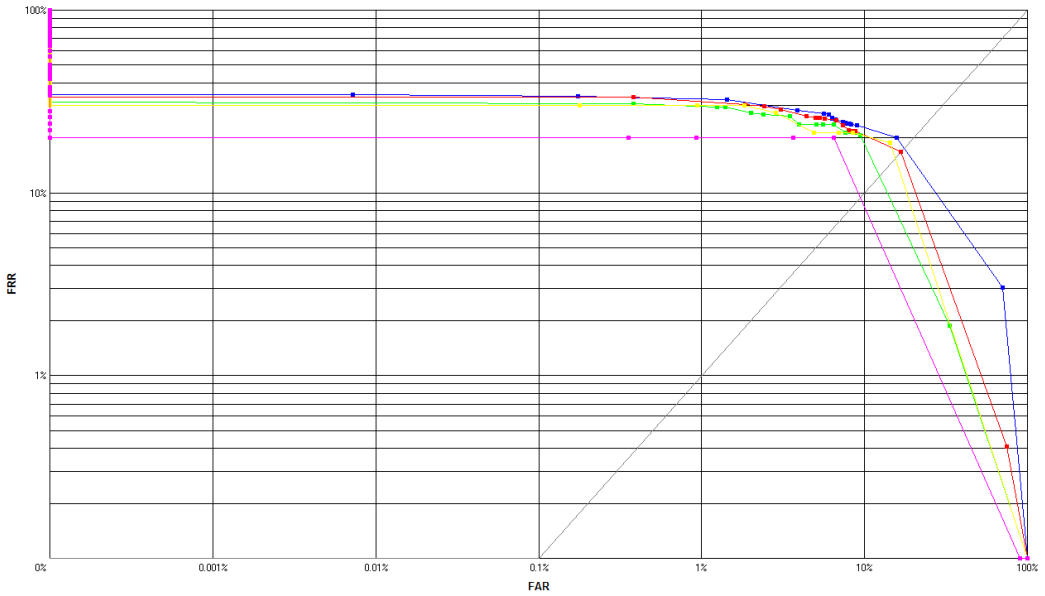
22 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas be filtro..

Kaip matome iš pirmo tyrimo, algoritmas susidoroja su 25 decibelų triukšmu. Deja, kaip matome, įrašo trukmė neįtakoja EER ir keturiasdešimt penkių sekundžių trukmės įrašo neužtenka idealiam kūrinio atpažinimui.

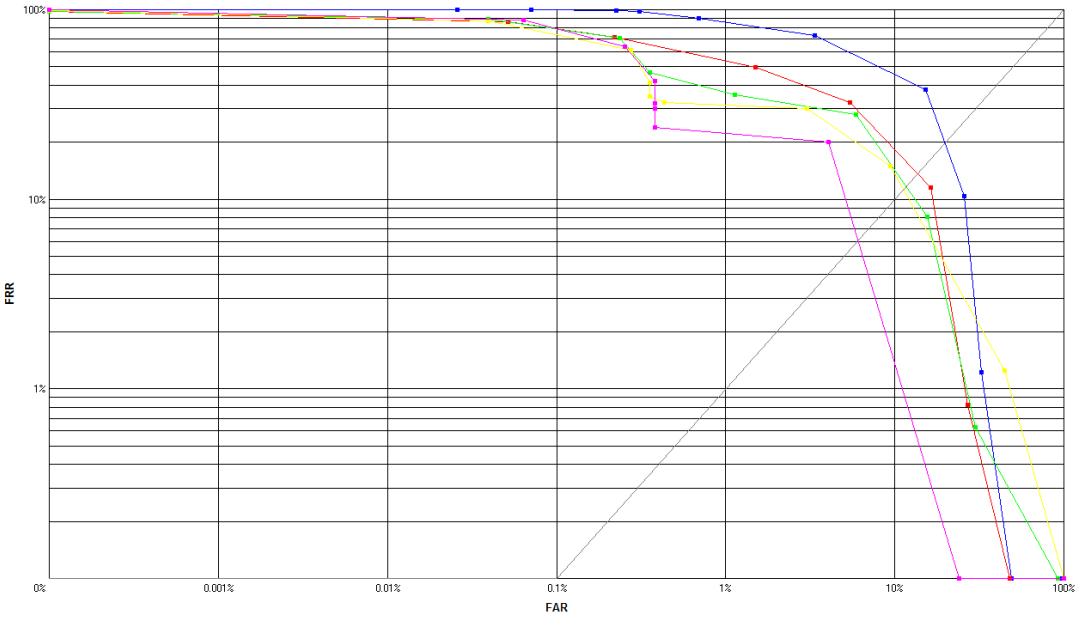
Kitas eksperimentas atliktas su 35 decibelų triukšmu naudojant greitąjį algoritmą.

Fragmento ilgis	**EER**		**0 FAR**		**0 FRR**	
5s	19,1 %	21,5%	34,6 %	100%	70,3%	32,4%
10s	16,8%	14,7%	33,6 %	98,4%	74,5%	26,8%
15s	15,7%	13,1%	31,3 %	98,1%	33,3%	29,9%
30s	18%	13,4%	30%	98,8%	14,4%	44,5%
45s	17,4%	12%	20%	100%	6,5%	4,1%

8 lentelė. Greitojo algoritmo priklausomybė nuo 35dB triukšmo.



23 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas naudojant filtrą.



24 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas be filtro.

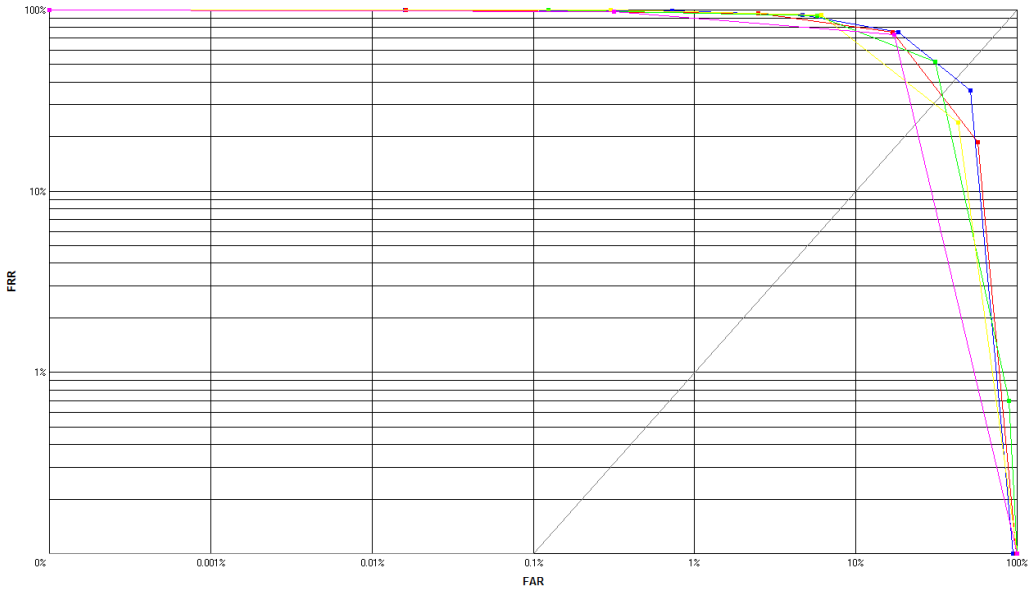
Iš šio eksperimento matome, kad padidinus triukšmo lygį, atpažinimo kokybė prastėja. Kokybę padeda pagerinti ilgesnės trukmės įrašai, deja, ir 45 sekundžių įrašo trukmės neužtenka geram atpažinimui.

Kaip matome iš atliktų trijų eksperimentų, filtro panaudojimas ir pagerina, ir pablogina rezultatus. Trumpesnės trukmės įrašams aukštų dažnių filtras pagerina atpažinimą.

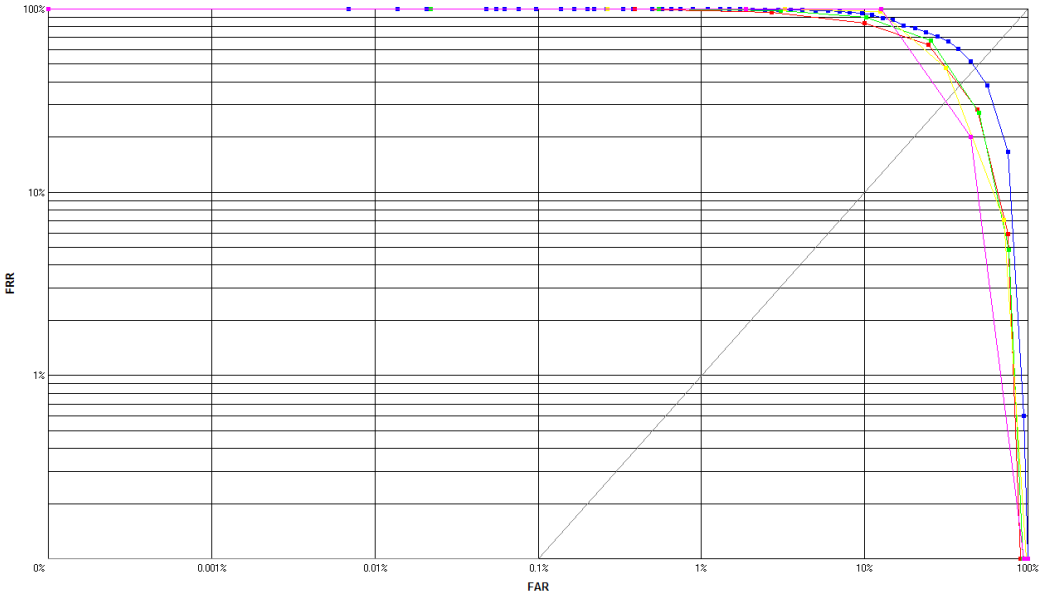
Toliau atliekamas eksperimentas su 45 decibelų triukšmu naudojant greitąjį algoritmą.

Fragmento ilgis	EER		0 FAR		0 FRR	
5s	44,4 %	48,1%	100%	100%	51,5%	94,4%
10s	41,2%	40,5%	100%	100%	57,2%	75,2%
15s	42,2 %	41,3%	100%	100%	88,9%	76,3%
30s	36,6 %	39,5%	100%	100%	43%	70.8%
45s	47%	37,7%	100 %	100%	17,3%	44,8%

9 lentelė. Greitojo algoritmo priklausomybė nuo 45dB triukšmo



25 paveikslėlis DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas su filtru.



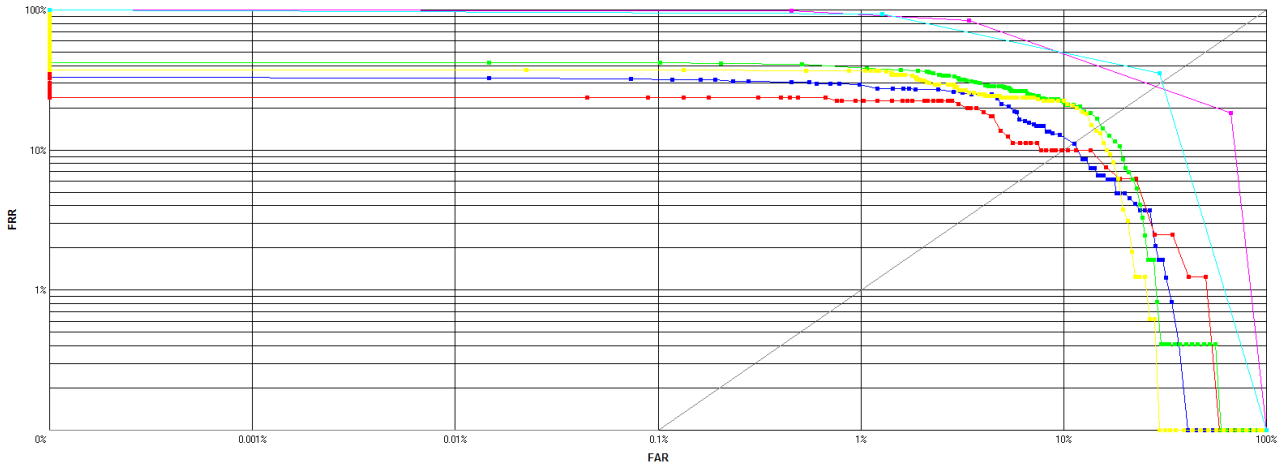
26 paveikslėlis. DET kreivė. Mėlyna - 5, raudona - 10, žalia - 15, geltona - 30, rožinė - 45 sekundės. Tyrimas be filtro.

Kaip matome, greitasis algoritmas su 45 decibelų garsu nesugeba teisingai atpažinti kūrinių.

Toliau atliekame tyrimus su lėtuoju algoritmu. Naudojame tą pačią duomenų bazę kaip ir greitojo algoritmo tyrime.

Triukšmo lygis	**Fragmento ilgis**	**EER**	**0 FAR**	**0 FRR**
25 dB	10s	11,2%	32,9%	30,1%
25 dB	30s	10%	23,8%	50,1%
35 dB	10s	15,2 %	42,20 %	100 %
35 dB	30s	14,2 %	37,50 %	100 %
45 dB	10s	39,4 %	100,00 %	100 %
45 dB	30s	36,00 %	100,00 %	100 %

10 lentelė. Lėtojo algoritmo priklausomybė nuo triukšmo.

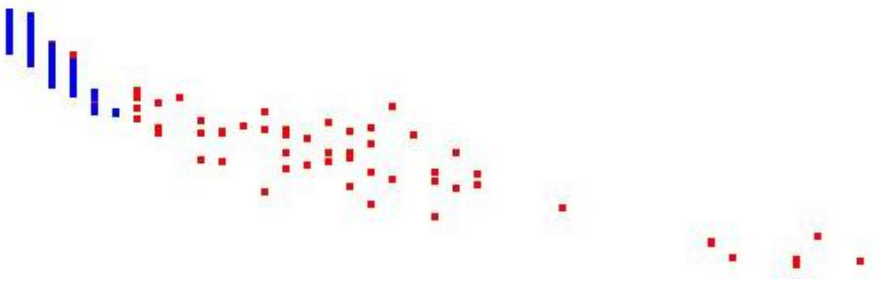


27 paveikslėlis. DET kreivė. Mėlyna - 25dB. 10s., raudona - 25dB. 30s., žalia - 35dB 10s, geltona - 35dB 30s., rožinė - 45dB 10s. , žydra - 45dB, 30s.

Iš eksperimentų rezultatų matyti, kad su didesniu triukšmu sėkmingiau susidoroja lėtasis algoritmas. Kita vertus, skirtumas tarp lėtojo ir greitojo algoritmų nėra didelis, todėl atsižvelgiant į situaciją galima pasirinkti vieną ar kitą.

4.4 Algoritmų apjungimo galimybė

Kitas tyrimas taikomas abiems algoritmams. Jis parodo algoritmų sujungimo galimybę. Šiuo tikslu parašiau pagalbinę programą, kuri geba pavaizduoti dviejų *DET* failų panašumo įvertinimus viename grafe Abscisių ašyje atidedami vieno *DET* failo panašumų įvertinimai, o ordinačių ašyje atidedami kito *DET* failo panašumų įverčiai tam pačiam palyginimui. Abiejų failų panašumo įverčiai yra normalizuoti [0 .. 600] atkarpoje.



28 paveikslėlis. 15 sekundžių lėto ir greito algoritmo panašumo įverčių pasiskirstymas. Mėlyna spalva – apsišaukėliai, raudona – teisingi.

29 paveikslėlis. 30 sekundžių lėto ir greito algoritmo panašumo įverčių pasiskirstymas

Kaip matome iš šių dviejų grafikų, sutampantiems kūriniams (raudona spalva) įverčiai pasiskirsto nutolę nuo nesutampančių kūrinių (mėlyna spalva).

Daroma išvadą, jog galime sujungti abu algoritmus siekdami geresnio rezultato.

4.5 Algoritmų techniniai duomenys

Šio eksperimento tikslas – parodyti, kiek vietos sumažėja naudojantis greituoju algoritmu ir koks yra greičio skirtumas. Užklausai pateikiami trisdešimties sekundžių atkarpos kūriniai.

	Pirmasis algoritmas	**Greitasis algoritmas**
Palyginimo greitis	~6s.	~0,5s.
Dainos įkėlimo greitis	~60s	~40s.
Užimamos dainos vieta	~183.159	~90.464

11 lentelė. Algoritmų techninis palyginimas.

Palyginimo greitis rodo dviejų kūrinių palyginimo greitį, tai yra, laiką reikalingą palyginti muzikos kūriniui iš duomenų bazės ir trisdešimties sekundžių užklausos. Iš eksperimento matome, kad algoritmui pavyko ženkliai sumažinti atpažinimo laiką. Be to, užimama duomenų vieta vienai dainai saugoti sumažėjo per pusę. Šis parametras gali būti dar labiau optimizuotas, nes savo įgyvendintoje aplikacijoje naudojau standartinį Java serializavimą.

4.6 Palyginimas su kitais algoritmais

Paskutiniame eksperimente palyginsiu rezultatus gautus su Ivonos Šurpcikos algoritmu [IVO11]. Algoritmus galiu palyginti todėl, kad naudojau tokią pat duomenų bazę, kokia buvo naudota ir Ivonos Šurpcikos algoritmo testavimui. Lyginsiu rezultatus gautus vidutinės kokybės garso įrašui, skirtingo ilgio užklausoms. Lyginsiu greitojo algoritmo rezultatus.

	Greitasis algoritmas	**Ivonos Šurpcikos**
10s.	EER: 5,64 % 0 FAR: 15,20 % 0 FRR: 100,00 %	EER: 22,9%, 0 FAR: 100%, 0 FRR: 91,2%
15s.	EER:3,46 % 0 FAR: 9,84 % 0 FRR: 100,00 %	EER: 16,9% 0 FAR: 100% 0 FRR: 89,3%
30s.	EER:4,04 % 0 FAR: 6,67 % 0 FRR: 100,00 %	EER: 9,1% 0 FAR: 60,6% 0 FRR: 41,5%
40s.	EER:0% 0 FAR: 0% 0 FRR: 0%	EER: 11% 0 FAR: 50% 0 FRR: 22,8%

12 lentelė. Greitojo algoritmo ir Ivonos Šurpcikos algoritmo palyginimas.

Iš rezultatų gautų lyginat algoritmus galima matyti, kad greitojo algoritmo rezultatai trumpesnėms užklausoms yra geresni.

Išvados

Atlikus magistro baigiamąjį darbą, yra įgyvendinti šie užsibrėžti tikslai:

1. Išnagrinėti esami algoritmai ir produktai, skirti muzikos indeksacijai ir paieškai. Atlikta analizė leidžia teigti, kad dauguma algoritmų yra komerciniai, todėl nėra viešai pateikiami algoritmų veikimo įvertinimai ir tikslūs algoritmai. Taip pat nėra prieinama duomenų bazė, kuria šie algoritmai testuojami. Pavyko gauti tik Ivanovos Šupcikos algoritmo ivertinimus ir duomenis testavimui.
2. Praktinėje dalyje pasiūlyti du algoritmai įrodo, kad muzikos atpažinimą galima atlikti naudojantis spektogramos požymiais.
3. Pirmojo algoritmo gautieji eksperimentų rezultatai parodo, kad telefonu įrašytam trisdešimties sekundžių vidutinės kokybės kūriniui gaunama 3% lygios klaidos tikimybė (EER), tai tenkintų daugumą vartotojų.
4. Antrasis algoritmas – pirmojo algoritmo geresnė versija. Tyrimų rezultatai parodė, kad antrasis algoritmas vienos dainos palyginime dvyliką kartų lenkia pirmąjį greičiu. Vidutinės kokybės ir trisdešimties sekundžių įrašo užklausai pasiekta 4% lygios klaidos tikimybė.
5. Atliktas sujungimo tyrimas parodė, kad yra galimybė sujungti abiejų algoritmų rezultatus norint gauti geresnį atpažinimo rodiklį.

Įgyvendintas antrasis algoritmas gali būti patobulintas, optimizuojant duomenų saugojimą. Taip pat algoritmas gali būti įgyvendintas daugiau nei vienos gijos aplinkoje. Tokiu būdu būtų paspartintas jo greitis. Algoritmas gali būti pritaikytas praktiškai:

1. Muzikos indeksacijai. Algoritmas neatsižvelgdamas į failo pavadinimą, gali ieškoti dublikatų kietajame diske.
2. Muzikos paieškos aplikacijai. Mobiliuose įrenginiuose įgyvendinta kliento aplikacija galėtų kreiptis į serverį ir pateikti atsakymą apie skambančią dainą.

Literatūros sąrašas

[RJL05] R.J. Lorimer, „MP3: Play MP3s from Java with Javazoom“, URL: http://www.javalobby.org/java/forums/t18465.html, 2005-04-27.

[MID07] „Midomi review: search by“, URL: singinghttp://www.consumingexperience.com/2007/09/midomi-review-search-by-singing.html, 2007-09-13.

[SOU11] http://www.soundhound.com.

[SHA11] „Shazam (service)“, URL: http://en.wikipedia.org/wiki/Shazam_%28service%29

[VAL10] Valeriy Lobaryev, Gene Sokolov, Alexandr Gordeyev, „Sloud Query-by-Humming Search Music Engine“, URL: http://www.sloud.com/download/Sloud_QBH_Search_Music.pdf

[EUG10] Eugene Weinstein, „Query By Humming:A Survey“, URL: http://cs.nyu.edu/~eugenew/publications/humming-summary.pdf

[PED02] Pedro Cano, Eloi Batlle, Ton Kalker, Jaap Haitsma, „A Review of Algorithms for Audio Fingerprinting“, URL: http://mtg.upf.edu/files/publications/MMSP-2002-pcano.pdf

[WAN03] Avery Li-Chun Wang, „An Industrial-Strength Audio Search Algorithm“, URL: http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf,

[SUR11] Nicolae Surdu, „How does Shazam work to recognize a song ?“, URL: http://www.soyoucode.com/2011/how-does-shazam-recognize-song, 2011-06-20.

[JAC09] Bryan Jacobs, „How Shazam Works“, URL: http://laplacian.wordpress.com/2009/01/10/how-shazam-works/, 2009-01-01

[LAS11] http://www.last.fm/

[HAI02] Jaap Haitsma, Ton Kalker, „A Highly Robust Audio Fingerprinting System“, URL: http://ismir2002.ismir.net/proceedings/02-FP04-2.pdf,

[KEL05] Yan Ke1, Derek Hoiem, Rahul Sukthankar, „Computer Vision for Music Identification“, URL: http://www.cs.cmu.edu/~yke/musicretrieval/cvpr2005-mr.pdf, 2005.

[SPE11] Spectogram, http://en.wikipedia.org/wiki/Spectrogram

[RIC09] Rice University ELEC 301, „ELEC 301 Projects Fall 2009“, 2009

[PCM10] http://www.digitalpreservation.gov/formats/fdd/fdd000016.shtml,

[SPE11] http://mi.eng.cam.ac.uk/~ajr/SpeechAnalysis/node43.html,

[IVO11] Ivona Šurpicka, „ Muzikos kūrinių atpažinimas “, 2011,

[Mar97] Martin, A. F, „The DET Curve in assessment of detection task performance“, 1997

[Bio12] https://kedras.mif.vu.lt/bastys/academic/ATE/biometrika/BioBalsas.pdf

[HAN12] http://mathworld.wolfram.com/HanningFunction.html

[APC12] http://commons.apache.org/math

[TEI12] http://mathworld.wolfram.com/TaylorSeries.html

[DJE12] http://www.program4pc.com/dj_editor.html#page=page-1