Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Pirkó Balázs

AKKORDFELISMERÉS ÉS GÉPI IMPROVIZÁCIÓ

KONZULENS
Dr. Bank Balázs

BUDAPEST, 2016

1 Tartalom

HALLGATÓI NYILATKOZAT

Alulírott **Pirkó Balázs**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 12. 18.

...…….

Pirkó Balázs

Összefoglaló

A dolgozatomnak két fő célja volt. Az első céljának zenefelvételek automatikus akkordfelismerését választottam, míg a másodiknak az ehhez szervesen kapcsolódó, számítógéppel való zenei improvizációra képes algoritmus implementálását tűztem ki.

Az akkord a nyugati zene sajátossága. Egyszerűen megfogalmazva azt mondhatjuk, hogy az akkord több hang egyidejű megszólalása, ami együtt egyfajta harmóniát ad. Egy gyakorlott zenész képes lehet arra, hogy hallás alapján felismerje az egyes akkordokat. Kevésbé gyakorlott zenészeknek - akik nem rendelkeznek még ezzel a képességgel - nagy segítség lehet egy akkordfelismerő szoftver. Egy ilyen alkalmazással könnyebben meg lehet tanulni olyan zenéket, amihez nem áll rendelkezésre kotta, vagy ez alapján akár zenei transzkripciót, feldolgozást is könnyebben lehet készíteni. Ezenkívül egy jól működő akkordfelismerő program magába foglalja annak a lehetőségét, hogy nagyméretű zenei adatbázist készítsünk, mely alkalmas lehet további gépi tanításra például zenei stílus felismerés vagy zenei improvizáció készítés kapcsán.

Az akkordfelismerés egy ma is folyamatosan fejlődő kutatási terület, teljesen egzakt algoritmus, mely tévesztés nélkül működne, még nem készült. Az eddig elkészült alkalmazásokban többféle módszerrel próbálkoztak, ilyen pl. a Rejtett Markov-modelles megközelítés, mintamegfeleltetés, neurális hálózatokkal való tanítás. Ezen módszerek az emberi döntéshozatalt próbálják matematikailag modellezni, jellemzően valószínűségi módszerek alkalmazásával. Mivel az egyszerű jelfeldolgozást alkalmazó módszerek nem tűntek elég hatékonynak, dolgozatomban az akkordfelismerést Rejtett Markov-modelles irányból közelítettem meg. Ehhez a szakirodalomban leírt eszközöket használtam fel, és azt fejlesztettem tovább saját ötleteken alapuló módszerekkel. Az elkészült programot *Matlab* környezetben implementáltam és annak pontosságát egy megfelelően nagyméretű zenei adatbázison teszteltem.

A dolgozatom második részében számítógéppel történő zenei improvizációkészítéssel foglalkoztam. Egy ilyen funkcióra képes szoftver alkalmas lehet a zenei improvizáció oktatására, illetve gyakorlás céljára virtuális zenésztársként is felhasználhatjuk, ha más zenészek éppen nem elérhetőek. Munkám során egy valós és egy nem valós idejű zenei improvizációra képes algoritmust implementáltam, ebből is nagyobb bonyolultsága és személyes érdeklődésem miatt a valós idejű algoritmusra koncentráltam. A nem valós idejű program leginkább csak teszt céljából készült. A valós idejű szoftver elindítása után 10 másodperc alatt meghatározza a mikrofonra érkező zenei jel tempóját, rászinkronizálódik, majd az általa felismert akkordok alapján egyszólamú zenei improvizációt készít. Ebben a programban - a gépi improvizáció igényeit figyelembe véve - egy egyszerűbb akkordfelismerő algoritmust alkalmaztam. Ezenkívül a programban implementáltam egy akkordszekvencia-felismerés funkciót is, amely a felismert akkordkör alapján jóslást ad a következő akkordra, és ez alapján folytatja tovább az improvizációt. A programot C++ nyelven implementáltam a JUCE keretrendszer segítségével. A program forráskódját, a lefordított, és önmagában is működő *exe* kiterjesztésű fájlt és egy négyakkordos szekvenciára a szoftver által készített dallamot mellékletként közzétettem.

Abstract

My thesis has two main aims. The first aim is the chord recognition of recorded music, and the second is making a music improvisation in real-time based on the recognized chords.

Chords are typical building blocks of Western European music. Basically, a chord is when more musical tones sound simultaneously, creating a harmony. A proficient musician is able to recognize chords based on hearing. For less experienced musicians – who are not able to recognize chords by ear – a chord recognizer software could be very helpful. This kind of software can help to learn music easier also in the case when no sheet music is available. It can also support music transcription or adaptation. In addition, a well-functioning chord recognizer program opens the possibility of making a large chord database, which then can be used for machine learning for musical genre recognition or automatic composition/improvisation, to name a few applications.

Chord recognition is still an evolving research area, by the very nature of the problem, exact solutions do not exist. Instead, many different approaches are used in the literature, some are based on pattern matching algorithms, on Hidden Markov-model approach (learning parameters of the model), or machine learning with neural network, and so on. These concepts model human decision mathematically with probabilistic methods. Because the simple signal processing methods first tried did not lead to adequate results, I have chosen the Hidden Markov-model as base of my algorithm. The starting point of the developed algorithm is thus an approach based on the Hidden Markov model published in the literature, which I have improved with own ideas and extensions. I have implemented the chord recognition algorithm in *Matlab* environment, and tested the accuracy of my method on a sufficiently large music database.

In second part of my thesis I focused on creating a music improvisation with computer. Ideally, a software, with such a feature could be used in teaching music improvisation, or as a virtual musician for practicing, if the real musicians is not available. During my work I implemented both a real time and an offline version of the algorithm. I have concentrated on the real time software, due to personal interest. I used the offline program for testing and development purposes. My real time software detects the tempo of the music - recorded by microphone - within 10 seconds. After that the program is synchronizing with the music, and creates monophonic music improvisation based on the detected chords. In this program I have implemented a simpler chord recognition algorithm, but the accuracy of this is enough for music improvisation. Besides that I also implemented a chord sequence recognizer feature, which is able to predict the next chord, based on the recognized chord loop, and continue the improvisation with this information. I have implemented this program in *C++* language within the JUCE framework. In the appendix I share the source code of the program, the executable file, and an example melody made by the real time software.

2 Bevezető

A dolgozatomnak - mint, ahogy a címéből is következtetni lehet- kettős célja volt. Az első céljának zenefelvételek automatikus akkordfelismerését választottam, míg a másodiknak az ehhez szervesen kapcsolódó, számítógéppel való zenei improvizációra képes algoritmus implementálását tűztem ki.

Először tehát az akkordfelismeréssel foglalkoztam. Az akkordfelismerés a *Music Information Retrieval* (MIR) egyik kutatási területe. A MIR interdiszciplináris tudományterület, melynek célja a zenében lévő információk kinyerése. Többféle zenei információt ismerhetünk fel, csakúgy, mint hangnem, tempó, akkordok, vagy akár a konkrét zenei hangokat is, ez utóbbi gépi kottázáshoz, transzkripcióhoz használható.

Az akkord a nyugati zene sajátossága. Egyszerűen megfogalmazva azt mondhatjuk, hogy az akkord több hang egyidejű megszólalása, ami együtt egyfajta harmóniát ad. Egy gyakorlott zenész képes lehet arra, hogy hallás alapján felismerje az egyes akkordokat. Kevésbé gyakorlott zenészeknek - akik nem rendelkeznek még ezzel a képességgel - nagy segítség lehet egy akkordfelismerő szoftver. Egy ilyen alkalmazással könnyebben meg lehet tanulni olyan zenéket, amihez nem áll rendelkezésre kotta, vagy ez alapján akár zenei transzkripciót, feldolgozást is könnyebben lehet készíteni. Egy ilyen program a zenei oktatásban is nagy segítség lehet, például arra, hogy ezáltal jobban megértsük a zenei szerkezeteket, hogy tanulmányozhassuk, és gyorsabban elsajátíthassuk a komponálási módszereket, adott stílusok jellegzetességeit, akkordmeneteit. Ezenkívül egy jól működő akkordfelismerő program magába foglalja annak a lehetőségét, hogy nagyméretű zenei adatbázist készítsünk, mely alkalmas lehet további gépi tanításra például zenei stílus felismerés vagy zenei improvizáció készítés kapcsán.

A dolgozatomban mindenekelőtt azon zeneelméleti alapokat ismertetem, amelyek elengedhetetlenek a továbbiak megértése szempontjából. Itt szó lesz arról, hogy milyen zenei hangok léteznek, hogyan számíthatjuk ki a frekvenciájukat, és hogyan kódolhatjuk őket. Bemutatásra kerül az akkord fogalma, ezen belül részletesen a dúr és a moll hármashangzatokkal foglalkozunk.

Ezek után ismertetésre kerülnek korábbi, az akkordfelismeréssel foglalkozó tudományos munkák. A tradicionális megközelítésű módszeren keresztül elmagyarázom, hogy mely tényezők azok, amelyek megnehezítik a pontos akkordfelismerést. A további módszereket három főbb csoportra osztottam fel: mintamegfeleltetéses, a rejtett Markov-modelles megközelítésű és a neurális hálózatot felhasználó algoritmusok.

A mintamegfeleltetéses algoritmusok közül részletesen ismertetésre kerülnek *Fujisima* és *Lee* algoritmusai. Fujisima Short Time Fourier-transzformációval vizsgálta a zenét, majd ebből készített PCP vektorokat, melynek az eltérését vizsgálta az általa megalkotott akkordadatbázis elemeivel. Lee felhasználta Fujisima munkáját. Az általa bevezetett újdonság az *EPCP* (továbbfejlesztett PCP) vektor használata volt, amely a zenei hangok felharmonikusainak a tulajdonságát használja ki.

A rejtett Markov-modelles (RMM) megközelítést használó algoritmusok közül *Sheh* és *Ellis*, illetve *Bello* és *Pickens* algoritmusait mutatom be. Sheh és Ellis 147-féle akkord felismerésére alkalmas algoritmust fejlesztett. A RMM-ben megfigyelési modellnek egyszerű Gauss-modellt használtak véletlenszerű átlagértékvektorral, és korrelálatlan kovarianciamátrixszal. Bello és Pickens ezt az algoritmust fejlesztette tovább. A zenei jelet a zene tempója alapján osztották fel, és a megfigyelési modellben a véletlenszerű átlagértékvektort, és a teljesen korrelálatlan kovarianciamátrixot cserélték le zeneelméleti tudás alapján inicializáltakra.

Végül a neurális hálózattal történő felismerések közül *Osmalskyj* módszerét mutatom be részletesen. Ebben az algoritmusban Osmalskyj a neurális hálózatot PCP vektorokkal tanítja, és ez alapján készít egy, a 10 leggyakrabban felhasznált hármashangzat felismerésére alkalmas programot.

A szakirodalmat megismerve dolgozatom első fő céljának azt választottam, hogy egy dúr és moll hármashangzatok felismerésére alkalmas algoritmust készítek, mely képes megmondani valós, stúdióban felvett, többhangszeres zenék akkordjait. Több kísérlet után a szakirodalomban megismert irányok közül a rejtett Markov-modelles megközelítést választottam. Ennek megértéséhez a matematika időbeli valószínűségi következtetések témakörével kellett megismerkednem. A RMM egy általános struktúra, amelyben a közvetlenül nem mérhető, időben gyorsan változó tényezőnek az állapotárára adunk valószínűségi becslést valamilyen matematikai módszerrel. Az akkordra tekinthetünk úgy, mint rejtett változó, ami csak dúr és moll hármashangzatokra szűkítve 24 állapotot vehet fel. Az RMM-nél fontos dolog lesz az, hogy a rejtett változóra adott valószínűségeket nem csak az aktuális mérésből számítjuk, hanem figyelembe vesszük azt is, hogy a mi történt a múltban, illetve offline alkalmazások esetén a jövőt is tekintetbe vehetjük. Ezenkívül bemutatom, hogy a modell paraméterei, és a valószínűségek csak kezdeti paraméterek, melyeket az adott megfigyelési szekvenciára vonatkozóan az elvárásmaximalizációs (EM) algoritmussal tudunk iteratív módon pontosabbá tenni. Továbbá ahhoz, hogy megmondhassuk, hogy a megfigyelési szekvenciát milyen legvalószínűbb rejtett változó sorozat okozta, az EM algoritmus elvégzése után használnunk kell a Viterbi-algoritmust is.

Az általam fejlesztet akkordfelismerő algoritmusban tehát RMM-es megközelítést használok. Alapvetően Bello és Pickens munkájából indultam ki. Az egyes mérésekhez tartozó időablakok meghatározásához tempódetektáló algoritmust készítettem, illetve egy olyan algoritmust, amely a felismert tempó alapján szinkronizálódik a zenére. Ennek felhasználásával nagyobb az esély arra, hogy akkordváltásnál van az időablak széle. Az általam használt RMM modell paramétereihez egy, az interneten is megtalálható zenei statisztikát használtam fel, illetve újszerű, a szakirodalomban nem fellelhető módszerrel határozom meg az érzékelő modellt. Ehhez a jelfeldolgozás módszereit alkalmazva olyan zenei leíró vektort készítettem, mely a szakirodalomban használt egyéb zenei leíróknál jobb alapot biztosít a további számításokhoz. Ezenkívül az általam írt akkordfelismerő programot is teszteltem. Tesztjeim során megállapítottam, hogy a rejtett Markov-modelles megközelítés jobb választás, mint a szakirodalomban megismert EPCP vektorral történő mintamegfeleltetős algoritmus, és megfigyeltem, hogy több helyen jobban működik, mint Bello és Pickens algoritmusa. Az akkordfelismerő programomhoz

további tesztek, és további fejlesztés szükséges a magasabb hatásfok eléréséhez, és további értékelésekhez.

A dolgozatom második fő célja gépi improvizáció készítése volt. A számítógéppel valós és nem valós idejű improvizációt is készíthetünk. Egy valós idejű improvizációra képes szoftver alkalmas lehet arra, hogy színesebbé tegye a hangszeres gyakorlást olyan esetekben, amikor a valódi zenésztársak nem érhetők el. Egy jól működő, nem valósidejű improvizációt készítő szoftvert pedig oktatási célokra használhatjuk olyan zenészeknél, akik szeretnék elsajátítani ezt a képességet.

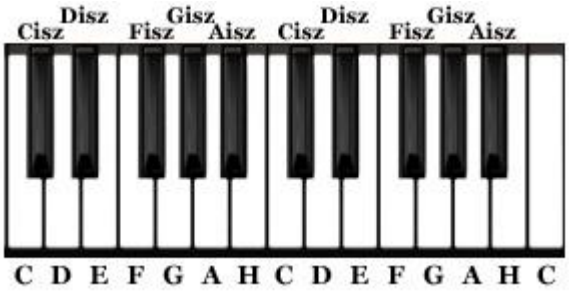
Dolgozatom zenei improvizációról szóló részében először röviden bemutatom az alapokat, majd a szakirodalomban fellelhető megoldások közül az Impro-Visor programot ismertetem, amely a felhasználó által megadott akkordok alapján, valószínűségi nyelvtant felhasználva rövidebb jazzdallamrészleteket képes generálni. Ezek után a saját algoritmusomat mutatom be. Először alapvetően tesztelés céljából készítettem egy Matlab programot, ami a bemeneti audiojelhez akkord és hangnemfelismerés segítségével offline módon generál zenei improvizációt. Az improvizáció készítéséhez véletlenszerű ritmust használok, a hangkészlete pedig az adott időszelethez tartozó akkordhangokból, illetve a zene hangneméből származtatott pentaton hangokból áll. A program valós, stúdióban felvett zenével való tesztelése során bíztató eredményeket adott. Ezek alapján úgy gondoltam, hogy van értelme egy valós idejű megvalósításnak is. A valós idejű szoftvert a *JUCE* keretrendszer segítségével implementáltam *C++* nyelven. A programom a mikrofonon keresztül beérkező audiójelet feldolgozza, és annak alapján egyszólamú zenei improvizációt játszik le. Ebben a programban egy viszonylag egyszerűbb akkordfelismerő algoritmust használok, viszont a tapasztalat azt mutatja, hogy egyszerűbb zenék esetén az ez alapján generált improvizáció hangjai elfogadhatóak. A ritmust itt is véletlenszerűen generálom. Ezt a módszert még mindenképp érdemes lenne továbbfejleszteni, mert így még előfordulhatnak zeneileg furcsa ritmusképletek. A program 10 másodperces gyakorisággal felismeri az éppen lejátszott dal tempóját, majd rászinkronizálódik, illetve olyan esetekben, amikor a lejátszott akkordmenetben talál egy szekvenciát, akkor a következő akkordot az alapján megjósolja. A program megírásában a különösen nagy kihívás a tempóra való szinkronizáció és a valós idejű kritériumok teljesítése.

A dolgozatom megírásakor a hangsúly végül a két nagy részből inkább az akkordfelismerésre tevődött át. Ennek oka egyrészt a gazdagabb szakirodalom, amivel az akkordfelismerés rendelkezik, másrészt pedig a rejtett Markov-modellekhez kapcsolódó mélyebb tudományos ismeretek elsajátítására való késztetés, mely más területeken is eredményesen felhasználható.

3 Zeneelméleti háttér

3.1 Zenei hangok

A zenében 12 törzshangot különböztetünk meg, ez a zongora billentyűin jól megfigyelhető:



1. ábra: zongorabillentyűzet a hangjaival

Az alsó sorban a fehér billentyűk hangjai, a felső két sorban a feketéké olvashatók. A frekvenciaspektrumon ez a 12 hang ismétlődik a következő szabály szerint [34]:

$$f_n = 440 * (2^{\frac{1}{12}})^n, \quad (1)$$

ahol az *n* mondja meg, hogy melyik hangról van szó. A *normál A* hangnál (definíció szerint *440 Hz*) az *n* nullával egyenlő. Ha mondjuk a normál A-tól balra lévő első *E* hanghoz tartozó *n* értéket szeretném megmondani, akkor meg kell vizsgálnom, hogy a zongorabillentyűkön hány darab félhangot kell lépnem, és milyen irányba. Ebben az esetben ötöt kell lépnem és balra (A→Asz→G→Fisz→F→E), így *n* helyére a -5-öt kell behelyettesítenem. Fontos megemlíteni, hogy az angolszász irodalomban *B*-nek mondják azt a hangot, amit a magyarban *H*-nak, a dolgozatban egy-két helyen előfordul, hogy az angol jelölést használom, mint például a 2. ábrán.

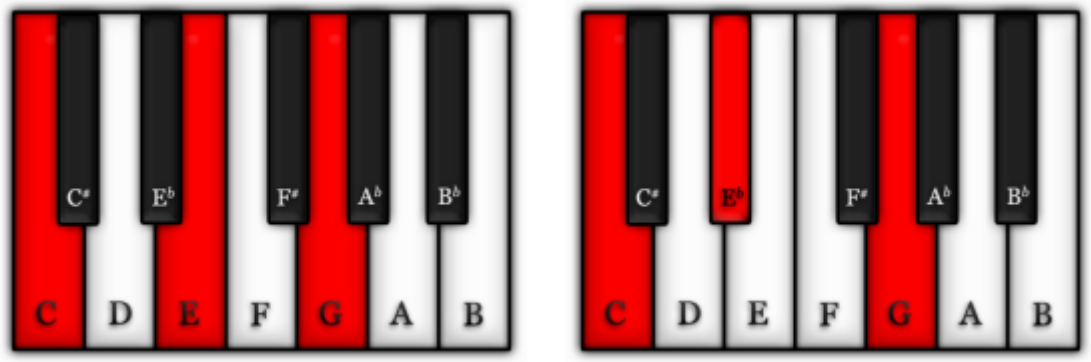
A dolgozatomban a C-től H-ig terjedő periódikusan ismétlődő 12 hangot oktávnak fogom nevezni (például 4. oktáv). A zenében alapvetően az oktáv nevet bármilyen 12 félhangnyi távolságra értjük.

3.2 MIDI kód

A MIDI kóddal a zenei hangokat kódoljuk. A MIDI kódolás szerint a *440 Hz* frekvenciájú „*A”* hang a *69*-es. A többi hang kódját megkaphatjuk, ha a 69-hez hozzáadjuk a félhangokba mért előjeles távolságát az „*A”*-tól. Az alapoktáv, amiben a 440 Hz-es „*A*” hang van, a 4. oktáv. A MIDI kódot a -1. oktáv „*C”*-jétől a 9. oktáv „*G”*-jéig számoljuk (összesen *127* zenei hang).

3.3 Akkord

Akkordnak, hangzatnak vagy harmóniának nevezzük a hangsor több meghatározott fokának együttes megszólalását. Sokféle akkord létezik, hármas, négyeshangzatok, és még megkülönböztetjük ezeknek az akkordfordításait is. Az akkordra nem kritérium, hogy konszonáns (összecsengő) legyen, az is akkord, amely disszonáns hangok összessége. A két hangból álló akkordot üres akkordnak nevezzük. Egy harmónia egyidőbeli megszólaltatásához legalább három zenei hang szükséges. Megkülönböztetünk *dúr* és *moll* akkordokat, melyek közül a dúr kissé vidámabb, szabadabb érzést kelt, a moll pedig melankolikusabb hangzású, szomorkás, sejtelmes. A 2. ábrán egy *C-dúr* és egy *c-moll* akkord látható zongorabillentyűkön felrajzolva [13].



(a) C-Dúr akkord (b) C-moll akkord

2. ábra: dúr és moll akkordok a zongorabillentyűzeten [13]

A munkám során a felismerési feladatban az egyszerűség kedvéért csak a dúr és moll akkordok felismerését valósítottam meg. A nyugati zenében ezek a leggyakrabban használt akkordok.

A dúr és a moll akkord egyaránt 3 hangból áll. Van egy alaphangja, amiről az elnevezését kapja (pl. *C-dúr*, *a-moll*). Az akkord képzése a következő módon történik:

a) dúr:

1. hang: alaphang (pl.: **C**)

2. hang: az alaphangtól 5 félhang (nagyterc) távolságra lévő hang (pl.: C - **E**)

3. hang: az alaphangtól 8 félhang (kvint) távolságra lévő hang (pl.: C - **G**)

b) moll:

1. hang: alaphang (pl.: C)

2. hang: az alaphangtól 4 félhang (kisterc) távolságra lévő hang (pl.: C - **Esz**)

3. hang: az alaphangtól 8 félhang (kvint) távolságra lévő hang (pl.: C - **G**)

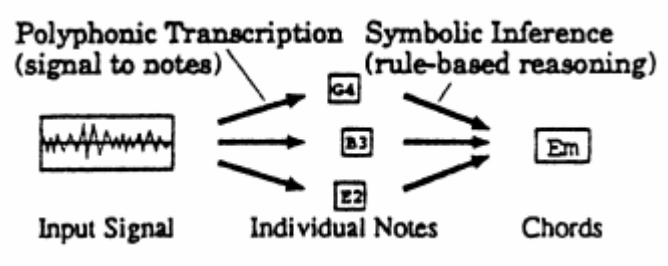
Az egyes akkordoknak akkordfordításai is léteznek. Abban az esetben, ha az akkord alaphangja a basszus szólamot képezi, akkor alaphelyzetről beszélünk, viszont ha az akkord más összetevője képzi azt, akkor valamilyen akkordfordításról. A hármashangzatnak két fordítása létezik: szext fordítás, és kvartszext fordítás. Például a C-dúrra nézve az alapeset: C-E-G, a szext fordítás E-G-C, és a kvartszext fordítás a G-E-C [14].

Az akkordfordítást a munkám során nem vettem különböző esetnek, azaz a felismerés során csak azt figyeljük, hogy az akkordban szereplő hangfajták az adott időpillanatban jelen vannak-e. Tehát például a C-dúr esetén nem kell feltétlenül C-E-G sorrendben lenni a hangoknak a frekvenciaspektrumon, lehet E-G-C is, és lehet az egyes típusú hangokból több is, pl.: C3-G3-C4-E4-G4 (a 3, 4 az adott oktávot számozza).

4 Már létező akkordfelismerő algoritmusok

4.1 Tradicionális megközelítés

Ennek a megoldásnak az alapgondolata rendkívül egyszerű, és nagyon kézenfekvő, bár a hatékonysága közel sem megfelelő. A hagyományos megközelítés szerint a bemeneti polifonikus audio jelből az egyes időpillanatokban detektáljuk a hangokat, majd megmondjuk, hogy ezek a hangok milyen akkordot alkotnak (3. ábra). Ehhez a zenét időtartományban felosztjuk kisebb szakaszokra. Az egyes szakaszoknak vesszük a Fourier-transzformáltját, és ismerve a zenei hangok névleges frekvenciáját, megmondjuk, hogy melyik hang milyen intenzitással szólalt meg. Sajnos ez mégsem ilyen egyszerű az alábbiak miatt. Egy akkord megszólalásának rengeteg variációja létezik. Ez egyrészt függ a hangszertől, hiszen különböző felharmonikusakat állítanak elő, illetve különböző időtartománybeli képet mutatnak, például az egyes időbeli lecsengések változnak. Másrészt függ a dinamikától (*forte, mezzo forte, piano*), játékstílustól (*arpeggio, staccato, legato, tepping, slap*), az egyes, hangszer által kiadott hangot formáló eszközök (sustain pedál, tremoló kar, effektpedál) használatától. Ezenkívül függ a felvétel készítésének körülményeitől (reflexiómentes szoba vagy egyszerű terem, mikrofon minősége), mivel a környezeti zajok a bemeneti audio jel spektrumában hiba jellegű komponensként jelennek meg [8]. Végül a feladatot nehezíti még a Diszkrét Fourier Transzformáció frekvenciatartománybeli átlapolódása és a harmóniák időbeli átlapolódása. Az akkordfelismerés ezekből a hibákból adódóan nehéz feladat [1]. Megjegyzendő, hogy a tradicionális megközelítéssel akár kottázó programot is készíthetnénk, viszont az akkordfelismeréshez nincs szükség a konkrét megszólaltatott hangok megállapítására.



3. ábra: tradicionális megközelítés [1]

4.2 Akkordfelismerés mintamegfeleltetéssel

4.2.1 Fujishima módszere

Az első komolyabb akkordfelismerő módszer a *Fujishima* által 1999-ben kidolgozott eljárás [1]. Az algoritmus blokkvázlata a 4. ábrán látható. Időben felosztjuk a bejövő zenei jelet kis részekre, melyeket külön-külön vizsgáljuk, azaz az aktuálisan vizsgált $x(n)$ zenerészlet Diszkrét Fourier Transzformált (*DFT*) spektrumát képezzük ($X(k)$). Ezek után az $X(k)$ spektrumban található intenzitásokból egy 12 elemű vektort készítünk. A cikk ezt nevezi *Pitch Class Profile*-nak, röviden *PCP* vektornak. A PCP vektor egyes elemei a 12 féle félhang közül reprezentálnak egyet-egyet. A PCP vektort a következő algoritmus szerint származtatjuk [1]:

$$PCP(p) = \sum_{M(l)=p} \|X(l)\|^2 \quad (2)$$

$$M(l) = \begin{cases} -1, ha\ l = 0 \\ round\left(12log_2\left(\frac{f_s\left(\frac{l}{N}\right)}{f_{ref}}\right)\right) mod\ 12, ha\ l = 1,2,\dots,N/2-1 \end{cases} \quad (3)$$

A $p = 0,\dots,11$, ami az egyes fél hangoknak felel meg. f_{ref} a választott referencia frekvencia (valamelyik mélyebb C hang frekvenciája), ahol az $M(l)$ értéke 0. Az f_s*l/N reprezentálja a frekvencia bineket, amin l szerint lépkedünk végig. $M(l)$ értéke a moduló osztás miatt $0,\dots,11$ értékeket vehet fel. Az $M(l)$ függvény tehát megadja, hogy az l-edik DFT bin milyen hangnak felel meg. Jellegre ez egy logaritmikusan változó lépcsőzetes függvény.

Az eljárás az elkészített PCP vektort előre elkészített mintákkal hasonlítja össze. Az akkordok mintái a cikk által *Chord Type Templates*-nek (*CTT*) nevezett adatbázisban találhatóak. A módszer összesen 27-féle akkordtípust (dúr, moll, szűkített, bővített hármashangzat, szeptim akkordok, stb.) különböztet meg. A CTT-ben lévő vektorokban 1-es szerepel annál a hangnál, ahol az adott vektorhoz tartozó akkord felhasználja azt a hangot, a többi elemhez 0-t rendelünk. Például a *C* szeptimakkordhoz a [1,0,0,0,1,0,0,1,0,0,0,1] tömböt rendeljük. Az akkordfelismeréshez a szerző két fő mintakeresési algoritmust használt fel: a legkisebb négyzetek módszerét, illetve a súlyozott összeg képzést.

A *legkisebb négyzetek módszerénél* a kiszámított PCP vektor és az egyes CTT vektorok ($T_c(p)$) eltérését vizsgáljuk négyzetes értelemben. A CTT vektorok közül a legkisebb eltéréssel rendelkezőt vesszük az akkordfelismerés eredményének [1].

$$Score_{nearest,c} = \sum_{p=0}^{11} (T_c(p) - PCP(p))^2 \quad (4)$$

A *súlyozott összeg képzéséhez* az elkészített PCP vektornak és az egyes, CTT-ből származó ($W_c(p)$) súlyozó vektoroknak a skaláris szorzatát vizsgáljuk. Amelyik $W_c(p)$-vel érjük el a legnagyobb skaláris szorzatot, ahhoz tartozó akkordot vesszük a felismerés eredményének [1].

$$Score_{weighted,c} = \sum_{p=0}^{11} W_c(p) * PCP(p) \quad (5)$$

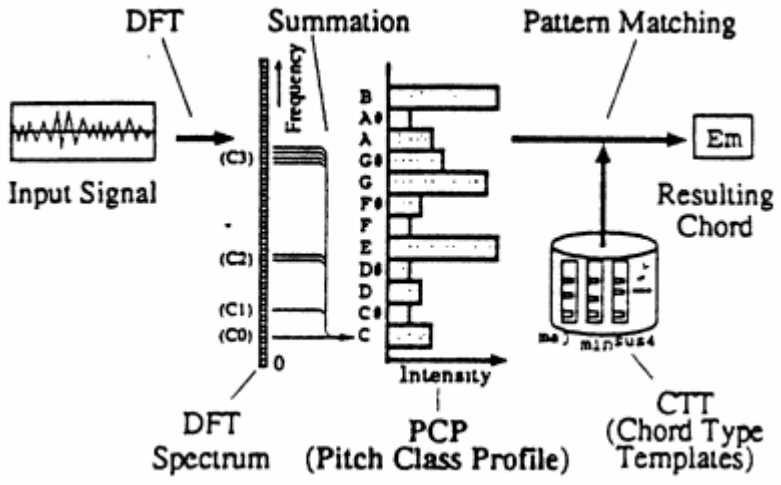
Fujisima a PCP vektorok készítéséhez többféle heurisztikát használt fel.

Főbb heurisztikák:

- Feltételezzük, hogy az akkord általában több PCP keretig tart, így átlagolással simítást végzünk annak érdekében, hogy csökkentsük a zajt
- Az akkordváltás hirtelen történik, akkor, amikor a PCP vektor megváltoztatja az irányát, így nekünk a PCP vektor irányát kell figyelnünk

Kevésbé fontos heurisztikák:

- Kevésbé fontos frekvenciatartományokat (nagyon alacsony vagy magas frekvenciák) nem vesszük figyelembe
- Ablakozás használata a DFT számításánál
- Csönd felismerés, a szükségtelen analízis elkerülése végett



4. ábra: akkordfelismerés a Fujisima-féle módszerrel [1]

A Fujishima-féle módszer nagy pontosságot leginkább olyan zenében tud elérni, amiben csak egy hangszer játszik. Fujisima az algoritmusát kétféle módon tesztelte. Először egy *YAMAHA PSR-520* szintetizátorral, ahol 3 különböző hangszínnel játszották a 27 féle akkordot. Az *Occalina* hangszínre 100%, míg a *GrandPno* és *Strings1*-re 93% lett az eredmény. A másik tesztben egy viszonylag egyszerű, egy hangszeres komolyzenei darabról készült felvételt elemez, melyre 94%-os eredményt kapott.

4.2.2 EPCP vektoros módszer

12 elemű *PCP* vektort többféleképpen készíthetünk, de az alapgondolat általában hasonló [2]. A rövid zenerészletnek vesszük a DFT spektrumát, és kiszámítjuk abból a logaritmikus frekvenciafelbontású *Q-transzformált* spektrumot ($X_{CQ}(k)$), ami az emberi fül frekvenciafelbontását tükrözi [4]. A *k*-adik spektrális komponenst a következőképpen definiáljuk [2]:

$$f_k = (2^{1/B})^k f_{min}, \quad (6)$$

ahol $b = 1,2,\ldots,B$ a binek száma egy oktávon belül. Itt megjegyzendő, hogy a bin ebben az esetben nem a DFT frekvenciabinjeire értendő, hanem arra, hogy egy oktávnyi frekvenciaspektrumot mennyi részre osztjuk fel. *B*-nek legalább 12-nek kell lennie. A szakirodalomban található $B = 24, 36$ is [3,6,7,8], amivel finomabb felbontás érhető el. $X_{CQ}(k)$-ból az alábbi kifejezéssel könnyen számítható a *PCP* vektor, amit egyes helyeken *chroma*-nak is hívnak [3,6,7,17][2].

$$CH(b) = \sum_{m=0}^{M-1} \left| X_{CQ}(b + mB) \right| \quad (7)$$

Az EPCP vektoros módszer *Fujisima* munkájának a továbbgondolása, amit *Kyogu Lee* dolgozott ki [2]. Az algoritmusában a PCP vektor helyett annak a továbbfejlesztett verzióját, az *EPCP*-t használja (*Enhanced Pitch Class Profile*), melynek a korrelációját vizsgálja előre elkészített mintákkal.

A probléma a korábbi algoritmussal a következő. A Fujisima-féle CTT adatbázis csak bináris vektorokból áll. Ez egy idealizált modell, a valóságban általában a nem akkordhangoknak megfelelő PCP elemek is megjelennek valamekkora, nullánál nagyobb amplitúdóval. Ennek az egyik fő oka, hogy az egyes hangok felhangjai is megszólalnak, és azt is beleszámítjuk a PCP vektorba. Ez probléma lehet akkor, amikor két hármashangzat csak egy hangban tér el egymástól. Ilyen például a C-dúr (C-E-G) és az a-moll (A-C-E) esete. Előfordulhat olyan, hogy egy a-moll PCP vektorában a G hang nagyobb összintenzitással jelentkezik, mint az A. Ennek az oka lehet például, hogy felhangok, vagy nem akkordhangok is megszólalnak. Ekkor a PCP mintákkal való korrelációból származó érték a C-dúrnál lesz maximális, így az algoritmus ezt veszi a felismerés eredményének. Az EPCP-vel az ilyen jellegű hibák számát tudjuk csökkenteni.

Az EPCP készítése a DFT spektrum helyett a *HPS*-ből (*Harmonic Product Spectrum*) indul ki. A HPS-t korábban periódikus jelben való alapfrekvencia detektálásra, vagy emberi beszédben csúcsdetektálásra használták [5]. Ennek az alapgondolata a

következő. Amikor egy ember, vagy egy hangszer kiad egy hangot, akkor megszólal az alapfrekvencia, illetve annak egész számú többszörösei is. Ha a frekvenciatengely mentén skálázzuk a spektrumot egy egész számmal és ezek szorzatát vesszük, akkor az így kapott spektrum maximális csúcsa lesz az alapfrekvencia becslője [2].

$$HPS(\omega) = \prod_{m=1}^{M} |X(m\omega)| \quad (8)$$

$$F_0 = argmax_{\omega}\{HPS(\omega)\}, \quad (9)$$

ahol *M* a figyelembe vett harmonikusak száma, F_0 pedig a becsült alapfrekvencia. A HPS egyszólamú zenéknél, emberi hangnál jól működik. Ahhoz, hogy akkordfelismerésre használhassuk, nem egész számmal, hanem 2 hatványaival skálázzuk a spektrumot. Így tulajdonképpen minden oktávot egy helyre transzponálunk [2].

$$HPS(\omega) = \prod_{m=0}^{M} |X(2^m\omega)| \quad (10)$$

Az így számított HPS spektrumból számítjuk ki az EPCP vektort, melynek a korrelációját vizsgáljuk előre elkészített akkord mintákkal.

Az algoritmus teszteléseséhez a következő paraméterbeállításokat használták. A tesztelendő zenét *11025 Hz*-en alulmintavételezték. Az zenei jel feldolgozásához *8192* minta hosszúságú ablakozó függvényt használtak (ez *743 ms*-nak felel meg). A relatív hosszú ablak azért szükséges, hogy olyan akkordokról is kaphassunk információt, amelyeket bontva játszanak (ún. *arpeggio* akkord). Az EPCP vektor készítése *36 bin/oktávos* Q-transzformációval készült, a figyelembe vett frekvenciatartomány: *96…5250 Hz*. A Q-transzformáció elvégzéséhez felhasználták *C. Harte* és *M. B. Sandler* chroma készítési módszerét, melyben a lehetséges félrehangolt hangszerek hibáját küszöbölték ki olyan módon, hogy az egyes, mért intenzitáscsúcsokat újraigazították a zenei hangok intenzitáscsúcs-eloszlásai alapján [6]. Az algoritmus teszteléshez egyetlen zeneművet használtak fel: *Bach C-dúr Prelúdium*. Az EPCP-s módszer eredményét a PCP-vel hasonlították össze. Megfigyelhető, hogy a PCP-hez képest kevesebb a hibás akkordfelismerés. A harmonikusan közel lévő akkordokkal kapcsolatos probléma az EPCP-vel kevesebb helyen történik. A PCP-vel szemben helyesen felismeri a *d-moll* (a PCP-nek az eredménye a *D-dúr*) és az *a-moll* (a PCP szerint *C-dúr*) akkordot, viszont előfordul, hogy a *h-moll* helyett *D-dúrt* mond.

4.3 Akkordfelismerés rejtett Markov-modell használatával

4.3.1 Sheh és P. W. Ellis módszere

Sheh és *P. W. Ellis* által publikált cikkben [7] már nem pusztán jelfeldolgozási, hanem ahhoz kapcsolódóan statisztikai és valószínűségszámítási módszereket is olvashatunk. A munkájukban a Fujisima által végzett kutatások eredményét vették alapul. Az algoritmus alapgondolatát az 5. ábrán tudjuk nyomon követni. Sheh és Ellis 147-féle akkordot felismerő algoritmus implementálását tűzte ki célul.

4.3.1.1 Jelfeldolgozási rész

Első lépésként a *11025 Hz*-en mintavételezzük a zenei jelet, majd feldaraboljuk azt *N=4096* mintaszámú, egymással átfedésben lévő részekre. Ezután short-time Fourier transzformációt végzünk [7]:

$$X_{STFT}[k,n] = \sum_{m=0}^{N-1} x[n-m]w[m]e^{-j2\pi km/N}, \quad (11)$$

ahol *k* indexeli a frekvenciatengelyt ($0 \leq k < N$-1), m pedig az időtengelyt. A *w[m]* egy *N* pontos Hanning ablak. A meglévő spektrumból PCP vektort készítünk. Fujisima algoritmusához képest ebben finomabb felbontású (24 elemű) PCP vektort használunk [7].

$$p(k) = \left\lfloor 24 * log_2\left(\frac{k}{N} * \frac{f_{sr}}{f_{ref}}\right)\right\rfloor mod\ 24 \quad (12)$$

$$PCP(p) = \sum_{k:p(k)=p} |X(k)|^2 \quad (13)$$

4.3.1.2 Rejtett Markov Modell

Az RMM alkalmazása az akkordfelismeréshez abból az ötletből származik, hogy a beszédfelismeréshez ez meglehetősen hasonló probléma, és ahhoz már készültek elég jó programok RMM alkalmazásával. Az RMM lényege, hogy van egy rejtett változó, amit nem ismerünk, de van egy megfigyelésünk, amiből az ismeretlen változóra tudunk valószínűségi alapon következtetni. A rejtett változó az időben gyorsan változhat. Az RMM-mel részletesebben később találkozunk.

Az RMM megalkotásához statisztikai megfigyeléseket kell végezni. A Sheh és P.W. Ellis-féle munkában Beatles dalokból készített PCP vektorokat használtak fel az RMM paramétereinek a beállításához (tanítási folyamat). 18 dallal végeztek tanítást, és 2 dallal teszteltek.

Sheh és Ellis az RMM érzékelőmodelljének egy 12 dimenziós *egyszerű Gauss* modellt feltételezett. Ennek a modellnek a megalkotásához szükséges a 12 dimenziós átlagvektor, illetve a 12 dimenziós kovariancia mátrix. Az átlagérték vektort véletlenszerűen inicializálták, tekintve arra, hogy az elvárásmaximalizációs algoritmus később beállítja azt a megfelelő helyre. A kovarianciamátrix megalkotásához azt feltételezték, hogy a PCP vektor egyes elemei korrelálatlanok, így a mátrix diagonálisán

kívül az összes elem nulla. A meglévő μ átlagértékvektor és a Σ kovarianciamátrix paraméterekkel az egyszerű Gauss-modell a következőképpen írható fel [15, 18].

$$p(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}(\det(\Sigma))^{\frac{1}{2}}}\exp\left(-\frac{1}{2}D^2\right) \quad (14)$$

A (14)-es képletben a d a dimenziót jelenti, D^2-et pedig az alábbi módon definiáljuk:

$$D^2 = (x-\mu)\Sigma^{-1}(x-\mu)^t \quad (15)$$

4.3.1.3 Elvárásmaximalizáció

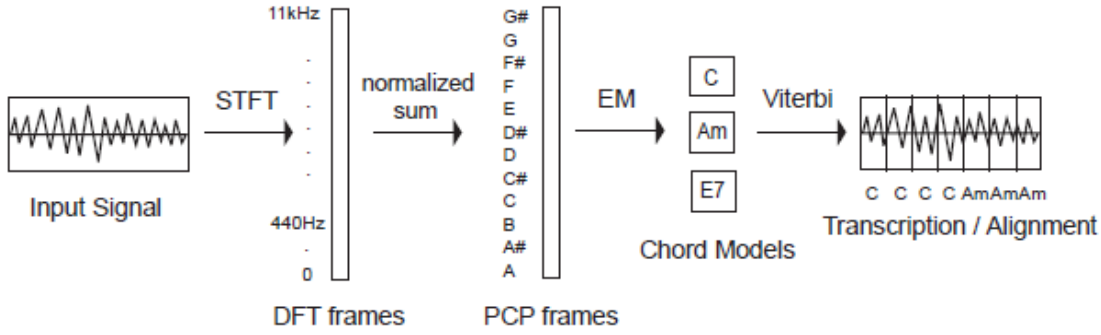
Itt feltételezzük, hogy rendelkezésre áll a $P(X,Q|\Theta)$ együttes sűrűségfüggvény, a megfigyelés és a rejtett változó között. X jelöli a megfigyelést, Q az ismeretlen paramétert. A mi esetünkben a megfigyelés a PCP vektor, és rejtett változó az akkord. Θ jelenti a RMM aktuális paramétereit. A modell megalkotása után (kezdeti paraméterek meghatározása) iteratívan számítjuk a következő kifejezést [7]:

$$E(\log P(X,Q|\Theta)) = \sum_{Q} P(Q|x,\Theta_{old}) \log\big(P(X|Q,\Theta)\big) P(Q|\Theta) \quad (16)$$

Ezen képlet könnyebben értelmezhető [24] (41) és (42) egyenletei alapján, és a dolgozatom (44), (45) és (46) képletei ebből közvetlenül származtathatóak. Ezen eljárással az RMM paramétereit folyamatosan javítjuk. Az algoritmus addig fut, amíg a javulás nem kisebb egy előre meghatározott ε-nál.

4.3.1.4 Viterbi-algoritmus

Az RMM modell a paraméterek felhasználásával egy adott megfigyeléssorozathoz rendeli hozzá a rejtett változók legvalószínűbb sorozatát.



5. ábra: Sheh és P.W.Ellis akkordfelismerő algoritmusának blokkvázlata[7]

A Sheh és P.W. Ellis által kidolgozott módszer a tanítást és a tesztelést is Beatles-dalokkal végezte. Ilyen feltételek mellett az algoritmus 75%-os pontosságot tudott produkálni.

4.3.2 Bello és Pickens módszere

4.3.2.1 Chroma készítés

Bello és *Pickens* is úgy döntött, hogy az algoritmusában a PCP vektort használja fel az adott időszelethez tartozó mérés reprezentációjaként [3]. A PCP-t ebben a munkában *chromának* hívják. A chroma számítás ablakozással és konstans Q-transzformációval történik olyan, módon, mint *Lee* munkájában ((6)-os képlet) [2]. A zenei jelet 11025 Hz-en alulmintavételezték, és az f_{min}=98 Hz és f_{max}=5250 Hz közötti energiacsúcsokat vették csak figyelembe. A nagyobb pontosság érdekében ők is 36 bin/oktáv felbontással dolgoztak.

A chromák készítésénél figyelembe vették azt is, hogy valódi felvételek gyakran nem tökéletesen behangolt hangszerekkel történik. Így Lee-hez hasonlóan [2] ők is felhasználták *Harte* és *Sandler* chroma hangolási algoritmusát [6], melyet egy egyszerűsített verzióban implementáltak. Először is vették az összes csúcsot a chromagramban. Egy hisztogramot készítettek ezen adatokkal, és megvizsgálták, hogy az egyes eloszlásokban van-e némi eltolódás a várt értékektől. Ha van, akkor az utal arra, hogy a felvétel kissé félrehangolt hangszerekkel készült. Ebben az esetben az eloszlásból számítanak egy korrekciós faktort, és ezzel korrigálják ki a chromagramot. Végül az áthangolt chromagramot aluláteresztő szűrővel szűrik, hogy csökkentsék a létrejött éleket.

4.3.2.2 Tempófelismerés és a zene részekre osztása

Harmóniák váltása általában az ütemekre történik, és gyakran egy akkord hosszabb ideig szól, mint például *Lee* munkájában a DFT ablak hossza (743 ms). Így ez a felbontás néha szükségtelen, sőt gyakran zavaró az akkordfelismerés szempontjából. Ezen okokból kifolyólag Bello és Pickens egy tempófelismerő algoritmust implementált, melynek segítségével osztották fel a zenét kisebb szakaszokra. Az algoritmust *Davies* and *Plumbley* publikációja alapján írták [16], ennek az egyszerűsített verzióját valósították meg.

4.3.2.3 Rejtett Markov-modell

Az akkordfelismerést Bello és Pickens is RMM segítségével végezte. Az RMM inicializálása a következőképpen történt.

a) Kezdeti valószínűségi eloszlás:

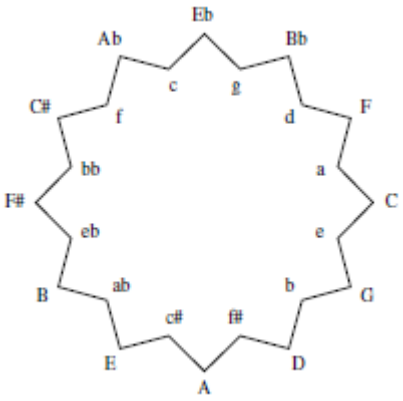
   Mivel semmilyen okuk nem volt arra, hogy apriori módon bármelyik állapothoz magasabb valószínűséget rendeljenek, így a kezdeti valószínűségi eloszlás (π) értéke 1/24 mind a 24 állapotra.

b) Állapotátmenetvalószínűségi mátrix:

   Ez a mátrix mutatja meg, hogy ha egy adott állapotban vagyunk, akkor mekkora az egyes lehetséges következő állapotok valószínűsége. Az állapotátmenetvalószínűségi mátrix ($\boldsymbol{A}$) inicializálásához Bello és Pickens azt a zeneelméleti tudást használta fel, hogy az egymáshoz képest jobban konszonáns

hármashangzatok gyakrabban követik egymást. Például a C-dúrt csak nagyon ritka esetben követné egy Fisz-dúr akkord, viszont annak a valószínűsége, hogy egy a-moll hármashangzat jön, viszonylag magas. A hármashangzatok közelségének a mértékét a *módosított kvintkörrel* tudjuk szemléltetni. Ez látható a 6. ábrán. A kvintkör lényege ebben a felhasználásban a következő: a belső kvintkörön találhatóak a moll akkordok, a külsőn pedig a dúrok. Ha vagy a belső vagy a külső körben elindulok az adott akkordtól az óramutató járásával megegyező irányban, akkor a közvetlen utána következő hármashangzat alaphangja kvint (öt hang) távolságra található.

A módosított kvintkörön a közvetlen egymás mellett lévő akkordok harmonikusan is közeliek. Általánosan mondható, hogy a módosított kvintkörön minél közelebb van két akkord, annál közelebb van harmonikusan is.



6. ábra: módosított kvintkör [3]

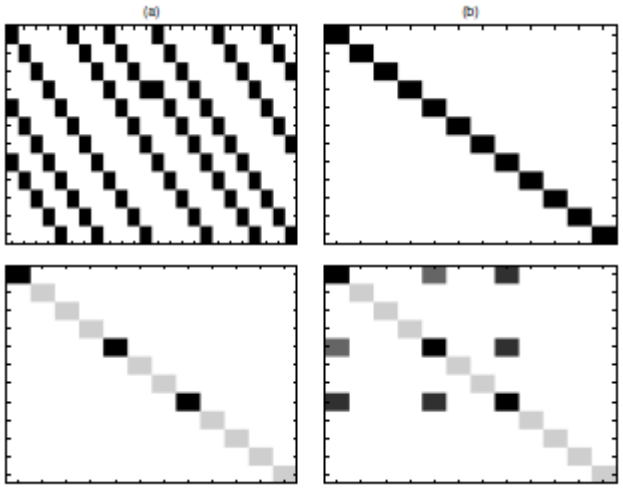
Ezek alapján az egyes átmenetekhez Bello és Pickens a következőképpen rendelt valószínűségeket [3]:

$$\frac{(12-d)+\varepsilon}{144+24\varepsilon}, \quad (17)$$

ahol d a két akkord abszolút távolsága a módosított kvintkörön, ε pedig egy megfelelően kicsi szám.

c) Érzékelőmodell:

Az érzékelőmodell (*B*) megalkotásához a megfigyeléseknek folytonos eloszlási függvényt feltételeztek, egy egyszerű többváltozós Gaussian eloszlást



7. ábra: az átlagértékvektor és a kovarianciamátrix inicializációja [3]

használva minden állapotra, melynek a paraméterei a μ átlagértékvektor és a Σ kovarianciamátrix. Sheh véletlenszerűen inicializált μ vektort és olyan Σ mátrixot használt, amelyben a nem diagonális elemek mind nullák, azaz a chromák elemeit teljesen korrelálatlannak feltételezte [7]. Bello és Pickens már többféle inicializálást is kipróbált. Ez látható a 7. ábrán. Átlagértékvektornak bináris vektort választottak minden akkordhoz, melynek adott eleme 1-es értéket kap, ha az azt reprezentáló hang szerepel az akkordban, illetve 0-t, ha nem, hasonló módon, mint Fujisima CTT-jében [1]. A kovarianciamátrixokat háromféle inicializálással próbálták ki, ami a C-dúr akkord esetére a 7. ábrán látható *b, c* és *d* jelű mátrix. Az első a Sheh által felhasznált diagonális mátrix (*b*) [7]. A második a súlyozott diagonális mátrix, ahol egy adott akkordhoz tartozó korrelációs mátrixban a nem akkordhangokra vonatkozó diagonális elemek kisebb súllyal szerepelnek (*c*). A harmadik kovarianciamátrixot zeneelméleti megfontolások alapján inicializálták a következőképpen. Az akkordhangok erősen korrelálnak egymással, és ez a korreláció szimmetrikus. Az egyes hangok saját magukkal is korrelálnak. Így egy szimmetrikus mátrixot kapunk. A konkrét értékek hozzárendeléséhez felhasználták azt az empirikus tényt, hogy egy hármashangzatban a kvint fontosabb szerepet tölt be, mint a terc. Ennél fogva a kovarianciát az alaphang és a kvint között 0.8-nak, a terc és a kvint között 0.8-nak, és az alaphang és a terc között 0.6-nak határozták meg. A diagonálisban szereplő nem akkordhangoknak 0.2 értéket adtak. A 7. ábra *d* jelű mátrixa ábrázolja ezt.

4.3.2.4 Eredmények

A készített akkordfelismerő programot Bello és Pickens többféle paraméterbeállítás mellett tesztelte. Az eredményt a következő táblázat foglalja össze. A legjobb eredményt akkor kapták, amikor:

- a zenét ritmus alapján osztották fel
- a kezdeti valószínűségi eloszlást egyenlő (1/24) valószínűséggel,
- az állapotátmenetvalószínűségi mátrixot a módosított kvintkör alapján inicializálták
- a megfigyelési modellnél az átlagértékvektornak a bináris vektorokat,
- kovarianciamátrixnak a zeneelméleti tudást is magába foglalót választották
- az újrabecslést a π, A és B paraméterek közül csak a π vektorra és az A mátrixra hajtották végre.

$\boldsymbol{\pi}$	**A**	**B**		újra-becslés	Találati pont [%]		
		$\boldsymbol{\mu}$	$\boldsymbol{\Sigma}$		CD1	CD2	Össze-sen
1/24	véletlenszerű	bináris mintavektor	csak diagonális	π, A, B	22.88	29.83	26.36
1/24	véletlenszerű	bináris mintavektor	súlyozott diagonális	π, A, B	34.14	36.24	35.19
1/24	véletlenszerű	bináris mintavektor	diagonális + zeneelméleti információ	π, A, B	33.13	44.36	38.74
1/24	kvintkör	bináris mintavektor	diagonális + zeneelméleti információ	π, A, B	38.09	47.75	42.93
1/24	kvintkör	bináris mintavektor	diagonális + zeneelméleti információ	π, A	58.96	74.78	66.87
1/24	kvintkör	bináris mintavektor	diagonális + zeneelméleti információ	π, A	**68.55**	**81.54**	**75.04**

1. táblázat : Bello és Pickens eredményei

4.4 Akkordfelismerés neurális hálózattal

4.4.1 Osmalskyj módszere

Osmalskyj az akkordfelismerő programját neurális hálózat segítségével implementálta [8]. Ezenkívül összehasonlítási alapnak egy általa választott mintamegfeleltetéses algoritmust is implementált. Az általa készített program célja a nyugat-európai zenében leggyakrabban felhasznált 10 akkord felismerése. Ezen akkordok a következőek: *A, Am, Bm, C, D, Dm, E, Em, F, G*.

4.4.1.1 A neurális hálózat bemenete

A neurális hálózat bemenetének az adott zenerészletet kompakt módon leíró vektort, a PCP vektort használta fel. Az általa készített PCP vektort a következőképpen definiálta [8]:

$$PCP^*(p) = \sum_l \|X(l)\|^2 \, \delta(M(l), p), \quad (18)$$

ahol *p=1,…,12*. *M*(*l*)-t függvényt ugyanúgy kell értelmezni, mint a Fujisima-féle módszerben. $\delta(.,.)$ a *Kronecker delta*, amely olyan függvény, mely 1-et ad, ha az argumentumában szereplő két érték megegyezik [9]. A PCP készítést a 12 mellett kipróbálta 24, 36 bin/oktáv paraméterrel is, viszont nem talált jelentős változást, így az algoritmus gyorsítása érdekében maradt a 12-nél. A PCP képzés így gyakorlatilag a Fujisima-féle módszerrel azonos módon történik.

4.4.1.2 Az összehasonlításhoz felhasznált mintamegfeleltetéses algoritmus

Osmalskyj ideális *PCP* vektormintákat készített minden akkordhoz (az akkordhangokon kívül mindegyik hanghoz tartozó intenzitás értéke nulla). A tesztelésre felhasznált zenerészletekből készített *PCP* vektorokat ezen mintákkal hasonlította össze a legközelebbi szomszéd módszerrel, a *Bhattacharya-távolságot* felhasználva.

Két vektor Bhattacharya-távolságát a következőképpen definiáljuk [11]:

$$-ln\big(\rho(P_1, P_2)\big), \quad (19)$$

ahol ρ (a *Bhattacharya-együttható*):

$$\rho(p_1, p_2) = \sum_x \sqrt{p_1(x)p_2(x)}. \quad (20)$$

4.4.1.3 Tanítóhalmaz

Mint a gépi tanulási módszereknél általában, a neurális háló esetén is elengedhetetlen megfelelő mennyiségű tanító adathalmaz, hogy felépíthessük a modellünket. Ehhez Osmalskyj saját akkordadatbázist készített, melyek elérhetőek az interneten. A felvételek *44,1 kHz* mintavételi frekvenciával, *16 bites* kvantálással és *16384* minta (*0.37 mp*) hosszúságú ablakkal készültek. Az adatbázisát két nagy részre osztotta.

Az első részét csak gitárral készítette, viszont abból négyfélét használt fel (nejlon húros klasszikusgitár, és három különböző akusztikus gitár). Mind a négyfajta gitárral 25-ször játszotta el mind a 10-féle akkordot, és ezt felvették egy reflexiómentes szobában, nagy sávszélességű mikrofonnal, illetve egy zajos szobában, egyszerű, elő adásokhoz használt mikrofonnal. Tehát mind a 10-féle akkordról 100 darab közel zajmentes felvétel és 100 darab zajos felvétel készült, így az adatbázis első részhalmazának az elemszáma 2000. Az akkordok lejátszásakor figyeltek arra is, hogy az többféle játékstílusba történjen (*arpeggio, staccato, legato*).

A második adathalmazt négyféle hangszerrel készítette (gitár, zongora, hegedű, harmonika). Ez egy jóval kisebb adathalmaz, csak 100 akkordot vettek fel hangszerenként (minden akkordfajtából 10 darab) melyet független teszthalmaznak használt fel.

4.4.1.4 Az elkészített neurális hálózat tulajdonságai

A tanítandó neurális hálózat előrecsatolt és kétrétegű (rejtett réteg, kimeneti réteg). A neurális hálózat ismeretlen paraméterei az *ún.* súlyok. Ezeknek azon értékeit keressük, melyek a tanítóhalmazhoz leginkább illeszkednek. A tanításához Osmalskyj egy klasszikus gradienscsökkentő algoritmust használt fel, negatív log-likelihood költségfüggvénnyel, melyet a [10] szakirodalom *11.4* fejezete mutatja be részletesen. A neurális hálózat szerkezete a következő. Definiálva van 12 bemeneti változó, amely megfelel a 12 zenei félhangnak (PCP vektor elemei), illetve a kimenete egy 10 elemű vektor, melyben az egyes elemek értékei az egyes felismerendő akkordok valószínűségeit adják meg. Az egyéb paramétereket az alábbi táblázat foglalja össze.

Paraméter	**Érték**
Rejtett rétegek száma	1
Neuronok száma a rejtett rétegben	35
Tanulási faktor	0.001
Momentum	0.25
Súlypusztítás	0.0

2. táblázat: Osmalskyj által használt neurális hálózat paraméterei

4.4.1.5 Eredmények

Osmalskyj elsőként az optimális tanítóhalmazt kereste. Ehhez a 2000, gitárral felvett akkordokkal háromféle lehetőséget próbált ki. Először csak a zajmentes akkordokkal tanította a neurális hálózatot, másodszor csak a zajossal, míg harmadjára mindkettőt vegyesen használta fel. Az egyes tanítóhalmazok 700, míg a hozzájuk tartozó teszthalmazok 300 akkordot tartalmaztak, a vegyes tanítóhalmaz esetén fele-fele arányban. A teszt eredménye a 3. táblázatban látható.

Teszthalmaz/Tanítóhalmaz	Zajmentes	Zajos	Zajmentes és Zajos
Zajmentes	96.0%	95.0%	96.0%
Zajos	88.3%	94.0%	92.7%

3. táblázat: a megfelelő tanítóhalmaz kiválasztása

Látható, hogy a zajmentes tanítással érünk el a legrosszabb eredményt. A másik kettővel nagyjából azonos az összes hibák száma (0.05*300+0.06*300 és 0.04*300+0.073*300). Ezek után a szerző azzal a feltételezéssel élt, hogy a kevert

halmazzal való tanításból kapott modell kevésbé függ a zajtól, így azt részesítette előnyben.

Osmalskyj ezek után a kevert halmazzal k=10-szeres keresztvalidálást végzett. A keresztvalidálásnál az egyes, meglévő mintákat k diszjunkt részhalmazra osztjuk. Ebből k-1 darab részhalmazt tanításra, a maradék egy részhalmazt pedig tesztelésre használjuk. Ezt a módszert megismételjük az összes többi részhalmazra úgy, hogy mindig más-más részhalmaz lesz a teszthalmaz, a többit pedig tanításra használjuk. [12].

Végül az így kapott neurális hálózatot a második nagyobb adathalmazzal tesztelte, amelyben a különböző hangszerek akkordjai voltak felvéve. Ez az adathalmaz teljesen független azoktól, amelyeket a neurális hálózat tanításához használt fel. A teszt eredményei a mintamegfeleltetéses algoritmus eredményeivel összehasonlítva az alábbi táblázatban találhatók.

Hangszer	Legközelebbi szomszéd módszer	Neurális hálózat
Gitár	8%	1%
Zongora	20%	13%
Hegedű	19%	5%
Tangóharmonika	32%	4%

4. táblázat: Osmalskyj egyhangszeres akkordfelismerésének az eredménye

5 Időbeli valószínűségi következtetés

5.1 A témakör relevanciája

Az akkordfelismerést minél pontosabban akarjuk végezni, annál inkább kevésnek bizonyulnak a pusztán jelfeldolgozással kapcsolatos módszerek. A szakirodalomban az újabb és hatékonyabb megközelítések szerint az akkordfelismeréshez valószínűségi modellt kell alkalmaznunk. Az alapkoncepció az, hogy az akkordra úgy tekintünk, mint egy diszkrét valószínűségi változóra, ami az időben gyorsan változhat, és az aktuális állapot erősen korrelál a korábbi, későbbi állapotokkal, illetve előzetes megfigyelésekkel, amik egyfajta szabályrendszerként a zene természetét írják le. Később látni fogjuk, hogy ez a szabályrendszer csak egy kiindulópont, ami az aktuális megfigyelések függvényében dinamikusan változhat. Bemutatásra kerül több, valószínűségi modellt felhasználó algoritmus, amely időbeli megfigyelésekből, és előzetes statisztikákból képes jobb becslést adni az egyes időpontokhoz tartozó valószínűségi eloszlásokra.

A munkám során Bello és Pickens akkordfelismeréssel foglalkozó cikkéből [3] indultam ki. Itt találkoztam először a Rejtett Markov Modell alapú megközelítéssel, és az Elvárásmaximalizációs algoritmus fogalmával. Ezen a szálon elindulva irodalomkutatást végeztem, mely során betekintést nyertem a valószínűségszámításnak egy ma is dinamikusan fejlődő területébe, az időbeli valószínűségi következtetések témakörébe. Ehhez nagy segítségemre volt a *Mesterséges Intelligencia Modern megközelítésben* c. könyv [19], ami többek között ennek a témakörnek az alapjait tárgyalja. A dolgozat *4.2-4.3* pontjában alapvetően erre támaszkodtam. A következő fejezetekben az időbeli valószínűségi következtetés számomra releváns részeit mutatom be.

5.2 Alapfogalmak

Elsőként ismerkedjünk meg az ágens fogalmával. *„Az ágens egy olyan autonóm működő program vagy gép, mely érzékelői segítségével érzékeli a világ aktuális állapotát és beavatkozói segítségével változtat rajta”*. Ez az ágens megfigyeléseket, méréseket végez, és a környezet aktuális állapotát akarja nyomon követni. A zajos környezet, részleges érzékelés, illetve az a bizonytalanság, ahogy a környezet az idő függvényében változik, ezt nem teszi lehetővé determinisztikus módon, így az ágens a környezet aktuális állapotáról csak egy valószínűségi becslést tud adni. Vegyünk egy példát: van egy cukorbeteg ember, akiről az orvos szeretné megállapítani, hogy éppen milyen állapotban van. Ehhez rendelkezésünkre állnak bizonyítékok: jelenlegi inzulinadagok, élelmiszer-bevitel, vércukorszint és inzulinszint mérésének az eredménye. Ezen megfigyelések alapján az orvos becslést ad a beteg aktuális állapotára. A vércukorszint, inzulinszint, stb. időben változhat, így a páciens aktuális állapota is.

Most akkor nézzük meg, hogyan lehet ezt matematikailag modellezni. A környezet az időben változik. Az időt kezelhetjük folyamatosnak, vagy diszkrét pillanatfelvételek sorozatának. Vannak változóink, ebből bizonyosak megfigyelhetőek

(ezen halmaz jelölése: E_t), némelyek viszont nem (ezt pedig X_t-vel jelöljük). Az egyszerűség kedvéért úgy vesszük, hogy X_t, és E_t nem változik (az akkordfelismerésre ez igaz). Ezen változók minden pillanatfelvételnél vagy időpontban felvesznek valamilyen értéket. A nem megfigyelhetőekre csak valószínűségi becslést tudunk adni. A t időpillanatbeli megfigyelésnél $E_t=e_t$ jelölést használjuk, azaz e_t a konkrét megfigyelés.

Mi csak olyan modellel fogunk foglalkozni, ahol az egyes megfigyelések rögzített, véges időintervallumokon történnek, azaz az időpillanatokat felcímkézhetjük egész számokkal. Feltesszük továbbá, hogy az állapotsorozat t=0-tól kezdődik, míg a bizonyítékok csak a t=1-től kezdenek beérkezni. Ezenkívül az $a:b$ jelölést használjuk az adott változó a-tól b-ig tartó sorozatának a jelölésére [19].

5.3 Következtetés időbeli modellekben, rögzített modell paraméterek mellett

Ebben a részben azt fogjuk vizsgálni, hogyan lehet becsülni az egyes időpontok feletti valószínűségi eloszlást. Ehhez két algoritmust vizsgálunk meg. Ebből az egyik az éppen számított állapothoz képest csak a múltat veszi figyelembe, míg a másik a jobb valószínűségi becslés érdekében a jövő alakulását is beleviszi, mint egy visszafelé ható befolyásoló tényező. A legpontosabb valószínűségi becslést a jövő ismeretét is felhasználó algoritmus fogja adni. Ezután egy olyan algoritmust vizsgálunk meg, amellyel az időben változó, nem megfigyelhető paraméter legvalószínűbb időbeli sorozatát kapjuk meg.

5.3.1 Szűrés

Ezzel tudjuk meghatározni a t időpillanatig történő megfigyelések alapján a jelenlegi állapot feletti a posteriori eloszlást, feltéve, hogy t=1-től érkeztek a bizonyítékok. A szűrés egyszerűen, online módon is elvégezhető az előző állapotig tartó szűrés eredményének a felhasználásával. Matematikailag a következőképpen néz ki [19]:

$$P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t})), \quad (21)$$

azaz a t+1-edik állapot valószínűségi eloszlása függ a t-ediktől, illetve az új bizonyítékváltozótól.

Ahhoz, hogy f függvényt felírhassuk, felhasználjuk a modellünk feltételezéseit, és egyéb matematikai módszereket:

1. Felosztjuk a bizonyítékot két részre ($e_{1:t+1} \rightarrow e_{1:t}, e_{t+1}$)

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}|e_{1:t}, e_{t+1}) \quad (22)$$

2. Felhasználjuk a Bayes-szabályt [19]:

$$P(Y|X,e)=P(X|Y,e)P(Y|e) /P(X|e) \quad (23)$$

Ez alapján a kifejezésünk:

$$\alpha\, P(e_{t+1}|X_{t+1}, e_{1:t})\, P(X_{t+1}|e_{1:t}) \quad (24)$$

alakba írható át, ahol α egy normalizációs konstans

3. Felhasználjuk a modellünknek a bizonyítékra vonatkozó Markov-tulajdonságát (*„feltesszük, hogy a bizonyítékváltozók egy t időpillanatban csak az aktuális állapottól függnek”*) [19]:

$$P(E_t|X_{0:t},E_{0:t-1})=P(E_t|X_t) \quad (25)$$

Ezen tulajdonság alapján a további módosítás:

$$\alpha\, P(e_{t+1}|X_{t+1})\, P(X_{t+1}|e_{1:t}) \quad (26)$$

Ezen formából jól látható, hogy az aktuális állapot becslését megkaphatjuk a következő tagokból:

- Frissítés új bizonyítékkal: $P(e_{t+1}|X_{t+1})$
- Következő állapot egylépéses előrejelzése: $P(X_{t+1}|e_{1:t})$
- Normalizációs konstans, ami azt biztosítja, hogy az X_{t+1} állapot feletti valószínűség-vektor összege 1 legyen: α

4. $P(X_{t+1}|e_{1:t})$ tagot x_t lehetséges állapotai szerint összegezzük:

$$P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t, e_{1:t}) P(x_t|e_{1:t}) \quad (27)$$

5. Kihasználva a Markov-tulajdonságot [19]:

$$\alpha P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t}) \quad (28)$$

Ezzel megkaptuk a kívánt rekurzív képletet. Ezen képletből a modellünk paramétereinek ismeretében már kiszámíthatjuk a következő állapot szűrt valószínűségi eloszlását. Az egyes szorzótényezők rendre:

- Normalizációs konstans
- Az érzékelőmodellből kinyerhető információ
- Az állapotátmenet modell
- A *t* időpillanatbeli állapot szűréssel kapott valószínűségi eloszlása

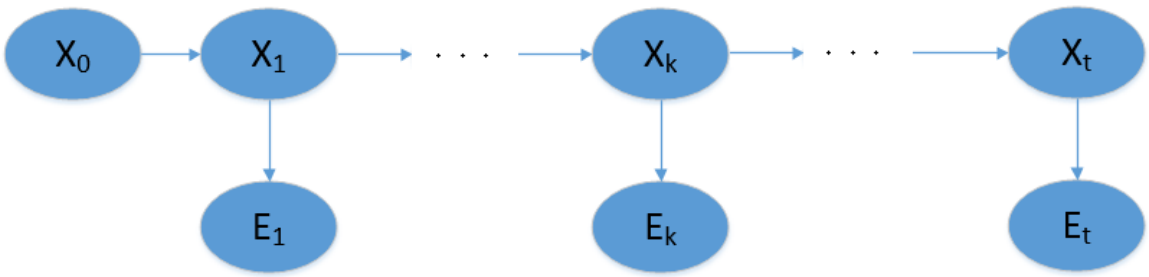
$P(x_t \mid e_{1:t})$ becslésre gondolhatunk úgy, mint egy előre haladó üzenetre (jelöljük: $f_{1:t}$ – vel), amit az új bizonyíték ismeretében minden egyes átmenetnél frissítünk. Így felírva a folyamat [19]:

$$f_{1:t+1} = \alpha * ELŐRE(f_{1:t}, e_{1:t}) \quad (29)$$

Az előre üzenetek kiszámításához szükséges definiálni az előre üzenetek kezdeti értékét, azaz az $f_{1:0}$-t. Az $f_{1:0}$-t a rejtett változó kezdeti valószínűségi eloszlásaként definiáljuk. Ezen vektort megadhatjuk előzetes statisztikák alapján [19].

5.3.2 Simítás

A simítással egy adott időpillanat felett jobb valószínűségi eloszlást tudunk meghatározni, ugyanis ez az algoritmus felhasználja az adott időpillanat után történt megfigyelések információit is. A szűréshez hasonlóan ezt is matematikai formába akarjuk önteni. Ehhez ismét fel fogjuk használni a modellünk tulajdonságait, illetve egyes matematikai összefüggéseket.



8. ábra: a simítás egy k időpillanatbeli a posteriori valószínűségi eloszlást ad meg, felhasználva a teljes, 0 és t időpillanatok közötti megfigyeléseket [19]

Első lépésként bontsuk szét a valószínűség kiszámítását két részre. Az egyikben csak az adott időpillanatnál nem későbbi (az indexe *k* vagy annál kisebb) bizonyítékok szerepelnek, míg a másikban csak a későbbiek (az indexe *k*-nál nagyobb, de *t*-nél kisebb).

1. Írjuk fel a valószínűséget úgy, hogy a bizonyítékváltozók két csoportja külön látszódjon a feltételben

$$P(X_k|e_{1:t}) = P(X_k|e_{1:k}, e_{k+1:t}) \quad (30)$$

2. A Bayes-tételt [19] felhasználva átalakíthatjuk a szétbontott formába:

$$\alpha\, P(X_k|e_{1:k})\, P(e_{k+1:t}|X_k, e_{1:k}) \quad (31)$$

3. Itt ismét felhasználjuk a feltételes függetlenséget, amit a feltételeztünk a modellünkre

($P(E_t|X_{0:t},E_{0:t-1})=P(E_t|X_t)$ felhasználásával [19]):

$$\alpha\, P(X_k|e_{1:k})\, P(e_{k+1:t}|X_k) \quad (32)$$

4. Írjuk fel a képletet az alábbi jelöléssel:

$$\alpha\, f_{1:k}\, b_{k+1:t}, \quad (33)$$

ahol definiáltuk $b_{k+1:t}$–t, mint hátra üzenetet.

Most látható, hogy a simítással kapott valószínűségi eloszlás a hátra üzenettel, mint szorzótényezővel tér el a szűrt eloszlástól.

Második lépésként hozzuk olyan alakra a hátra üzenetet, hogy a modellünk paramétereivel könnyedén tudjuk vele számolni.

1. Írjuk fel a hátra tagot olyan formában, ahol az x_{k+1} szerint összegzünk

$$P(e_{k+1:t}|X_k) = \sum_{x_{k+1}} P(e_{k+1:t}|X_k, x_{k+1})P(x_{k+1}|X_k) \quad (34)$$

2. Majd ismét, a modellünkre igaz feltételes függetlenség miatt:

$$\sum_{x_{k+1}} P(e_{k+1:t}|x_{k+1})P(x_{k+1}|X_k) = \sum_{x_{k+1}} P(e_{k+1}, e_{k+2:t}|x_{k+1})P(x_{k+1}|X_k) \quad (35)$$

3. Végül mivel feltételezzük, hogy e_{k+1} és $e_{k+2:t}$ feltételesen függetlenek x_{k+1} feltétek mellett [19]:

$$\sum_{x_{k+1}} P(e_{k+1}|x_{k+1})P(e_{k+2:t}|x_{k+1})P(x_{k+1}|X_k) \quad (36)$$

Ezzel megkaptuk azt a képletet, amivel a modellünk paramétereinek ismeretében már kiszámíthatjuk a következő állapot simított valószínűségi eloszlását. Az egyes szorzótényezők rendre:

- Az érzékelőmodellből kinyerhető információ
- A *k*+2-edik időpillanathoz tartozó hátra üzenet
- Az állapotátmenet modell

A hátra üzenet számítása analóg az előre üzenetével. Ebben az esetben is megadhatunk egy tömör, a rekurzív számítást szemléltető alkot [19]:

$$b_{k+1:t} = HÁTRA(b_{k+1:t}, e_{k+1:t}) \quad (37)$$

A hátra üzenetek kiszámításához szükséges definiálni a hátra üzenetek kezdeti értékét, azaz az $b_{t+1:t}$-t. Ezt minden esetben egy olyan oszlopvektorral definiáljuk, amiben csak 1-es szerepel, és a hossza a lehetséges állapotok számával egyezik meg [19].

5.3.3 A modell paraméterei mátrixos alakban, egy X_t állapotváltozó esetén

A számítások könnyebb kezelhetősége érdekében a levezetett képletet mátrixos alakra hozzuk. Jelöljük X_t állapotváltozó lehetséges értékeit 1,..,S egészekkel.

Ekkor a $P(X_{t+1}| X_t)$ állapotátmenet-modell egy SxS nagyságú mátrix lesz, ezt jelöljük $\boldsymbol{T}$-vel. Ennek az egyes elemeit a következőképpen kapjuk meg: $T_{ij} = P(X_t = j|X_{t-1} = i)$. T_{ij} itt konkrétan az i állapotból a j állapotba való átmenet valószínűsége.

Az érzékelőmodellt is mátrixos alakba tudjuk hozni. Itt ismert a bizonyítékváltozó, így a modellnek csak az erre vonatkozó valószínűségeit használjuk fel, azaz a $P(E_t=e_t|X_t)$ oszlopát. Minden időpontra külön készítünk egy diagonális mátrixot, aminek a diagonálisában szerepelnek a bizonyítékváltozó által meghatározott oszlop elemei. Ezt a mátrixot jelöljük $\boldsymbol{O}_t$-vel.

Az előre és hátra üzeneteket pedig egy-egy oszlopvektorral tudjuk leírni. Így az előre és a hátra egyenletek egyszerűbb felírását kaphatjuk [19]:

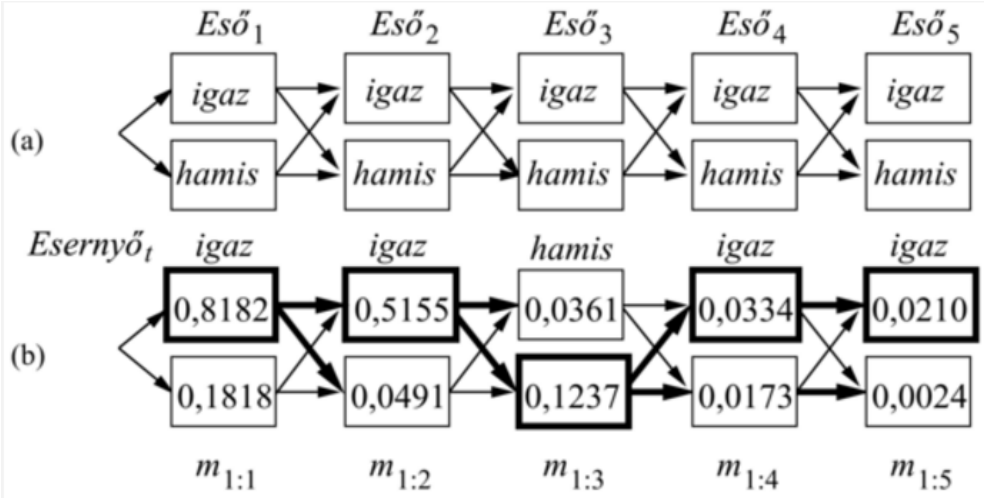
$$\boldsymbol{f}_{1:t+1} = \alpha \boldsymbol{O}_{t+1} \boldsymbol{T}^T \boldsymbol{f}_{1:t} \quad (\textit{ELŐRE-egyenlet}) \qquad (38)$$

$$\boldsymbol{b}_{k+1:t} = \boldsymbol{T}\boldsymbol{O}_{k+1}\boldsymbol{b}_{k+2:t} \quad (\textit{HÁTRA-egyenlet}) \qquad (39)$$

5.3.4 A legvalószínűbb sorozat megtalálása – Viterbi-algoritmus

Ebben a fejezetben a cél egy olyan algoritmus bemutatása, ami egy adott megfigyeléssorozatból képes megmondani, hogy mi lehetett az ezt kiváltó rejtett változó legvalószínűbb sorozata. Gondolhatnánk azt, hogy úgy képezzük a sorozatunkat, hogy először felhasználjuk a simító algoritmust, amivel megmondjuk a rejtett változó egyes időpillanatok feletti a posteri eloszlását, majd minden időpillanatnál kiválasztjuk a maximális értéket. Ez a megközelítés hibás, ugyanis a simítással ez egyes időpillanatok feletti valószínűségi eloszlást kapjuk meg, ezzel szemben a legvalószínűbb sorozat megtalálásánál az összes időpillanat feletti együttes eloszlást kell kiszámítanunk. Tehát egy másik algoritmusra van szükségünk.

Az új algoritmus egyszerűbb megértéséhez egy gráfot készítünk, aminek a csomópontjai az állapotok, az élei az állapotátmenetek. Minden időpillanathoz felvesszük az összes lehetséges állapotot és egymás alá rajzoljuk, majd a két szomszédos időpillanathoz tartozó állapotok közé úgy rajzolunk éleket, hogy *t*-1-hez tartozó összes



9. ábra: Trellis-diagram példa [19]

állapota össze legyen kötve *t*-hez tartozó összes állapotával. A gráf egy irányított gráf, a korábbi időpillanathoz tartozó állapotokból a későbbi felé mutat. Ezt az ábrát hívjuk Trellis-diagramnak. A 9. ábrán láthatunk erre egy példát. A példánk arról szól, hogy egy földalatti létesítményben vagyunk, és szeretnénk megmondani, hogy kint esett-e az eső. Ez lesz tehát a rejtett változónk. A megfigyelésünk pedig az, hogy egy ember, aki minden nap kintről érkezik, hozott-e esernyőt. Felírhatjuk tehát a lehetséges állapotokat minden időpillanatra: esett az eső (igaz), nem esett (hamis), és alá írhatjuk a megfigyelési sorozatot az öt megfigyelt napra (hozott-e esernyőt): {igaz, igaz, hamis, igaz, igaz}.

Most ebben a gráfban keressük azt az útvonalat, ami a legelejéről indul, a legutolsó oszlopban végződik. Ehhez használjuk fel a Viterbi-algoritmust, amihez szükségünk lesz a kezdeti valószínűségi eloszlásra, az állapotátmenet-modellre, az érzékelőmodellre.

A Viterbi-algoritmus azt használja ki, hogy van egy rekurzív kapcsolat az x_{t+1} és állapotokba vezető útvonalak és az x_t állapotba vezető legvalószínűbb útvonalak között. Ez matematikailag megfogalmazva [19]:

$$\max_{x_1 \dots x_t} P(x_1, \dots x_t, X_{t+1}|e_{1:t+1}) =$$
$$= \alpha\, P(e_{t+1}|X_{t+1}) \max_{x_t} \left\{ P(X_{t+1}|x_t) \max_{x_1 \dots x_{t-1}} P(x_1 \dots x_{t-1}, x_t|e_{1:t}) \right\} \quad (40)$$

5.4 Elvárásmaximalizáció – az időbeli valószínűségi modell paramétereinek az újrabecslése

Ebben a fejezetben egy olyan iteratív algoritmust vizsgálunk meg, ami a modellünk paramétereit (átmenetvalószínűségi-modell, érzékelő-modell, kezdeti valószínűségi eloszlás) javítja olyan módon, hogy az új paraméterekkel és a megfigyelési

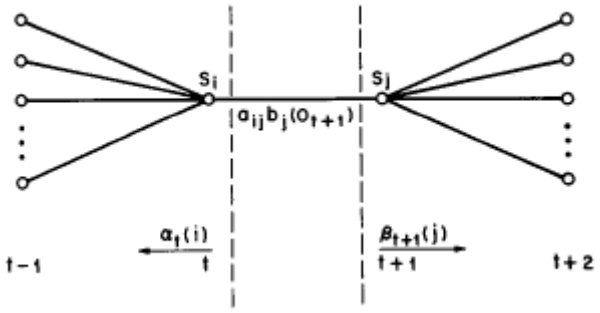
szekvenciával számított valószínűségek jobb becslései lesznek a valóságnak. A *4.4*-es fejezethez *Lawrence R. Rabiner* 1989-ben megjelent cikkét [24] vettem alapul.

Használjuk most a következő jelöléseket!

- $\alpha_t(i)$ az előre üzenet *i*-edik állapotra vonatkozó értékét a *t*-edik időpillanatban
- $\beta_{t+1}(j)$ a hátra üzenet *j*-edik állapotra vonatkozó értékét a *t*+1-edik időpillanatban
- a_{ij} az állapotátmenet mátrix *i*-ből *j*-be való átmenethez tartozó értékét
- $b_j(\boldsymbol{O}_{t+1})$ a megfigyelő modell *j*-edik állapotra vonatkozó értékét a *t*+1-edik időpontban történt megfigyelés
- π_i a kezdeti valószínűségi eloszlás

A Rejtett Markov Modell paramétereinek újrabecsléséhez szükséges definiálnunk egyes változókat. Elsőként $\xi_t(i,j)$-t definiáljuk, ami nem más, mint annak a valószínűsége, hogy a rejtett változó a *t* időpillanatban S_i állapotban van, és *t*+1 időpillanatban S_j-ben, azaz [24]:

$$\xi_t(i,j)=P(q_t=S_i, q_{t+1}=S_j \mid O, \lambda) \quad (41)$$



10. ábra: ξt(i,j) értelmezése [24]

A $\xi_t(i,j)$ változót ki tudjuk fejezni az előre és a hátra algoritmus által kapott valószínűségi eloszlások segítségével [24]:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \quad (42)$$

Ezután definiáljunk $\gamma_t(i)$-t, annak a valószínűségét, hogy t időpillanatban S_i állapotban vagyunk.

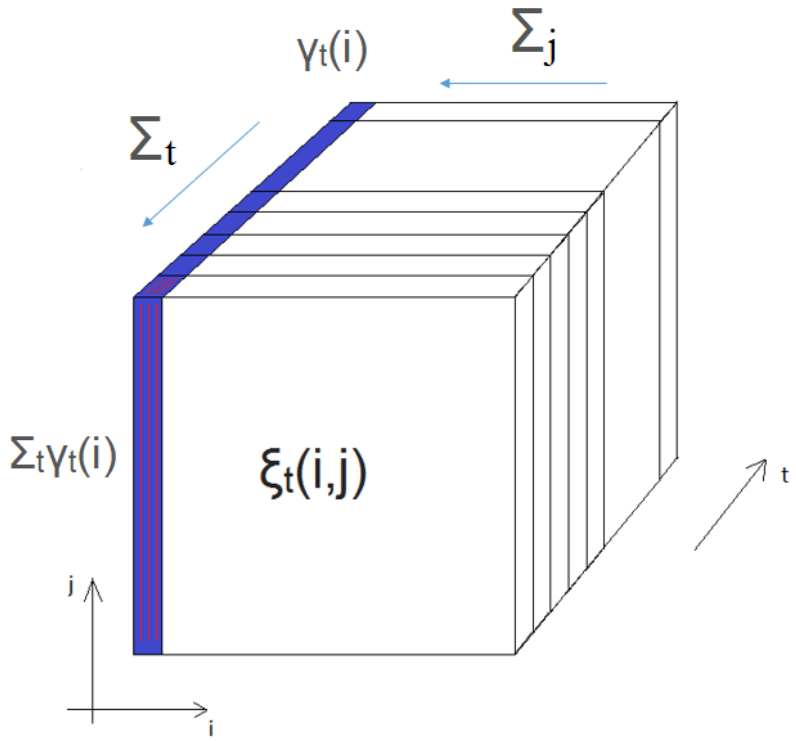
$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(\mathrm{i},\mathrm{j}) \quad (43)$$

További paramétereket kaphatunk meg, ha az idő szerint is végzünk összegzéseket:

$\sum_{t=1}^{T-1} \gamma_t(i) = az\ S_i$ állapotból való állapotátmenetek számának várható értéke

$\sum_{t=1}^{T-1} \xi_t(\mathrm{i,j})$
$=$ S_i állapotból S_j állapotba való állapotátmenetek számának a várható értéke

A 11. ábra az egyes összegzéseket mutatja be szemléletesen. Látható, hogy van a *ξ(i,j)* mátrix, ami minden egyes időpillanatban változik, így az összes számunk egy többdimenziós mátrixot alkot, amelyet az ábrán először *j* szerint, majd az idő szerint összegezzük, végül egyetlen vektort kapva.



11. ábra: a paraméterek újrabecsléséhez szükséges segédváltozók szemléltetése

Felhasználva az előbb definiált mennyiségeket újra tudjuk becsülni a modellünk paramétereit a következő módon [24]:

$\overline{\pi_\iota} = \mathrm{Az\ A_t} = 1$ időpillanatbeli várható gyakoriság $\mathrm{S_i}$ állapotban $= \gamma_1(i)$ (44)

$$\overline{a_{\iota\jmath}} = \frac{S_i \text{ állapotból } S_j \text{ állapotba való állapotátmenetek számának a várható értéke}}{Az\ S_i \text{ állapotból való állapotátmenetek számának várható értéke}} = \frac{\sum_{t=1}^{T-1} \xi_t(\mathrm{i,j})}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (45)$$

$$\bar{b_j}(k) = \frac{\text{A várható értéke az időnek, amíg j állapotban van } v_k \text{ megfigyelés mellett}}{\text{Az } S_i \text{ állapotból való állapotátmenetek számának várható értéke}} =$$

$$= \frac{\sum_{\substack{t=1 \\ O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \quad (46)$$

Az újrabecslést iteratív módon folytathatjuk addig, amíg az újrabecslés során keletkezett új paraméterek az eggyel korábbi iterációtól már nem térnek el jelentősen.

6 Akkordfelismerés Rejtett Markov Modelles megközelítésben

6.1 Az akkordfelismerés RMM modelljének definiálása

Mint korábban említettük, az akkordfelismerésre a pusztán jelfeldolgozási módszerek nem elég hatékonyak. Annak érdekében, hogy pontosítsuk az egyes időpillanatokhoz tartozó döntésünket, az akkordfelismerést valószínűségi becslésként kezeljük. A zene természetéből adódóan ezen feladathoz a Rejtett Markov Modelles megközelítés a célszerű, mert a rejtett változó, ami maga az akkord, az időben gyorsan változhat.

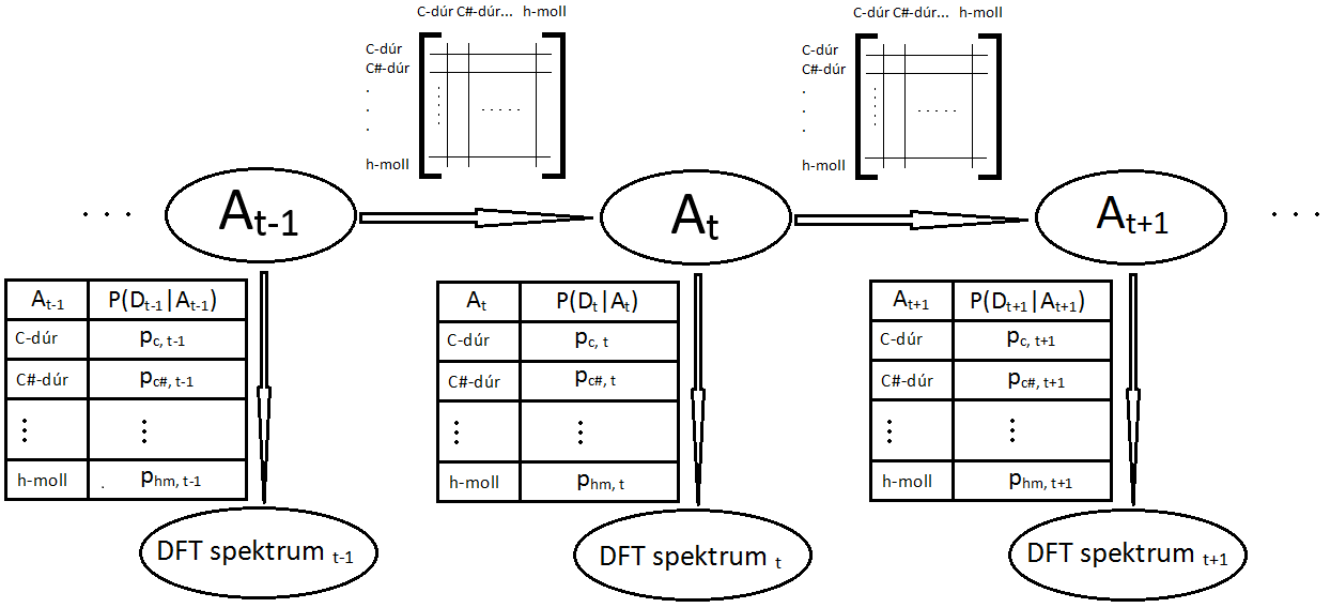
Először szükséges a modell paramétereit definiálnunk. Ezt a 12. ábrán tudjuk nyomon követni. Az akkord a rejtett változónk, aminek nem ismerjük az értékét. Az időt felosztjuk kis szeletekre. Ahhoz, hogy konkrétan meghatározzuk az időablak hosszát a következő információt használjuk fel. Tudjuk, hogy az akkord általában az ütem alapján változik, és meglehetősen ritka, hogy ez a változás a negyedhangnál gyorsabban történne. Ha az időablakokat a ritmussal szinkron választjuk meg, akkor kisebb az esélye annak, hogy az időablak közepén történik az akkordváltás. Ezért mindenekelőtt szükséges a zene tempójának a meghatározása is. Ezután az abból számított periódusidő alapján feldaraboljuk a zenét kis, egymást némileg átfedő szakaszokra. Ezen szakaszokban tehát úgy vesszük, hogy az akkord nem változik. Az ábrán a t időpillanatban az akkordot A_t-vel jelöljük.

Minden időpillanatban történik egy megfigyelés is. Ez a megfigyelés a zenei jel adott időpillanathoz tartozó DFT spektrum. Ebből fogjuk számítani az érzékelőmodell segítségével a $P(D_t|A_t)$ valószínűséget.

Munkám során csak az egyszerűbb dúr és moll jellegű hármashangzatok felismerésével foglalkozok. Ezek szerint az akkord változó összesen 24 lehetséges állapotot vehet fel. A t-1 és t időpillanatokhoz tartozó állapotok közti átmenetek valószínűségeit a 24x24-es állapotátmenetvalószínűségi mátrix adja meg. Ezt egy előzetes statisztika segítségével inicializáljuk. A statisztikát úgy készítjük, hogy megvizsgálunk több zenét, és a benne történő akkordváltásokat számoljuk. Minél gyakoribb egy akkordváltás, annál nagyobb értéket rendelünk a mátrixban annak megfelelő eleméhez.

Az érzékelőmodell minden időpillanatban megmondja, hogy mi a valószínűsége annak, hogy a megfigyelt DFT spektrumot látjuk, feltéve, ha a rejtett változó adott, ismert érték, röviden a $P(D_t|A_t)$ valószínűséget adja meg. Ez minden időpillanatban egy 24 hosszúságú vektor, ami minden állapotfeltételre megmondja ezt a valószínűséget. Bonyolultsága miatt külön, a *6.4* fejezetben tárgyaljuk.

Az akkordok sorozata az *5.2*-es fejezetben leírtak alapján t=0-ról indul, míg a megfigyelési szekvencia t=1-ről. A t=0-hoz tartozó akkord valószínűségi eloszlásának a kezdeti valószínűségi eloszlást mondjuk.



12. ábra: Akkordfelismeréshez készített Rejtett Markov Modell

6.2 Az időszeletek meghatározása

Az időszeletek meghatározásának az az alapvető célja, hogy ezáltal el tudjuk választani az időegységekhez tartozó megfigyelést. Az egyes időszeleteket egy diszkrét időpillanat-sorozattal tudjuk felcímkézni. A zenei jel ezen módon történő feldarabolását a tempó alapján végezzük. A feldarabolt részekből DFT spektrumot számítunk, aminek a segítségével következtetni tudunk arra, hogy az adott időszeletben milyen akkord szólalhatott meg.

A zenékre általában igaz, hogy a tempójuk kis mértékben ingadozik, sőt ritkább esetben ugyan, de az is elképzelhető, hogy a zene tempója megváltozik. Emiatt a tempót nem a teljes zenéből számítjuk globálisan, hanem először szétdaraboljuk a zenét 10 másodperces részekre, majd ezen részekben külön-külön határozzuk meg ezt az értéket. A tempódetektálást nagyjából 10 másodperces szakaszokon érdemes végezni. Tapasztalataim alapján ennél kisebb zenerészletre az általam implementált algoritmus nem mindig működik megfelelően, és ezzel nagyjából nyomon követhető a tempó kisebb-nagyobb mértékű változása.

A 10 másodperces szakaszoknál ezenkívül meghatározzuk az első beütés helyét. Ez az a pont, amit az első időszelet kezdőpontjának vesszük, és ahonnan kiindulva, a tempónak megfelelő periódusidővel haladva előre megkapjuk a további időszeletek határait. Ezenkívül a tempóhoz való jobb szinkronizáció érdekében felhasználunk még egy, a *6.2.2* fejezetben ismertetett algoritmust.

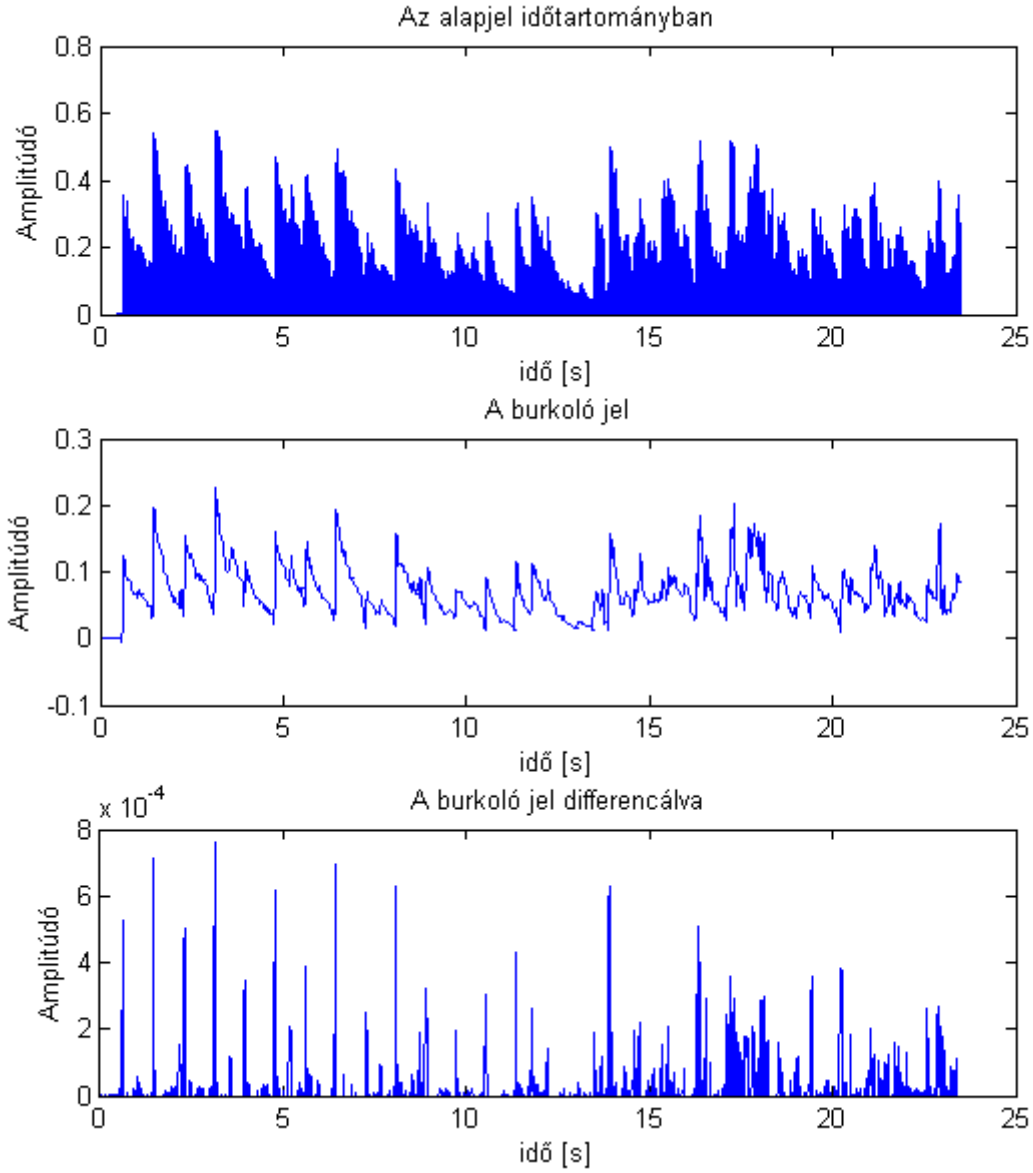
6.2.1 Tempódetektálás

Az akkordok általában a zene tempójára változnak, és a negyedhangnál ritkábban szoktak történni váltások. Ha a tempó alapján vágjuk szét a zenei jelet, akkor azzal csökkenthetjük a harmóniák időbeli átlapolódásából származó hibát. A tempódetektáló algoritmust [20] felhasználásával készítettem.

6.2.1.1 A tempómeghatározás előkészítése

A felhasznált tempódetektáló módszerben azt használjuk ki, hogy a zenékben periódikusan történnek beütések, amelyek az ütemet adják meg. Ezen beütéseknél a zene időtartománybeli jelében jól megfigyelhető felfutás van. Ezen felfutások távolságából fogjuk kiszámítani a tempót.

A jelet a gyorsabb feldolgozás érdekében decimáljuk, minden hatodik elemét tároljuk csak el. Ezután a jel abszolút értékét vesszük, majd azt egy aluláteresztő szűrővel szűrjük. A feladat megoldásához a választás egy 5-öd fokú *Butterworth*-szűrőre esett 220.5 Hz töréspontí frekvenciával. A zene általában – főként a ritmus hangszerek miatt - lecsengő jellegű jeleket tartalmaz, aminek a szűrés hatására csak a burkológörbéje marad meg. Ennek a görbének a beütéseknél meredek felfutása van. Ha a jelet differenciáljuk, akkor a differenciált jel pozitív értékei a felfutást, a negatív értékei a lefutást fogják jelenteni. A negatív értékek irrelevánsak, így azokat nullával tesszük egyenlővé. Ezen folyamat látható a 13. ábrán.



13. ábra: a tempódetektálás folyamatának a differenciálásig tartó része

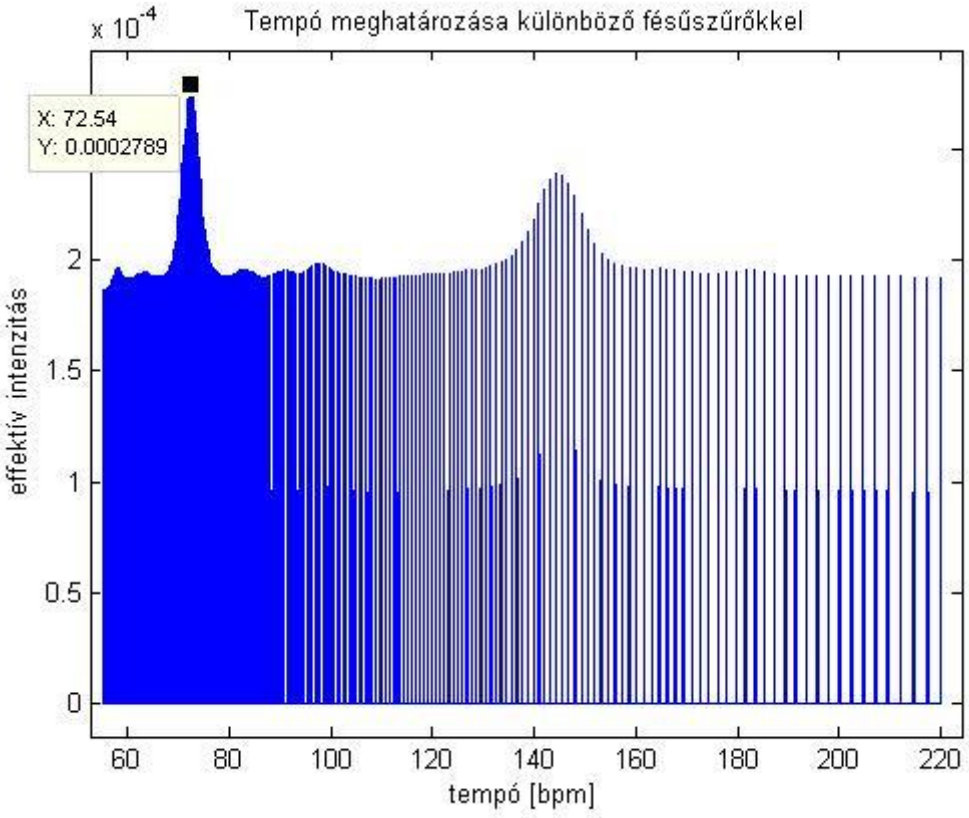
6.2.1.2 A tempó meghatározása

A differenciált jelből egy fésűszűrő segítségével határozzuk meg a zenerészlet tempóját. Mivel a zenék tempója 55 bpm és 220 bpm között szokott mozogni, ezért mi is csak ebben a tartományban vizsgálódunk. A fésűszűrő egymástól egyenlő távolságra lévő egységimpulzusok sorozata, aminek a távolságait az éppen vizsgált tempóhoz tartozó periódusidő adja meg. Például ha 44100 Hz mintavételi frekvencián, 6-tal decimálva *t* bpm tempót vizsgálunk, akkor 44100/6*60/*t* távolságra lesznek az egységimpulzusok egymás mellett. A fésűszűrőben található egységimpulzusok száma szabadon választható. Az általam írt alkalmazás már 3 egységimpulzussal is megfelelően működött.

A különböző tempókhoz tartozó fésűszűrőkkel konvolváljuk a differenciált jelet, majd az így kapott jel effektív értékét vesszük. A maximális effektív értékhez tartozó fésűszűrőnek megfelelő tempót fogjuk választani a zene tempójának. Az algoritmus azért

ad helyes eredményt, mert a konvolúciót értelmezhetjük úgy is, mint két függvény közti korreláció mértékét. A fésűszűrő minél jobban hasonlít a differenciált jelre (azaz a zene tempója szerint vannak az egységimpulzusok távolságai) annál nagyobb lesz a konvolváltjuk effektív értéke.

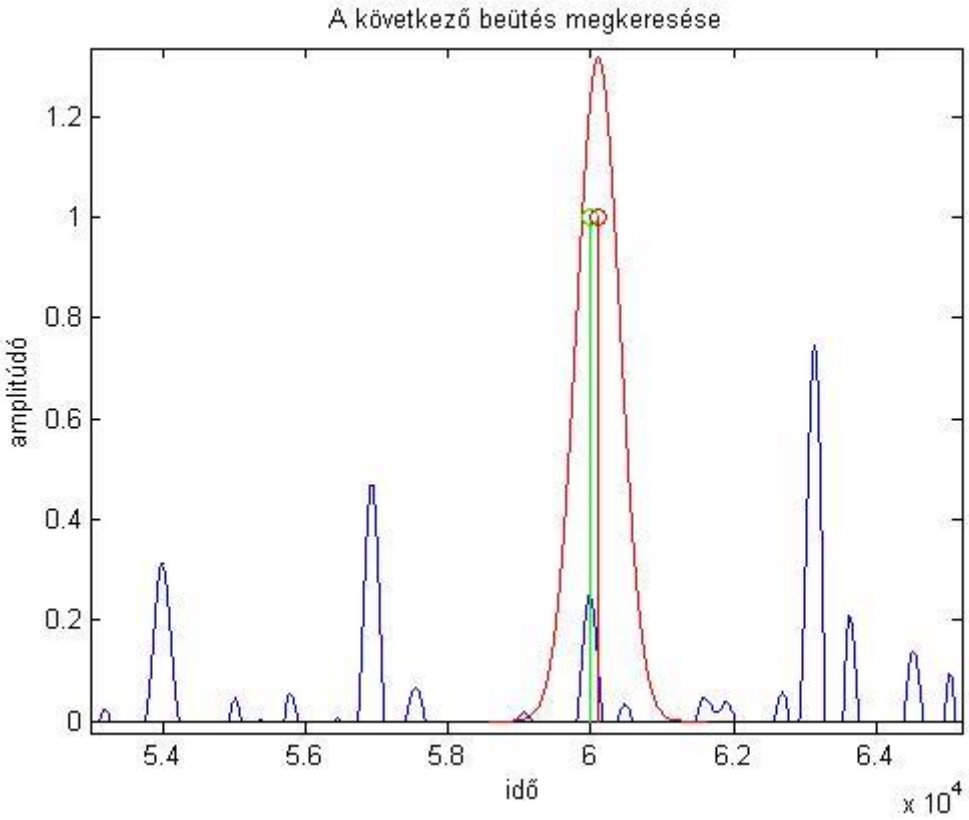
Az 14. ábrán látható egy konkrét példa, ahol megfigyelhetjük, hogy az egyes tempókhoz mekkora effektívértékek tartoznak. Ez a *Beatles* együttestől a *Let it be* c. dal első 10 másodpercéből lett számítva. A számítások csökkentése érdekében a tempót csak egy előre meghatározott felbontásban vizsgáljuk. Itt fontos átgondolni, hogy milyen pontossággal akarjuk meghatározni azt. Ebben az esetben 44100/6*60/55-höz közeli 2004-től a 44100/6*60/220-hez közeli 8004-ig állítottuk be a fésűszűrő csúcsának távolságát 25-ösével lépkedve.



14. ábra: a tempók és a hozzájuk tartozó effektív intenzitások (Beatles – Let it be)

6.2.2 Szinkronizáció

A tempó meghatározása után ahhoz, hogy megfelelően daraboljuk szét a zenét, meg kell találnunk az első időpillanatot, amikor biztosan történt egy beütés. Ehhez először a differenciált jelnek megkeressük a maximális értékét. Itt nagy valószínűséggel van egy beütés, ezért innen kiindulva keressük meg az első beütést. Jelöljük a helyét t_{max}-szal. Ezután a meghatározott tempóból periódusidőt számítunk, jelöljük ezt T-vel. A t_{max} pozícióból indulva időben visszafelé lépünk T-t. Legyen ez a hely t_1. Nagy valószínűséggel a differenciált jelnek ott lesz egy magas csúcsa, de ez a tempóingadozás miatt egy kicsit eltérhet. Mi viszont azt szeretnénk, hogy minden esetben - még ha a tempó ingadozik is - az egyes időszeletek határa a közelben lévő differenciált jel csúcsa legyen. Viszont ha magas csúcs nincs a közelben, akkor csak kisebb csúcshoz rendelje hozzá. Ezen elvárásainkat matematikailag a következőképpen fogalmazzuk meg. Definiálunk egy egydimenziós Gauss-görbe függvényt, aminek a várható értéke t_1 a szórása pedig $T/10$. A szórás értékét kísérletileg állítottam be. Ezután a differenciált jel és a Gauss-görbe szorzatát vesszük. Az így kapott függvény maximumát fogjuk venni az eggyel korábbi beütésnek. Ezt az algoritmust iteratívan folytatjuk addig, amíg a következő lépés már nem



15. ábra: a következő beütés megkeresése

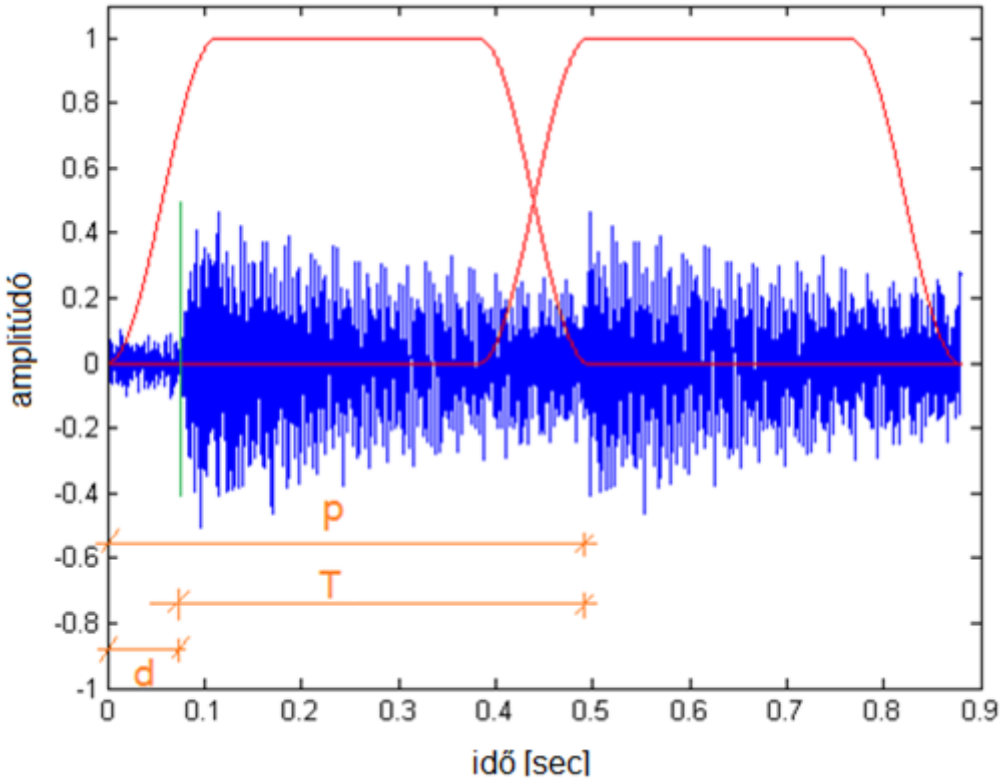
lenne benne a 10 másodperces szakaszban. Azt a csúcsot vesszük az első beütés helyének, ahol az algoritmus megáll. Az algoritmus működését a 15. ábra szemlélteti. A kék színű csúcsok a differenciált jel részei, a piros színű görbe a Gauss-görbe eloszlás. A piros színű egyes jelöli a következő beütés várható értékét, a zöld színű pedig a meghatározott beütést.

Az egyes időszeletek határát is ilyen módon határozzuk meg, csak visszafelé, azaz időben előre. Az első beütés helyétől indulunk, és *T*-ket lépkedünk előre, majd a Gauss-görbe alapján a legvalószínűbbet eltároljuk, mint a következő időszelet határt.

Fontos még megjegyezni, hogy ha a 10 másodperces részekre a tempómeghatározást úgy végezzük, hogy a mindegyik részt külön-külön szűrjük a Butterworth-szűrővel, akkor a szűrő tulajdonságából adódóan a szűrt jel legelején lesz egy felfutás, ami nem a zenei jelben lévő beütés miatt van. Ha ezt a jelet differenciáljuk, akkor az első beütést tévesen határozzuk meg, így egész 10 másodperces szakasz nem a negyedek alapján lesz feldarabolva, ezenkívül a tempófelismerésben is okozhatunk pontatlanságot. Erre azt a megoldást választottam, hogy az egész zenét egyszerre szűrtem meg, azt daraboltam fel 10 másodperces szakaszokra, és abból számítottam mind a tempót, mind pedig az első beütés helyét.

6.2.3 Ablakozás

Az egyes diszkrét időpillanatokhoz tartozó zeneszakaszokat nem csak a két beütés közötti résznek definiáljuk, hanem belevesszük ebbe a beütés előtti rövid szakaszt. Így időben némileg átfedett megfigyeléseink vannak. Ezt a módszert 16. ábrán láthatjuk. A felismert tempót *T*-vel jelöljük, a beütés előtti rövid szakaszt *d*-vel, és az egy teljes zenerészlet hosszát *p*-vel. A zöld egyenes vonal mutatja a beütés helyét az első zenerészletben.



16. ábra: az egyes időszeletek feldolgozása

6.3 Az akkordfelismerés RMM modelljének a kezdeti paraméterei

6.3.1 Állapotátmenetvalószínűségi mátrix

A kezdeti állapotátmenetvalószínűségi mátrix készítésének alapkoncepciója az, hogy több zene akkordmenetét megvizsgáljuk, és ebből feljegyezzük az egyes akkordátmeneteket, amiből statisztikát készítünk. Ha találunk egy *i*-edik állapotból *j*-edik állapotba való akkordváltást, akkor az állapotátmenet mátrixunk T_{ij}-edik eleméhez hozzáadunk egyet. A végén normálnunk kell az egyes sorokat, hogy valószínűségeket kapjunk. Ehhez C. Harte által gyűjtött adatbázist használtam fel, mely az elérhető az interneten [22][23]. Ebben 141 Beatles dalnak az akkordmenete van ilyen módon feldolgozva.

6.3.2 Kezdeti valószínűségi eloszlás

Ezt jelfeldolgozási módszerrel határozzuk meg. Az érzékelőmodell t=1 időpillanathoz tartozó értékét használtam fel, mint kezdeti valószínűségi eloszlás. Megjegyzendő, hogy ennek a paraméternek a megválasztása nem sokat számít, Sheh és Ellis véletlenszerűen inicializálta [7], míg Bello és Pickens 1/24-ed valószínűséget társított a vektor minden egyes eleméhez [3].

6.4 Az érzékelőmodell

Az érzékelő modell meghatározásánál *a* $P(D_t|A_t)$ valószínűségi eloszlást meghatározó módszert keressük. A megfigyelésünk egy DFT spektrum, melyben az intenzitások folytonos értéket vehetnek fel.

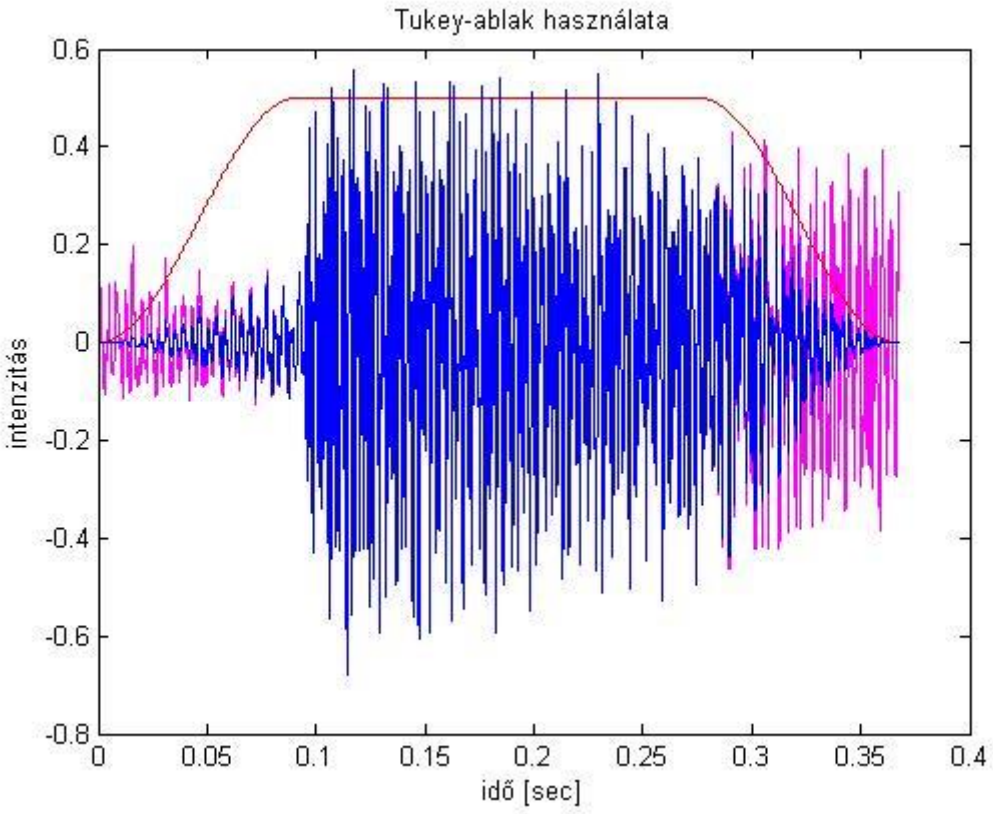
Az érzékelőmodellre a szakirodalomban többféle megközelítéssel is találkozhatunk, mint például az egyszerű Gauss-modell [3,7]. Ebben a dolgozatban ehelyett saját, jelfeldolgozáson alapuló módszert fogok alkalmazni.

Az érzékelőmodell a t diszkrét időpillanathoz a következő módszerrel rendeli hozzá a $P(D_t|A_t)$ 24 elemű valószínűségvektort (A_t = C-dúr, C#-dúr, … h-moll értékekre). A t időpillanatban volt egy megfigyelésünk, ami egy DFT spektrum. Ezt a spektrumot négy különböző módszerrel vizsgáljuk meg. A négy különböző módszer külön-külön ad egy $P(D_t|A_t)$ valószínűséget. Ezen valószínűségeket fogjuk összeadni, majd valószínűségi vektorra normálni. Ebből az 1-2. módszer bizonyos feltételek teljesülése mellett konkrétan kiválaszt egy akkordot és ahhoz rendel 1 valószínűséget a többihez nullát. Ha az adott módszer feltételei nem teljesülnek, akkor az a módszer nem mond semmilyen valószínűséget (csupa nulla értékű vektort ad hozzá az összevont $P(D_t|A_t)$ valószínűséghez). A 3-4. módszer minden akkordra ad egy valószínűséget. Ezt olyan módon teszi, hogy a DFT spektrumból saját jelfeldolgozási eljárást felhasználva mind a 3. mind a 4. módszer elkészíti a saját PCP vektorát.

A *6.4.1-3* fejezetekben azokat a jelfeldolgozási lépéseket vizsgáljuk meg, amiket a 4. módszeren kívül mindegyik felhasznál.

6.4.1 Ablakozás

A beolvasott rövid (0.5 másodperc nagyságrendű) zenerészletet először ablakozzuk. Ez azért fontos, mert a nem koherens mintavételezésnél a kiszámított DFT vektorban spektrumszivárgás jelensége figyelhető meg. Érdemes olyan ablakfüggvényt választanunk, amely Fourier-transzformáltjának kisintenzitású oldalhullámai vannak. Az időtartománybeli szorzás, frekvenciatartománybeli konvolúciónak felel meg, így a kisebb oldalhullámok kisebb spektrumszivárgást okoznak. Ettől viszont az egyes intenzitáscsúcsok szélesebbek lesznek, de mivel az algoritmusunkban lokális maximumokat fogunk keresni, ezért ez kevésbé probléma. Elég gyakori választás a *Hann-ablak*. Mi ennek egy kicsivel módosított verzióját használjuk, a *Tukey-ablakot,* ami a 17. ábrán látható. Ez az ablak két fél Hann-ablakból és a köztük lévő konstans 1 értékű részből áll. A korábbi, *6.2* fejezetben használt jelöléseket alkalmazva a fél Hann-ablak *d* hosszúságú, a Tukey-ablak pedig *p*. A ablak választásának oka az, hogy így a beütésnél, ahol a legintenzívebben szólal meg az akkord, ott 1 súllyal vesszük a zenei jelet, míg az előtte lévő résznél, illetve a lecsengés végénél csak kisebbel.



17. ábra: A Tukey-ablak (piros) használata, eredeti jel - magenta, ablakozott jel - kék

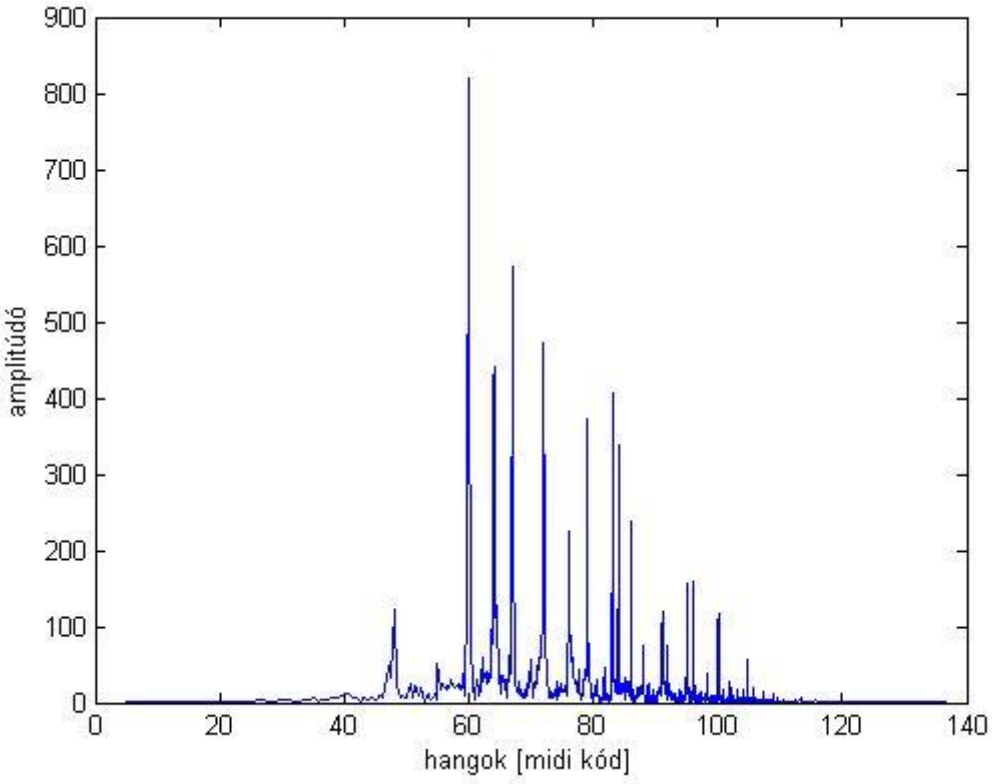
Az ablakozott jelnek ezután elkészítjük a Fourier-transzformáltját. A DFT tulajdonságaiból következően elég az így kapott tartomány felét felhasználnunk.

6.4.2 Frekvenciatengely átskálázása a standard MIDI kódra

A vizsgálatok megkönnyítése érdekében a frekvenciatengelyt átskálázzuk úgy, hogy az egyes zenei hangok frekvenciája helyén annak a standard MIDI kódja jelenjen meg. Ehhez a frekvenciakonverzióhoz a következő képletet használjuk:

$$f_{MIDI} = 12 * log_2\left(\frac{\frac{f}{440}}{(2^{\frac{1}{12}})\text{^}(-69)}\right) \quad (47)$$

A 18. ábrán egy konkrét zenerészlet a spektruma látható, ahol a frekvenciát átskáláztuk standard MIDI kódba.



18. ábra: zenerészlet spektruma, a frekvenciatengely standard MIDI kódban

6.4.3 Súlyozás

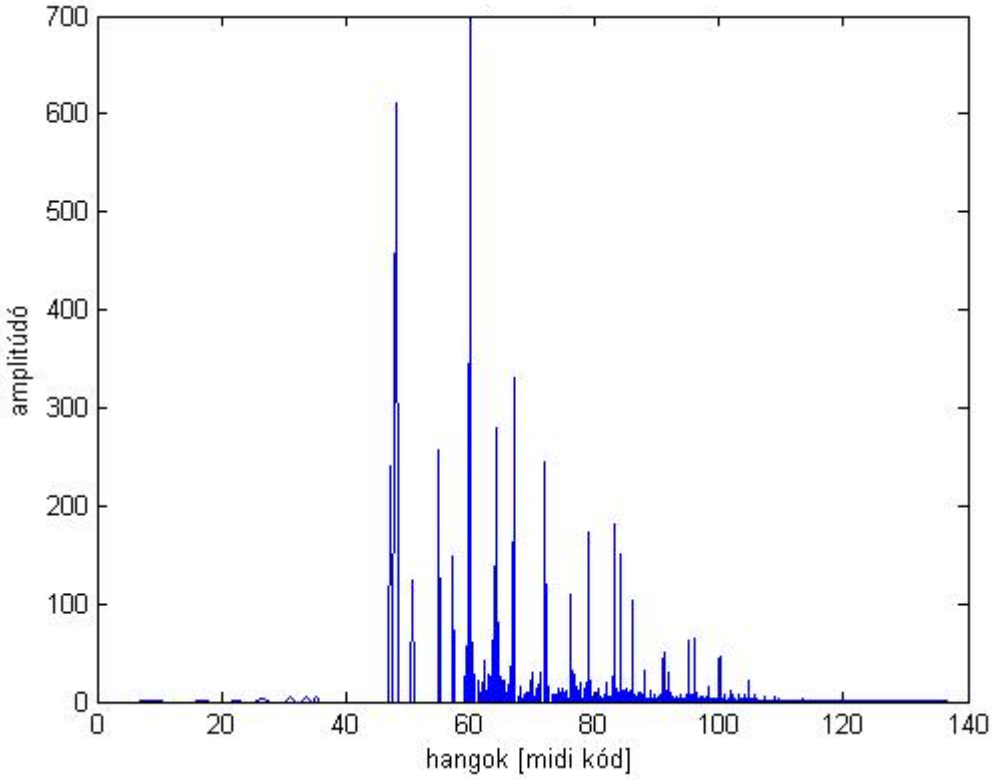
Következő lépésként megkeressük a spektrum lokális maximumait. Csak ezeket a mintákat tartjuk meg, a többit nullára írjuk át.

A zene egy rövid szakaszában lévő hangok közül az akkordra jellemző információkat a basszushangok általában jobban hordozzák, így a mélyebb hangokat erősebb súllyal vesszük figyelembe. Sok kísérletezés után a leghatékonyabbnak a következő bizonyult:

- C2 és a B3 közötti hangok: az ebben a tartományban lévő maximális intenzitás ötödénél kisebbeket elhanyagoljuk, majd a megmaradt részt 5-szörös súllyal vesszük figyelembe
- C2 alattiak: nem súlyozzuk
- B3, és az afölöttiek: elosztjuk a következő számmal:

$$\sqrt[4]{(adott\ hang\ midi\ kódja - 58)}$$

A 19. ábrán egy konkrét példát láthatunk egy zenerészlet spektrumára a súlyozás után.



19. ábra: zenerészlet spektruma lokális maximumok megtalálása, és a súlyozás után

6.4.4 Az *t* időpillanathoz tartozó valószínűségvektor meghatározása

Ezen vektor meghatározására tehát négy különféle módszert alkottam. Ezek a módszerek a megfigyelést különböző oldalról vizsgálják (fizikai, zeneelméleti megfontolások). A végeredmény az egyes módszerek által adott vektorok összege lesz. Ezzel az érzékelőmodell pontosabb valószínűséget tud mondani a megfigyelés alapján. Itt fontos megjegyezni, hogy a kiszámított valószínűségek nem a klasszikus értelemben vett valószínűségek, mivel az értékük lehet 1-nél is nagyobb. Viszont az algoritmusok, ahol felhasználjuk őket (Előre - Hátra, Viterbi, EM), azt szorzótényezőként használják fel, ahol a nagyobb szorzótényező nagyobb valószínűséget takar. Végül pedig ahol szükséges, hogy valószínűséget kapjuk, ott az α normalizációs konstanssal a vektort normálva megkaphatjuk azt.

6.4.4.1 Az 1. módszer

Az 1. módszerben a súlyozott spektrumból indulunk ki. A súlyozás után megkeressük a spektrum maximális amplitúdójú csúcsát, majd az annak a 10%-ánál

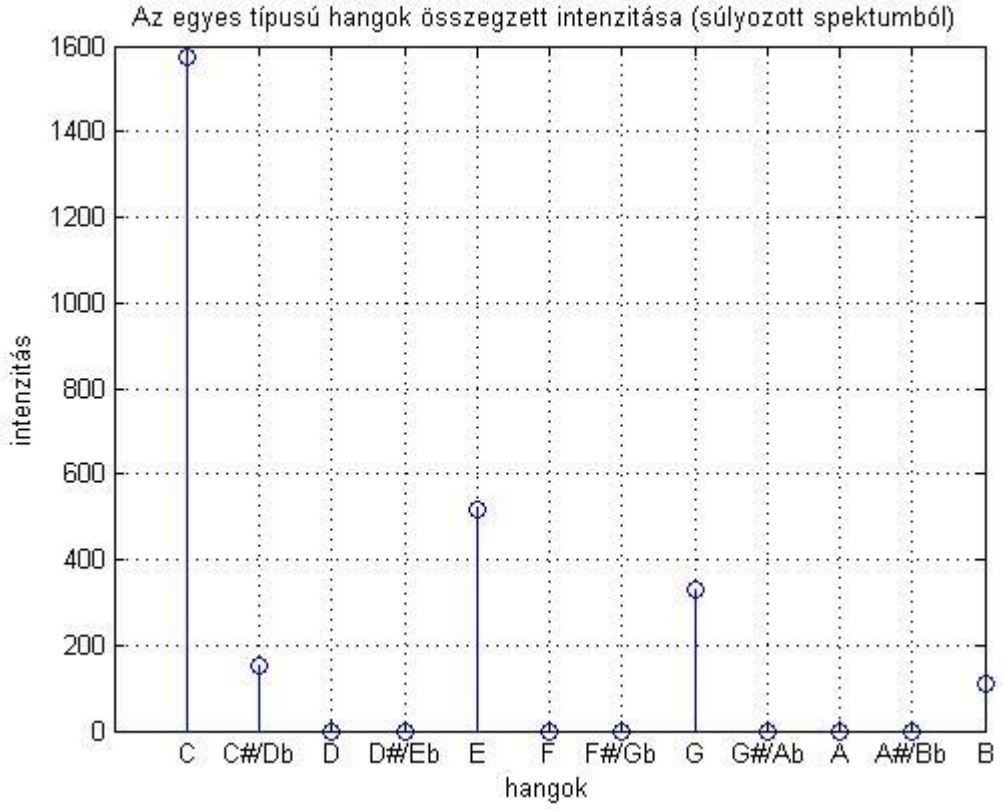
kisebb intenzitású tagokat elhagyjuk. A fennmaradó intenzitáscsúcsokhoz megkeressük, hogy melyik zenei hanghoz tartoznak. Ezek után a hangokat egy oktávra aggregáljuk, minden C, Cisz, D, … , H hang intenzitását külön-külön összeadjuk, majd eltároljuk egy 12 dimenziós vektorba. Ezen vektor 3 legnagyobb hangját keressük meg, majd megnézzük, hogy ez a három hang alkot-e dúr vagy moll hármashangzatot. Ha igen, akkor az 1. módszer 1 valószínűséget rendel ehhez az akkordhoz.

A találatot kísérleti megfontolásból a következőképpen súlyozzuk. Megkeressük 12 dimenziós vektorba a negyedik legintenzívebb hangot. Ha ez a hang kisebb, mint a harmadik legintenzívebb fele, akkor 1 valószínűséget kap az akkord. Ha viszont nagyobb, akkor a következő képlet szerint:

$$\frac{I_3 - I_4}{I_3/2}, \quad (48)$$

ahol az I_3 a harmadik, az I_4 pedig a negyedik legintenzívebb csúcs a 12 dimenziós vektorban. Ezen súlyozással minél közelebb van a két intenzitás értéke, az 1. módszer annál kisebb valószínűséggel állítja, hogy helyesen ismeri fel az általa mondott akkordot.

Egy konkrét zenerészletnél a súlyozott hangok egy oktávara való összegzésének az eredményét mutatja a 20. ábra. Itt láthatjuk, hogy az C, E és G hangok intenzitása a többihez képest nagy, amiből azt tudjuk megállapítani, hogy annak a valószínűsége, hogy ezt a megfigyelést egy C-dúr rejtett változó generálta, elég magas. A felvett zenerészlet valóban egy C-dúr akkord volt.



20. ábra: 1. módszerhez felhasznált vektor

6.4.4.2 A 2. módszer

A 2. módszer a spektrumon 3 csúcsot keres meg, amikhez tartozó hangokat az akkord alaphangjának valószínűsíti. Ez a 3 csúcs a következő:

- a súlyozott spektrumból az intenzitások egy oktávra való összegzése után a legintenzívebb csúcs
- a súlyozott spektrum 10 legintenzívebb csúcsából a legalsó
- a nem súlyozott spektrumból az intenzitások egy oktávra való összegzése után a legintenzívebb csúcs

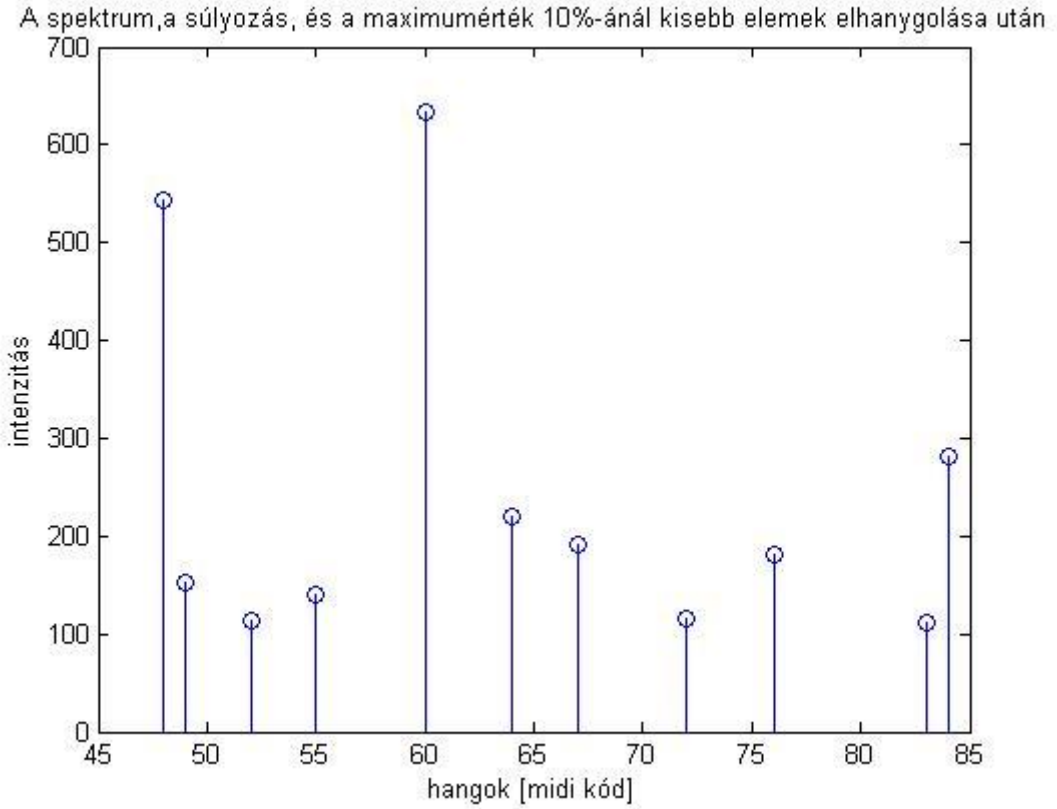
Ez a 3 hang tapasztalati alapon lett kiválasztva, illetve azon elméleti okból, hogy a basszushangok a zenében leggyakrabban az akkord alaphangját hordozzák. A hangok megkeresése után megnézzük, hogy van-e olyan hang, amelyiket legalább 2-szer választottuk ki. Ha nincs ilyen, akkor a 2. módszer nem állít semmit. Ha viszont van, akkor elkészítjük a spektrumból a PCP vektort, és megnézzük, hogy a megtalált hangtól 4 félhangra (nagy terc), illetve 3 félhangra (kis terc) lévő hangok intenzitásai közül melyik a nagyobb. Ha az első, akkor az akkordot dúros jellegűnek, egyébként mollos jellegűnek mondjuk.

A találat súlyozása kísérletezések alapján a következőképpen történik. Megvizsgáljuk, hogy a súlyozott, egy oktávra bontott hangokból melyik a második legintenzívebb. Ezzel és a legintenzívebbel elvégezzük ugyanazt a súlyozást, mint az 1. módszernél. A találat súlyozásának ez az első komponense. Ezenkívül megvizsgáljuk a dúros és a mollos jelleget adó hangok intenzitását is. Ezek különbségét elosztjuk a nagyobb intenzitással. Ez adja a súlyozás második komponensét. A két komponens számtani közepét véve kapjuk meg a találat értékét.

6.4.4.3 A 3. módszer

Csakúgy, mint az 1. módszernél, itt is a súlyozott spektrumból késztjük el a 12 elemű PCP vektort (20. ábra). Ehhez a spektrumnak csak a 0-tól a 7. oktávig lévő részét vesszük figyelembe, ugyanis ezen oktávokon kívüli alaphang általában nem szokott szerepelni a zenékben. Ezután a vektorban külön-külön kiszámítjuk az összes dúros és a mollos hármashangzatra, hogy mekkora az akkordhangok összintenzitása. A kapott 24 elemű összintenzitás vektort normáljuk. Ennek az eredménye lesz a 3. módszer által mondott $P(D_t|A_t)$.

A 21. ábrán egy konkrét példa látható arra, hogy mi marad a spektrumból, ha súlyozzuk, illetve a maximum érték 10%-ánál kisebbeket elhanyagoljuk. Az spektrumképet egy C-dúr akkorddal készítettem, amire részben utal a nagy intenzitású *60-as* midi kódú *C4* hang.



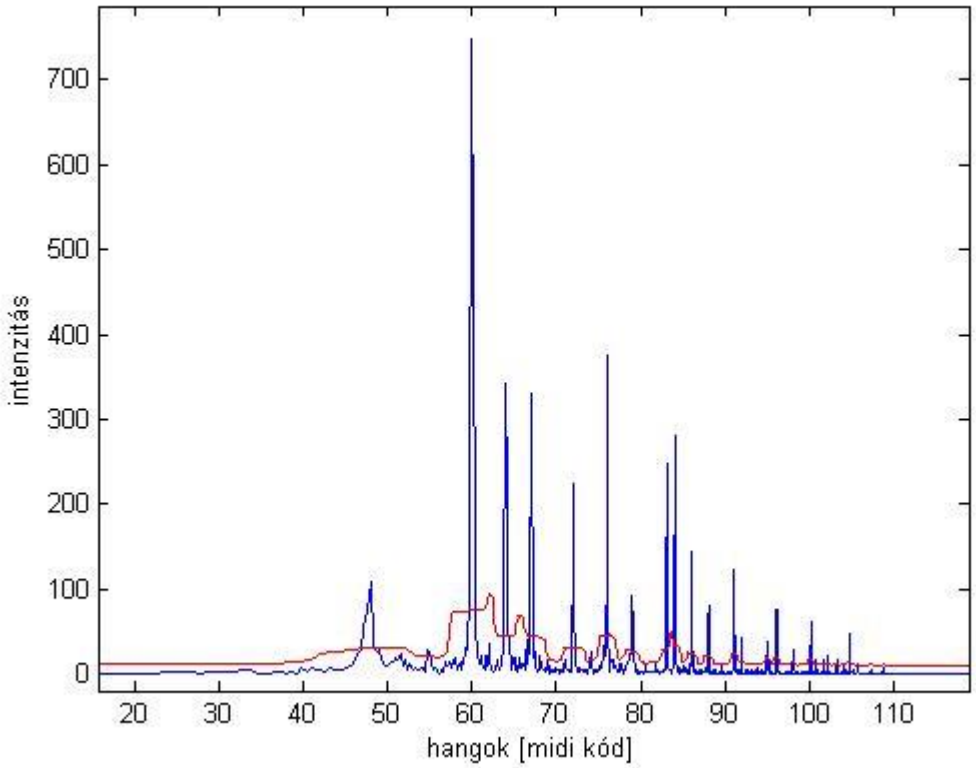
21. ábra: a 3. módszerhez felhasznált hangok és a hozzájuk tartozó intenzitások

6.4.4.4 A 4. módszer

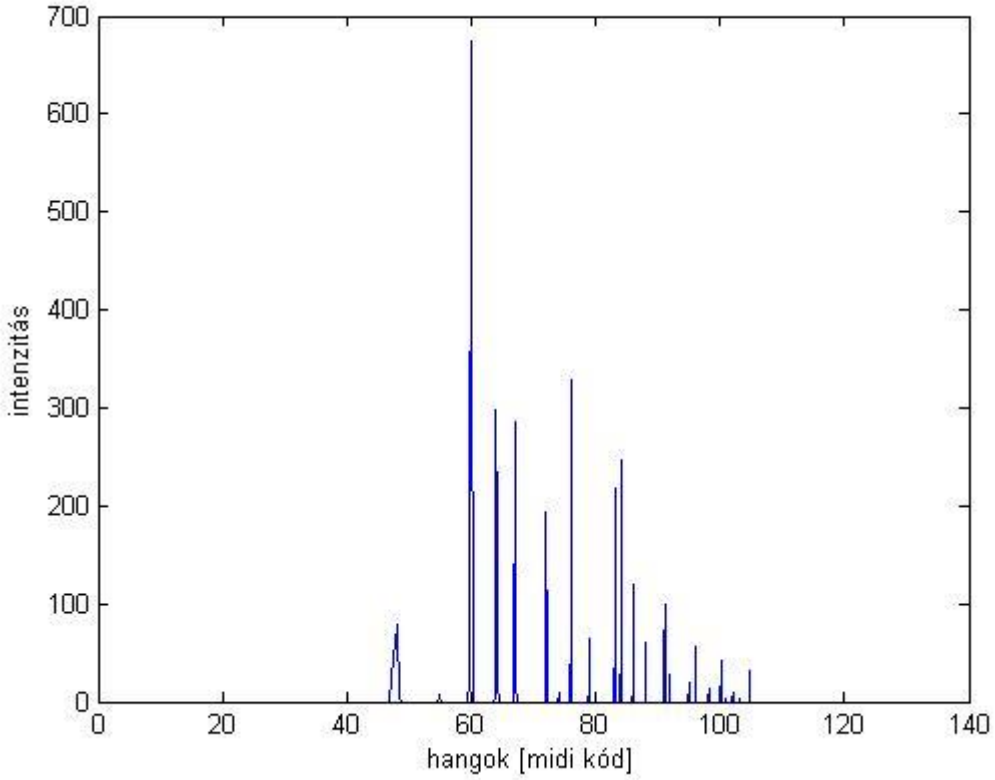
A 4. módszerben a nem súlyozott DFT spektrumból indulunk ki. Itt arra törekszünk, hogy eltávolítsuk a spektrumból a laposabb, zajszerű intenzitásértékeket, amik a számunkra fontos keskeny intenzitáscsúcsok mellett jelennek meg. Ezt a következőképpen érjük el. Ha a spektrumon csúszó ablakos átlagolást végzek, akkor a spektrum „kisimul", azaz a nagy, meredek felfutású csúcsok ellaposodnak. Ezután ha az átlagolt spektrumot az eredetiből kivonjuk, akkor megkapjuk a nagy, keskeny csúcsokat tartalmazó spektrumot. Programozástechnikailag az átlagolást úgy oldottam meg, hogy egy négyszög-ablakot konvolváltam az adott zenerészlet spektrumával. A kivonás után a negatív értékeket nullává írjuk át, ezáltal a számunkra nem fontos részek jelentős hányada eltűnik.

A kísérletezések során 30 egységimpulzussal való konvolúció bizonyult jónak, illetve a simított spektrumot 10 dB-el kellett még felfelé tolni, hogy a kivonás után csak a számunkra fontos csúcsok maradjanak meg. A 22. ábrán kék vonallal van ábrázolva a zenerészlet spektruma, és pirossal a simított spektrum. A kivonás eredménye pedig a 23. ábrán látható. Megfigyelhetjük, hogy a kisebb, szélesebb csúcsok eltűntek.

A 4. módszerben a kivonás után megmaradt csúcsokból az alsó 10-et használjuk fel. Ezek után az algoritmus megegyezik a 3. módszer algoritmusával.



22. ábra: az eredeti és a simított spektrum



23. ábra: a kivonás után megmaradt csúcsok

6.5 Az RMM modell paraméterinek újrabecslése az EM-algoritmus segítségével

A paraméterek újrabecsléséhez alapvetően az *5.4* fejezetben leírtakat használjuk. Először az állapotátmenet-valószínűségi mátrix (***T***), a kezdeti valószínűségi eloszlás (***P₀***), és az érzékelőmodell (***O***) alapján kiszámítjuk ez Előre - Hátra algoritmus által megkapott Előre és Hátra üzeneteket. Ehhez tudjuk, hogy az $f_{1:0}$ megegyezik a kezdeti valószínűségi eloszlással, a $b_{t+1:t}$ pedig egy 24 hosszú vektorral, amiben minden elem 1. Ezekkel az értékekkel inicializálunk, majd az Előre üzeneteteket előre terjesztéssel, a hátra üzeneteket pedig hátraterjesztéssel számítjuk ki. Az indexelés a megfelelő összepárosítását a következő táblázat foglalja össze:

$\mathbf{f_{1:0}}$	$\mathbf{f_{1:1}}$	$\mathbf{f_{1:2}}$	…	$\mathbf{f_{1:t-1}}$	$\mathbf{f_{1:t}}$	$\mathbf{f_{1:t+1}}$
$\mathbf{b_{0:t}}$	$\mathbf{b_{1:t}}$	$\mathbf{b_{2:t}}$	…	$\mathbf{b_{t-1:t}}$	$\mathbf{b_{t:t}}$	$\mathbf{b_{t+1:t}}$

5. táblázat: előre és hátra üzenetek

Ezt követően az Előre, Hátra üzenetekből, a ***T, O*** mátrixokból és a ***P₀*** vektorból kiszámítjuk minden időpillanatra, és minden *i*-ből *j*-be való állapotátmenetnél a $\xi_t(i,j)$ értéket. Ez tehát 24x24x(t+1) nagyságú számhalmaz.

Ezek után kiszámítjuk a modell újrabecsült paramétereit:

$$\overline{\pi_i} = \gamma_1(i) \quad (49)$$

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(\mathrm{i},\mathrm{j})}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (50)$$

$$\overline{b_j}(k) = \frac{\sum^{T}{}_{t=1} \;\; \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \quad (51)$$

Itt az érzékelőmodell újrabecsléséhez használt képlet számlálójában lévő kifejezést kell még értelmeznünk. Mivel a megfigyelésünk folytonos értékű valószínűségi változó, ezért a számlálót a következő módon számítjuk ki:

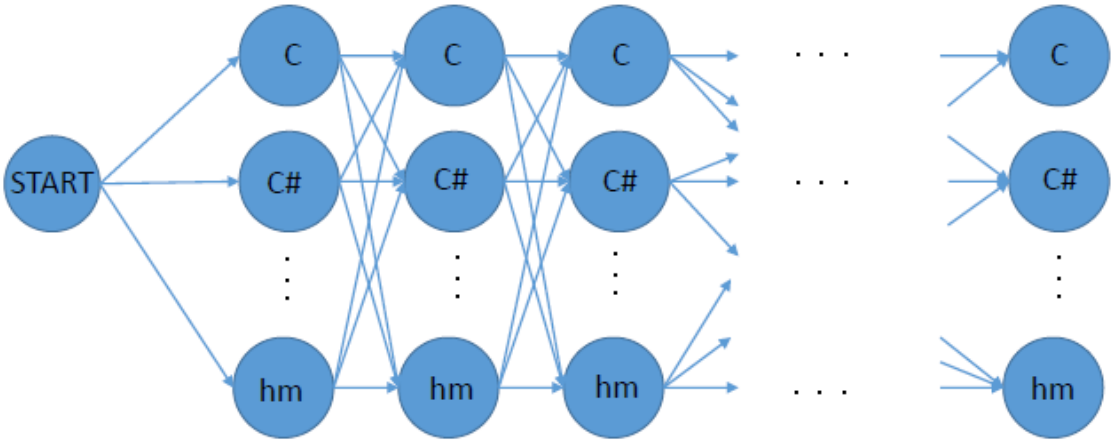
$$\sum_t \{ \gamma_t(i)[b_i(1) \quad \dots \quad b_i(t+1)] \} \quad (52)$$

Az újrabecsléseket iteratív módon hajtjuk végre addig, amíg az ez eggyel korábbi iterációban kapott szám elegendően kicsi értékkel tér csak el.

6.6 Viterbi-algoritmus

Az EM-algoritmussal kiszámított új RMM paraméterekkel most már lefuttathatjuk a Viterbi-algoritmust, hogy megkaphassuk az az akkordok legvalószínűbb sorozatát. Az algoritmushoz készített Trellis-diagram a 24. ábrán látható. Az algoritmust *5.3.4* fejezetben leírtak alapján hajtjuk végre. Az algoritmus során a Trellis-diagram mindegyik csúcsához egy-egy valószínűségi értéket rendelünk. A *START* csúcshoz

definíció szerint 1-et rendelünk. A csúcsokból alkotott első oszlop (első időpillanat, amikor megfigyelést végeztünk) valószínűségei az érzékelőmodell t=1 időre vonatkozó értékének, és a kezdeti valószínűségi eloszlásnak a skaláris szorzata. A következő időpillanatokhoz tartozó csúcsok kiszámítása egy kicsivel bonyolultabb. Vegyünk egy tetszőleges csomópontot. Jelölje ezt $A_{t,x}$. Ehhez a csomóponthoz az előtte lévő oszlopban szereplő összes csúccsal ($A_{t-1,C}, \ldots, A_{t-1,hm}$) ki kell számítani a következő értéket: $A_{t-1,y}$ csomópont valószínűségének, $A_{t-1,y}$-ból $A_{t,x}$-be való átmenet valószínűségének és $P(D_t|A_{t,x})$ valószínűségnek a szorzata. Ezen kiszámított értékekből keressük meg a maximálist. Ezt a valószínűséget fogjuk hozzárendelni $A_{t,x}$ csomóponthoz [21,24].



24. ábra: Viterbi-algoritmus az akkordfelismeréshez

7 A kidolgozott módszerek tesztelése, értékelése

7.1 Mintamegfeleltetéses algoritmus

Az általam használt érzékelő modell eltér a szakirodalomban használt gyakori modellektől (PCP vektor készítése majd mintamegfeleltetés, vagy egyszerű Gauss-modell). A saját algoritmusom érzékelő modellje kétféle dologból tevődik össze. Az első része (1-2. módszer) közvetlen megadja, hogy szerinte melyik akkord a legvalószínűbb, vagy nem mond semmit, mert csak bizonytalant tudna. A második része (3-4. módszer) pedig gyakorlatilag két, saját módon elkészített PCP vektorból adja meg bináris vektorral való mintamegfeleltetéssel, hogy az egyes akkordoknak mekkora a valószínűsége. Ebben a szakaszban a 3. és a 4. módszert fogjuk megvizsgálni, hogy jobban teljesít-e a szakirodalomban elterjedt módszereknél.

Az általam készített kétféle PCP vektor vektoriális összegét veszem, normálom, és ezt hasonlítom a bináris mintákhoz. Mivel ez kétféle PCP vektor összege, ezért ennek az *MPCP* (*Multi PCP*) nevet adtam.

Az PCP vektor fontos szerepet tölt be. Ez a megfigyelésünk, amit az adott zenerészlet leírójaként használunk. A PCP vektor tkp. a mérés eredménye, míg a modell többi paramétere csak statisztikai módon van definiálva. Ez alapján nem mindegy, hogy milyen leírót használunk.

Tesztjeimben a szakirodalomban leginkább elterjedt leírókkal (klasszikus PCP, EPCP) hasonlítottam össze az általam javasolt új leírót. A teszteléshez nagy segítségemre volt az Osmalskyj által készített felcímkézett akkordadatbázis [8], melyet közzé tett az interneten [25]. Ebből a popzenében széles körben elterjedt akusztikus gitárral és zongorával készített akkordokat használtam fel. Ez összesen 2200 akkord, melynek egy része zajos, a másik része pedig csendes, reflexiómentes szobába készült. Az akkordfelismerés eredményét az alábbi táblázat foglalja össze. A táblázatban a helyesen felismert akkordok száma látható százalékban. Látható, hogy a PCP vektor teljesített a legrosszabbul. A második legjobb az EPCP lett, míg a legnagyobb százalékban az MPCP ismerte fel az akkordokat. A teszteknél megnéztem azt is, hogy a kétféle PCP, melynek az összegéből jött ki az MPCP, külön-külön milyen eredményt ér el. A tapasztalat azt mutatja, hogy az MPCP eredménye jobb, mint az őket alkotó kétféle PCP-é. Ezen vizsgálat során felmerült bennem, hogy az MPCP-t kibővítem, és nem kettő PCP-ből készülne, hanem háromból. A harmadik eleme pedig az amúgy is jól teljesítő EPCP lenne. A teszt sikeres volt. Legmagasabb eredményt az EPCP-t is magába foglaló MPCP érte el, amely több, mint 2%-al megelőzte az EPCP-t.

PCP típus \ Hangszer	Gitár			Zongora			Eredmény
PCP	93,62			83			93,14
EPCP	95,86			86			95,41
MPCP (saját)	91,14	90,10	-	83	83	-	96,18
	96,62			87			
MPCP (EPCP-vel)	91,14	90,10	95,86	83	83	86	96,68
	97,10			88			

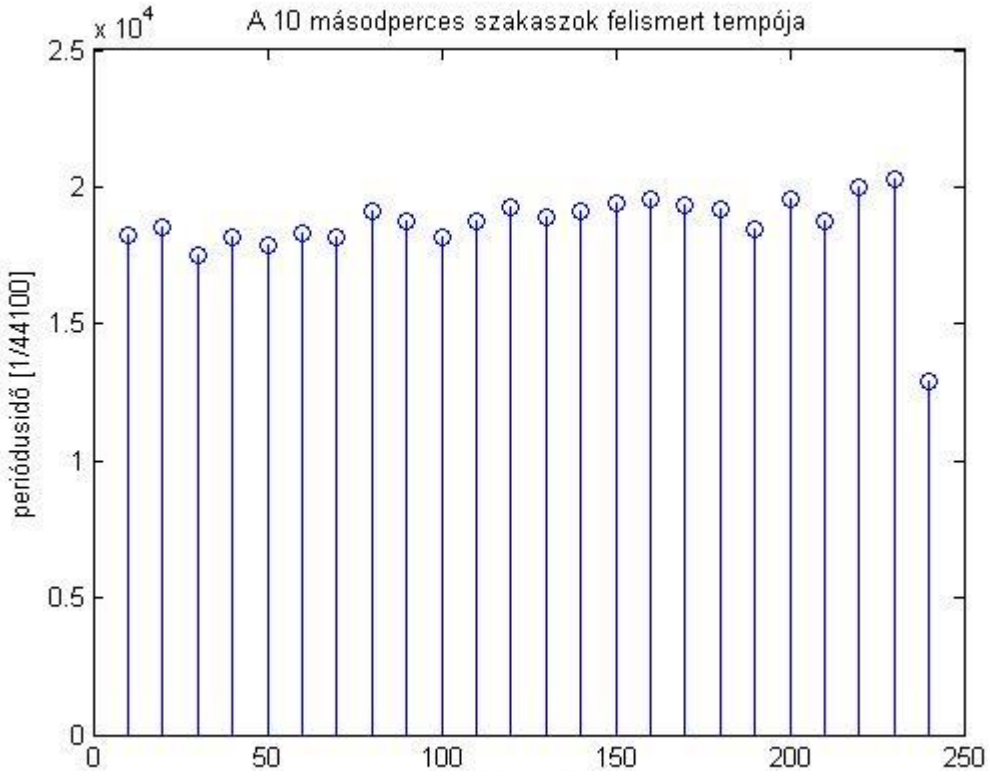
6. táblázat: MPCP összehasonlítása a PCP-vel és az EPCP-vel (az egyes értékek %-ban)

Ez azt mutatja, hogy a leggyakrabban használt PCP és EPCP vektoroknál lényegesen jobb eredményt lehet elérni a kifinomultabb jelfeldolgozást alkalmazó MPCP vektorral. Ezáltal a PCP-hez képest a hibás akkordfelismerések száma felére, az EPCP-hez képest pedig annak 70%-ára csökkent.

7.2 Akkordfelismerő program

7.2.1 Első teszt

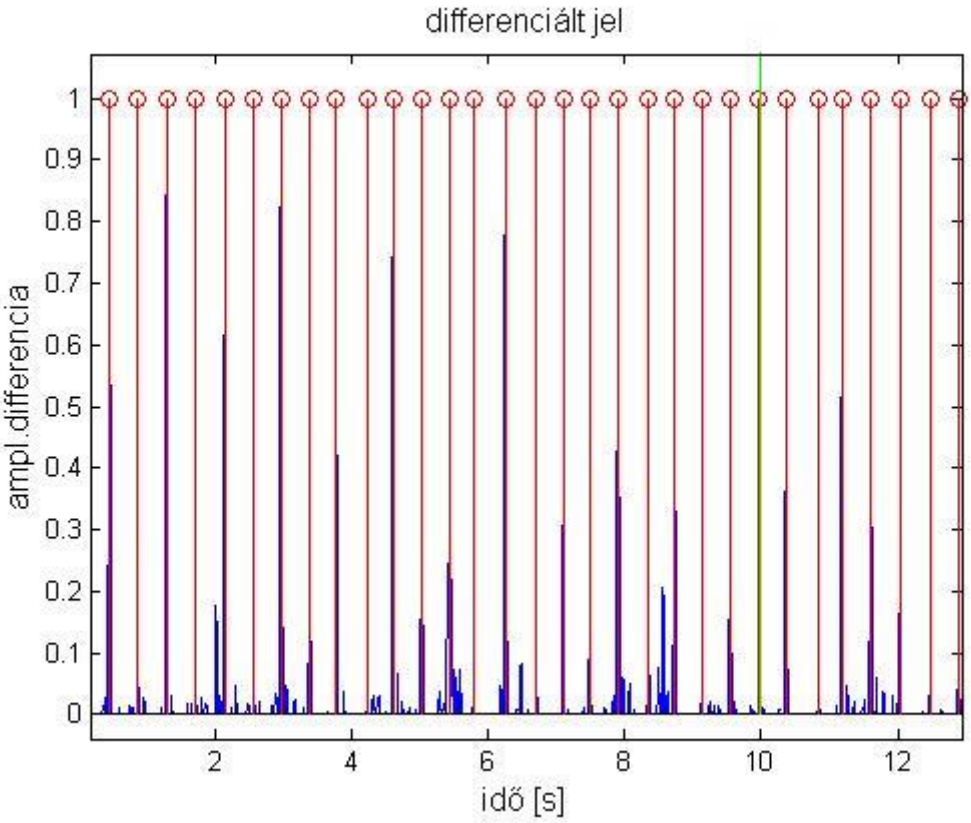
Az elkészített akkordfelismerő programot először a *Beatles* együttes *Let it be* c. dalán teszteltem, melyet a program összesen közel 800 zenerészletre osztotta fel, tehát ennyi akkordfelismerés történik. A tesztelés során csak a 3. és 4. módszert használtam érzékelő modellként. A program elsőként a zene tempóját határozza meg az egyes 10 másodperces szakaszokban. A vizsgált zeneszámnál megfigyelhetjük, hogy az ezen szakaszok felismert tempójában van némi ingadozás. Ezt a 25. ábrán tudunk megfigyelni. Az ingadozást azért tapasztalhatjuk, mert a Beatles együttes nem metronómmal vette fel



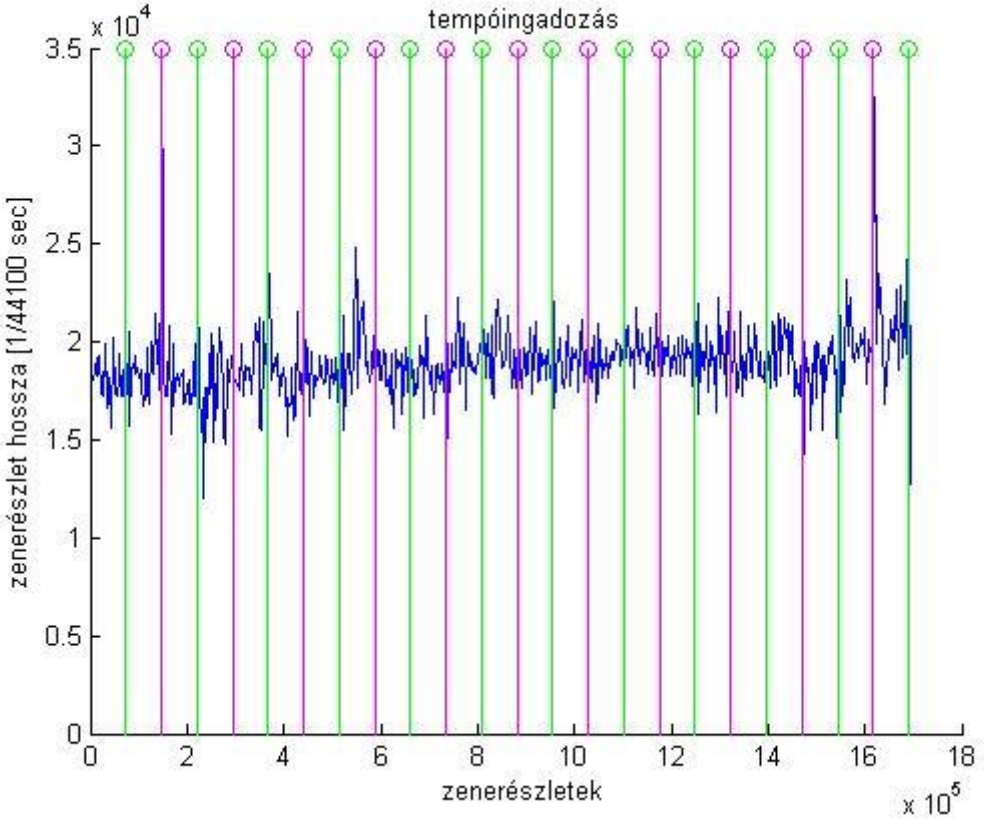
25. ábra: A 10 másodperces szakaszok felismert tempójának az ingadozása

ezt a dalt. Az akkordfelismerés szempontjából ez nehezítő tényező. Manapság ugyan a legtöbb könnyűzenei felvétel metronóm használatával készül, nem hátrány, ha ez nem feltétele a helyes akkordfelismerésnek.

A 26. ábrán a zene egységnyi részekre darabolásának az eredménye látható. Megfigyelhetjük, hogy az algoritmus a 10 másodperces zenerészletből felismert tempó alapján viszonylag egyenletesen, a zene dinamikájától függően osztja fel a zenét. Továbbá megfigyelhető, hogy a program megtalálja a differenciált jel nagyobb csúcsait. A 27. ábrán pedig azt mutatom be, hogy az egységnyi zenerészlet hossza (egy megfigyelés) hogyan változik.



26. ábra: A zenéből készített differenciált jel, és az egyes zenerészleteket határoló vonalak



27. ábra: zenerészletek periódusidejének a változása

Az egyes zenerészletekhez a program ezután $P(D_t|A_t)$ vektort rendel, felveszi a rejtett Markov-modell paramétereit, és elkezdi iteratívan keresni az egyes időpillanatokhoz tartozó jobb $P(D_t|A_t)$ értékeket. Ennél a zenénél az elvárásmaximalizációs algoritmus 19 iteráció után megáll. A kezdeti megfigyelési modell meghatározásakor még sok olyan $P(D_t|A_t)$ vektor van, amelynek a maximuma nem a megfelelő akkord, viszont a Viterbi-algoritmus a zenét globálisan nézi, a maximális valószínűségű utat keresi a Trellis-diagramban, amelynek az állapotátmenetvalószínűségi mátrixát úgy határoztuk meg, hogy a sajátmagába visszamenő állapotátmenet gyakori, így az egyes „kilógó" értékeket „visszahúzza" a környezetében lévők közé.

A programom eredményességét három másik algoritmussal hasonlítottam össze. Ebből az első kettő a szakirodalmi kutatásom során megismert EPCP-s módszer [2] és a Bello és Pickens-féle módszer [3]. A harmadik a piacon megtalálható *Chordify,* ami a legismertebb webes akkordfelismerő alkalmazás [26]. Az első kettőt sikerült implementálnom, viszont a harmadiknak az algoritmusa nincs közzé téve, így ezt csak felhasználói szinten tudtam megvizsgálni. A saját algoritmusom, és a Bello és Pickens-félénél is a tempó alapján szeparáltam a zenét kis részekre, azonos módon. Az EPCP-s algoritmusnak nem része a tempó alapú szeparáció, de azt is annak a felhasználásával vizsgáltam. Az EPCP-s és a többi algoritmus között a rejtett Markov-modell és az elvárásmaximalizásció használata a különbség, míg a saját programom, és a Bello és Pickens-féle között az érzékelő modell, és a RMM kezdeti paramétereinek a beállítása. Az összehasonlítás alapjául azt választottam, hogy az akkordváltásokat milyen jól ismeri fel. Tehát az jelent hibát, ha rossz akkord következik a sorban, vagy ha „beleragad" az előzőbe, és nem érzékeli az újat.

Elsőként az algoritmusomat a Chordify programmal [26] hasonlítottam össze. A valós és a felismert akkordokat a következő táblázatban láthatjuk, soronként párosítva.

Valódi akkordok	Felismert akkordok	
	Saját alkalmazás	Chordify
C G Am F C G F C	C G Am F C G F C	C G Am F C G F C
C G Am F C G F C	C G Am F C G F C	C G Am F C G F C
C G Am F C G F C	C G Am Am C G F C	C G Am F C G F C
Am G F C G F C	Am G C C G C C	Am G F C G F C
C G Am F C G F C	C G Am F C G F C	C G Am F C G F C
C G Am F C G F C	C G G F C G F C	C G Am F C G F C
Am G F C G F C	Am G F C G F Am	Am G F C G F C
Am G F C G F C	Am G F C G F C	Am G F C G F C
F C G F C	F C G F C	F Em C B G F C
F C G F C	F C G G C	F C G F C
C G Am F C G F C	C G Am F C G F C	C G Am F C G F Dm F
C G Am F C G F C	C G Am F C G F F	C G Am F C G F C
Am G F C G F C	Am G F C G F C	Am G F C G F C
C G Am F C G F C	C G G F C G d-moll C	C G Am F C G F C
C G Am F C G F C	C G G F C G F F	C G Am F C G F C
Am G F C G F C	Am G F C G F C	Am G F C G F C
Am G F C G F C	Am G F C G F C	Am C F C G F C
F C G F C	F C C F C	F Em C B G F C

7. táblázat: akkordfelismerés eredménye (Let it be)

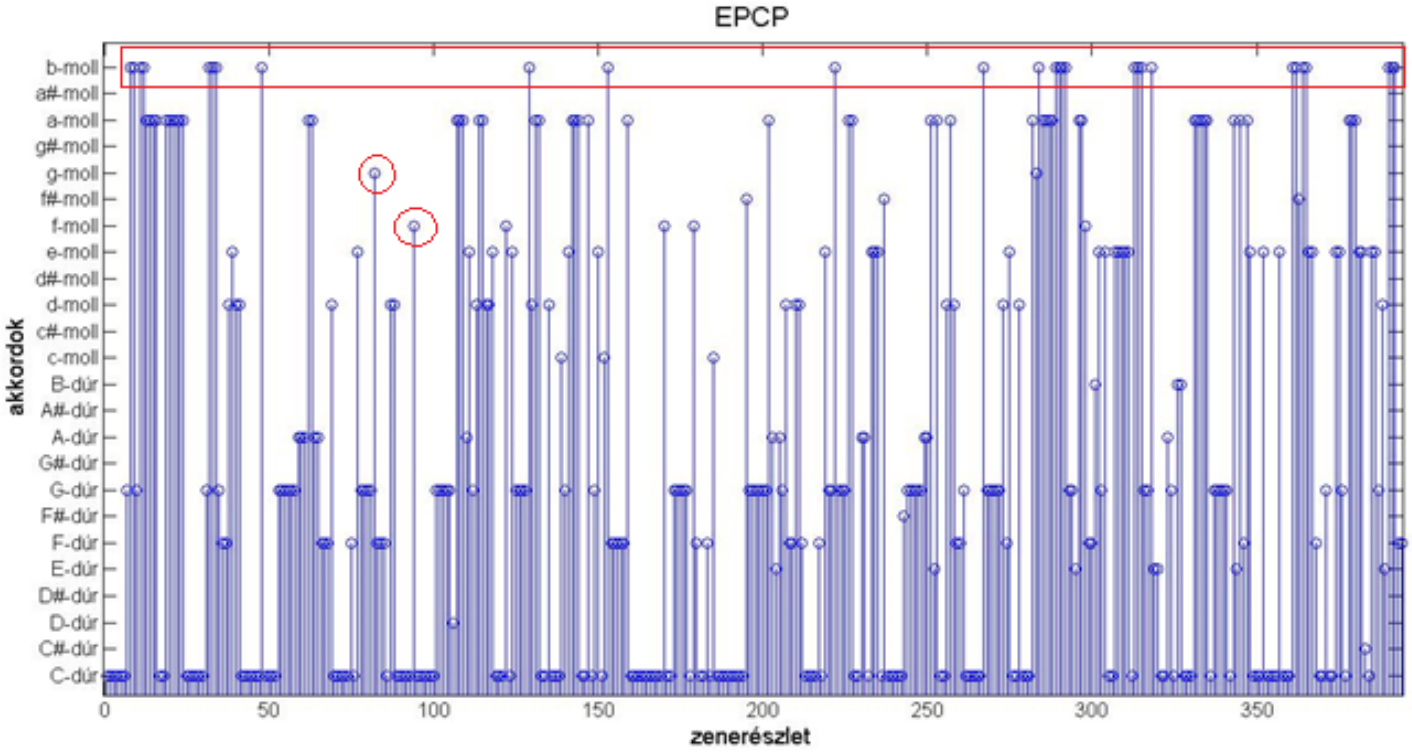
A programom által felismert akkordok nagyrészt megfelelőek, összesen 12 helyen ront. A hiba a legtöbb esetben úgy keletkezik, hogy a program nem detektálja az akkordváltást (ugyanabban az állapotban marad). Ez valószínűleg annak köszönhető, hogy az állapotátmenetvalószínűségi mátrix átlójában jóval nagyobb valószínűségi értékek vannak, mint máshol.

A Chordify által felismert akkordok is elég jól megfelelnek a valóságnak. A program mindössze 6 helyen hoz hibás döntést.

A hibásan felismert akkordokat pirossal jelöltem. Megjegyzendő viszont, hogy ezen akkordok a zenei érzetet figyelembe véve nem teljesen rosszak. A programomban az összes hibás akkord a Let it be hangnemében, C-dúrban marad. A Chordify-ra ez nem teljesen igaz, mivel szerepel benne kétszer is a B-dúr hármashangzat. Viszont a megfelelő zenei részbe hangszerrel bejátszva kipróbáltam a B-dúr akkordot, és tulajdonképpen nem szólt rosszul.

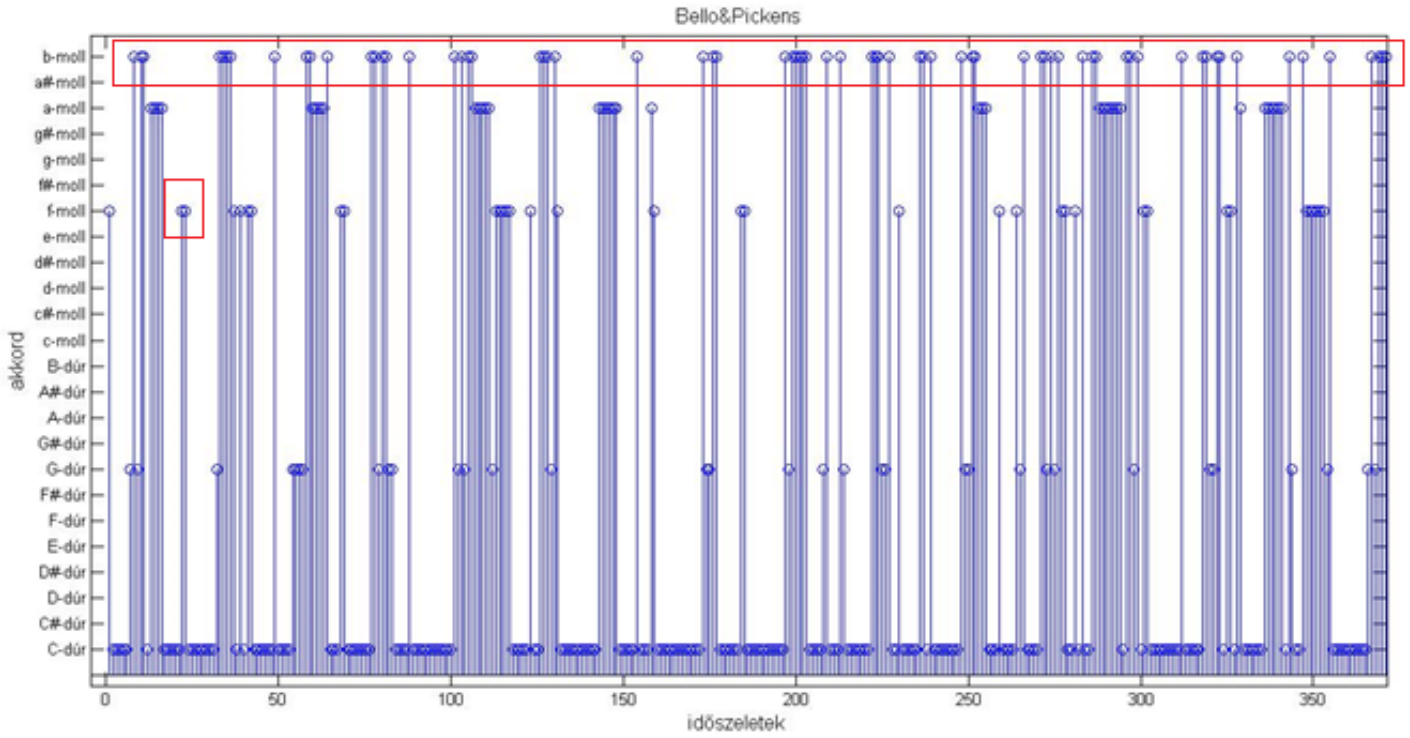
Ezek után a két, szakirodalomból megismert algoritmust próbáltam ki. Az EPCP-s algoritmus [2] eredménye a 28. ábrán látható. Megfigyelhetjük, hogy vannak helyek, ahol az algoritmus pontos eredményt ad, az is látszik, hogy a Let it be skálájában szereplő akkordok vannak dominánsan, viszont több olyan eset is látható, ahol nem skálába lévő akkordok szólalnak meg, ilyen eseteket láthatunk pirossal körberajzolva (b-moll, f-moll). Az összes hibás felismerést nem jelöltem be. Természetesen egy dal tartalmazhat nem skálába illő akkordot, de ebben a zenében nincs ilyen. Az EPCP-s algoritmus [2] nem használja fel a Viterbi-algoritmust, ami pedig nagyban javíthatná azt. A másik, szakirodalomból megismert algoritmus Bello és Pickens algoritmusa [3], amely már zenei információk alapján inicializált RMM-et használ, egyszerű Gauss modellként definiálja

az érzékelőmodellt, illetve PCP vektort használ a megfigyelés leírására. Az algoritmus eredménye a 29. ábrán látható. Megfigyelhetjük, hogy a Viterbi-algoritmus a tranziens-szerű kiugrásokat csökkenti. Több helyen egészen jól ismeri fel a dal akkordmenetét, de sok másik helyen téveszt. Néhány hibát itt is körberajzoltam piros színnel, de ez nem az összes (a b-mollok, f-mollok mind hibásak).

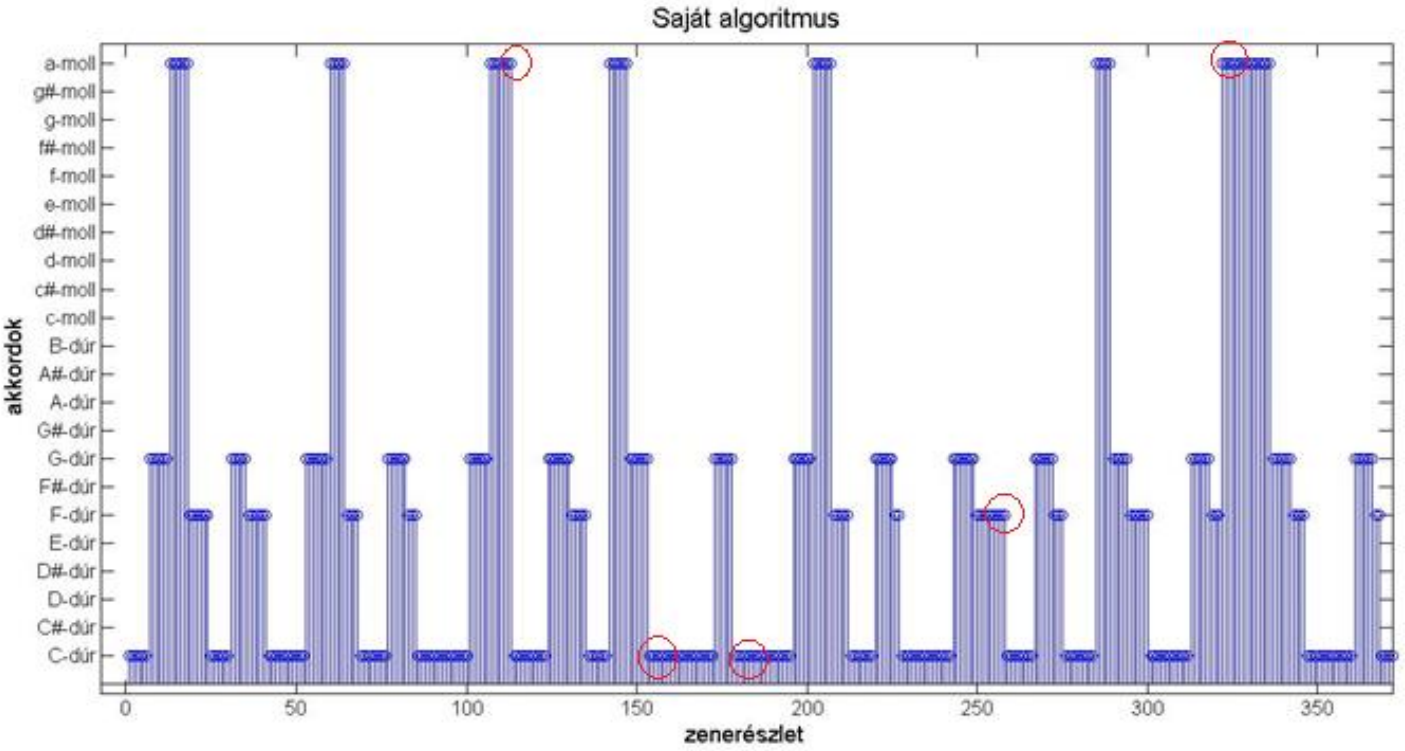


28. ábra: EPCP-s algoritmus eredménye (tesztzene: Beatles)

Ezzel szemben az általam írt algoritmus (30. ábra) itt kevesebb helyen téveszt. Jellemző hibája, hogy az állapotátmenetet nem ismeri fel, hanem beragad az előző állapotba vagy a következő állapot hamarabb megjelenik. A hibáit piros színnel rajzoltam körbe. Ezenkívül elmondható az is, hogy az összes felismert akkord a dal skálájában marad.



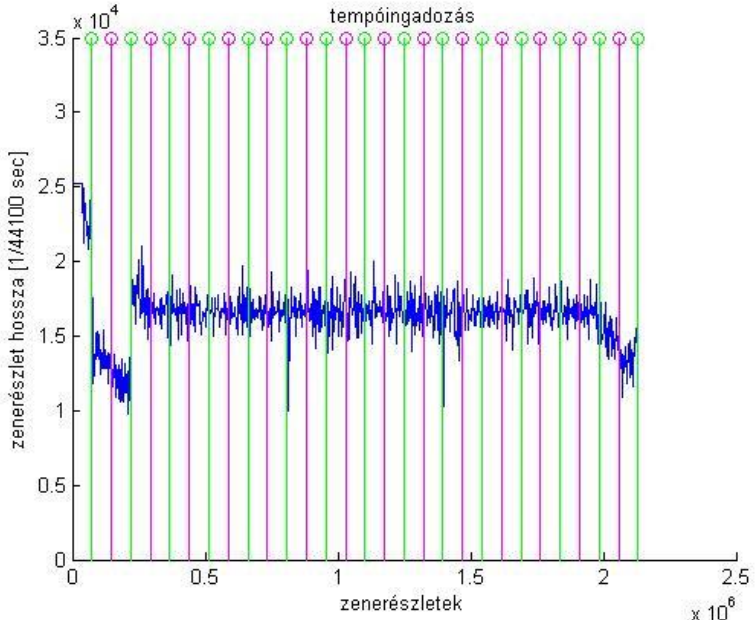
29. ábra: Bello és Pickens algoritmusának az eredménye (tesztzene: Beatles)



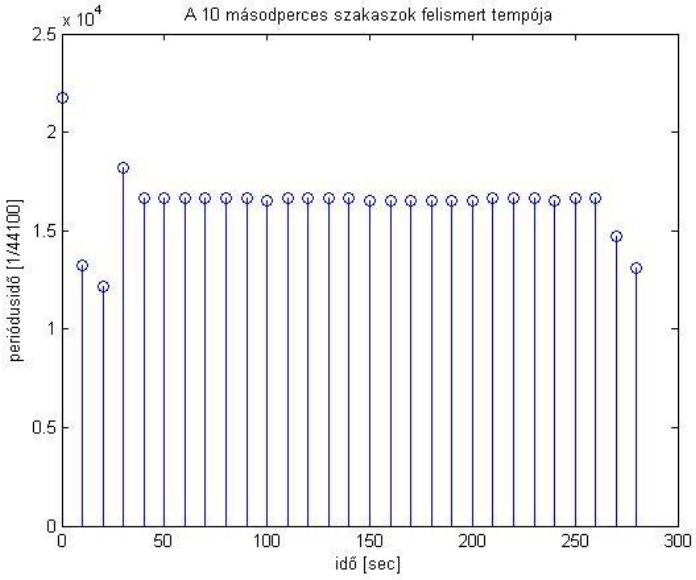
30. ábra: A saját algoritmusom eredménye (Beatles)

Második teszt

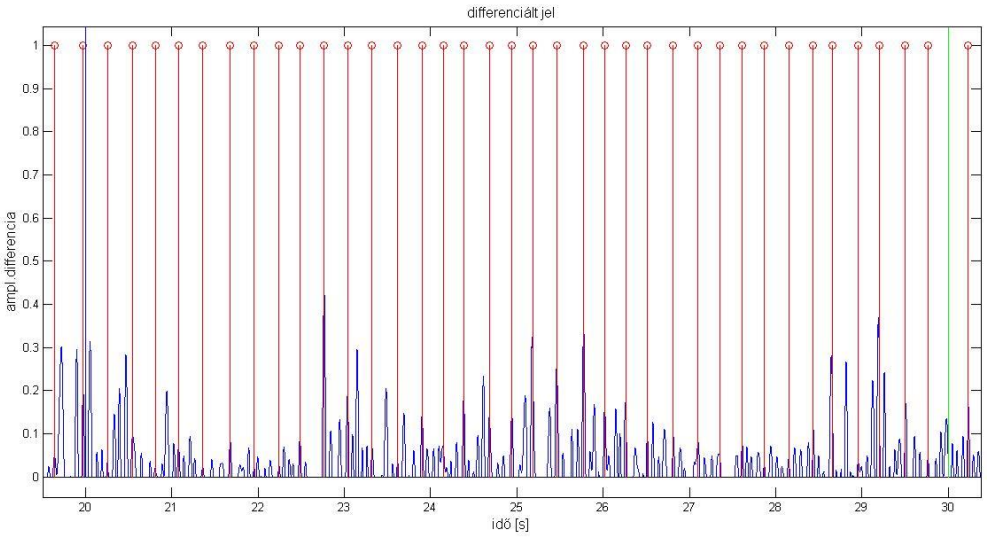
A második tesztet a Dreamer c. számon hajtottam végre. Látható, hogy a felismert tempó nem nagyon változik, és a bevezető zongorajáték után az egyes zenerészek hosszát is hasonlónak számítja az algoritmus. Viszont ha megfigyeljük, a differenciált jelnél előfordul olyan, hogy nagy csúcson nem megy keresztül határvonal. Ez azt jelenti, hogy zenerészleten beül történt valamilyen váltás, bár ez nem feltétlen a kísérő hangszerekkel történik.



31. ábra: A 10 másodperces szakaszok tempói (Dreamer)



32. ábra: zenerészletek periódusidejének a változása (Dreamer)



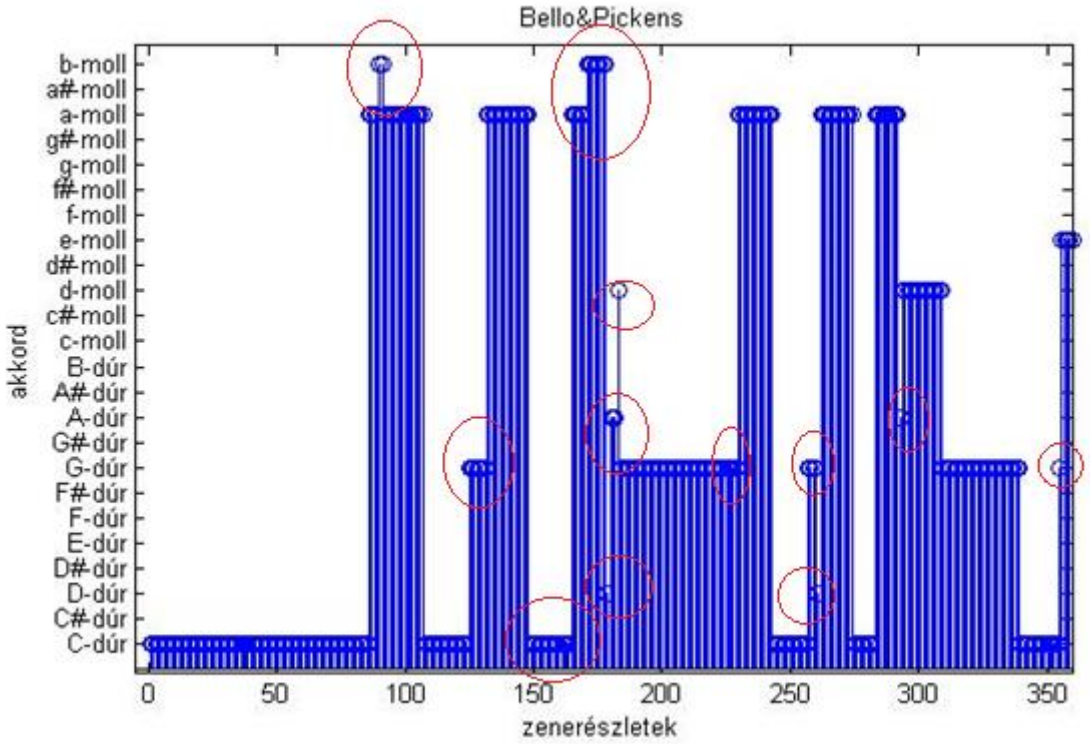
33. ábra: A zenéből készített differenciált jel, és az egyes zenerészleteket határoló vonalak

Valódi akkordok		Felismert akkordok	
		Saját alkalmazás	Chordify
Bevezető	C	C G	C
Vers 1	C Am	C Am	C Am
	C Am	C Am	C Em Am
	F Dm G	F G G	F Dm G
	C Am	C Am	C Am
	C Am	Am Am	C Am
	F Dm G	Am Dm G	F Dm G
Refrén 1	C	Am	C
	Am Em G	Am Em Em	Am Em G
	C	Em	C
	Am Em G	Am Em G	Am Em G
Vers 2	C Am	C Am	C Am
	C Am	C Dm	C Am
	F Dm G	Dm Dm Dm	F Dm G
Refrén 2	C	C	C
	Am Em G	Am Em G	Am Em G
	C	C	C
	Am Em G	Am Em G	Am Em G
Átvezető	Dm G	D G	Dm G
	Dm G	G G	Dm G
	Dm G	G G	Dm G
	Dm G	Dm C	Dm G
Szóló	C Am C Am	C Am Am Am	C Am C Am
	C Am F Dm G	Am Am Am Em C	C Am Em G
Vers 3	C Am	C Fm	C Am
	C Am	C Am G	C Am
	F Dm G	F D G	F Dm G
Refrén 3	C	C	C
	Am Em G	Am Em Em	Am Em G

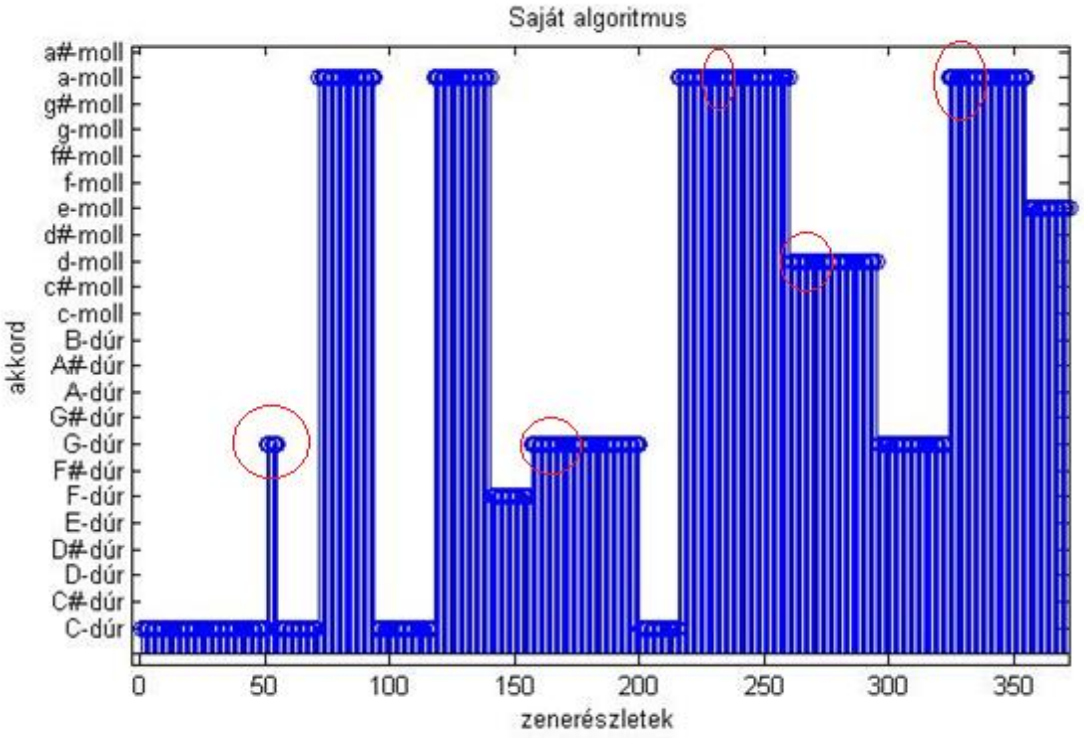
	C	Am	C
	Am Em G	Am Em G	Am Em G
	C	Am	C
	Am Em G C	Am Em G C	Am Em G
	C	C F	C
	Am Em G C	C C A G	Am Em G

8. táblázat: akkordfelismerés eredménye (Dreamer)

Ennek a dalnak az akkordjait az algoritmusom rosszabb arányban ismeri fel, itt 25-ször vét hibát, míg a Chordify csak 2-szer. Az EPCP-s algoritmus erre a dalra sokkal kevésbé értelmezhető eredményt adott, a sok kiugró érték között alig vehető ki a dal akkordmenete. Bello és Pickens algoritmusa a zene bevezetőjére, és első versszakra a 34. ábrán látható. Megfigyelhetjük, hogy a Viterbi-algoritmus használata mellett is megjelennek kiugró értékek, bár kisebb mértékben. A 35. ábrán látható az általam készített algoritmus, melyben hasonló módon a hiba leginkább abból adódik, hogy a Viterbi-algoritmus miatt néhány helyen a találat a környezetében lévő értéket veszi fel.



34. ábra: Bello&Pickens algoritmusának az eredménye



35. ábra: a saját algoritmusom eredménye

Megjegyzendő, hogy Bello és Pickens algoritmusa ebben a dalban egyes helyeken az általam írt algoritmushoz képest jobban teljesít, ha azt vesszük alapul, hogy hány darab zenerészlethez tartozó akkordot ismer fel helyesen. Viszont ezen részekben is előfordulnak a kiugró, hibás értékek.

A részletes összehasonlításhoz további tesztek szükségesek. Az akkordfelismerő szoftver tesztelése nehéz feladat, ugyanis ehhez adott időpontokhoz tartozó akkordokkal felcímkézett zenékre lenne szükség. Egy ilyen adatbázis készítése nagyon hosszadalmas feladat, és az interneten csak Osmalskyj egyhangszeres akkordjait [25] találtam meg.

8 Zenei improvizáció készítése

Gépi improvizáció alatt azt a folyamatot értjük, hogy a számítógép a számára átadott zenei információt feldolgozza, majd ehhez rögtönzött dallamot generál. A zenei jel feldolgozása és az improvizációkészítés történhet valós időben, illetve offline. Az input zenei jel lehet egy felvett, vagy éppen játszott zene, amely egy mikrofonon keresztül kapcsolódik a PC-hez, de lehet akár MIDI kódok sorozata, MusicXML [29], vagy egyéb zenei információ kódolására alkalmas formátumú fájl. Gépi improvizációt megvalósító algoritmus készítése többféle megközelítésben lehetséges, ehhez számos heurisztikát lehet kitalálni. Egy ilyen szoftver készítése legalább alapszintű zeneelméleti háttértudást igényel, de minél jobban ismerjük a zeneelméletet, annál több heurisztikát, szabályrendszert építhetünk bele az algoritmusba, amely a készített zene minőségét javíthatja. A következő fejezetben a szakirodalomban fellelhető megoldások közül a *Robert Keller* és *David Morrison* által elkészített algoritmust ismertetem, melyből készült implementáció a piacon is fellelhető *Impro-Visor* néven [27]. Majd pedig az az utáni fejezetben az általam készített algoritmust mutatom be.

8.1 Impro-Visor

8.1.1 Bevezető

Az Impro-Visor (*Improvisation Advisor*) egy ingyenesen letölthető program [27], amely előírt akkordsorozathoz készít egyszólamú jazz improvizációt. A szoftver valós időben nem működik, a program készítői az ezzel kapcsolatos eredményeket még nem érezték megfelelőnek. A szoftvernek inputként meg kell adni az akkordsorozatot, melyhez improvizációt szeretnénk készíteni, illetve konfigurálhatjuk, hogy az elkészített improvizációban milyen valószínűséggel jelenjenek meg az egyes zenei elemek, úgymint a ritmusértékek, a hang-típusok, a szünet. Ezenkívül a felhasználó korlátozásokat is megadhat, mint például a dallam maximális és minimális hangmagassága.

Az algoritmus készítőinek kétféle elképzelése volt a feladat megoldását illetően. Először olyan programban gondolkodtak, hogy az egyes akkordmenetekhez többféle, előre elkészített rövid dalrészleteket írnak, amiből a program választ egyet. Ezt a megoldást elvetették, ugyanis nagyon sokféle akkordmenet létezik, így nagyon hosszú időbe tartana egy ilyen adatbázis elkészítése. Az előnye viszont az lenne, hogy garantáltan megfelelő minőségű dallamot tudnánk generálni. A második ötlet az volt, hogy a rövid dalrészleteket a program dinamikusan generálja, valószínűségi nyelvtant felhasználva. Ezen megoldáshoz ugyan nem kell nagyméretű adatbázis, viszont megfelelő minőségű zenét nehezebb generálni. Keller és Morrison emellett a módszer mellett döntött. Az algoritmus megértéséhez elsőként némi áttekintés szükséges a formális nyelvek matematikájáról [28].

8.1.2 Formális nyelvek

A nyelv ebben az esetben karakterekből áll. A nyelvben felhasználható karakterek összessége a karakterkészlet (jele: Σ). A karakterekből alkotott sorozatot szövegnek nevezzük. Legyen Σ^* a karakterkészletből előállítható összes lehetséges szöveg halmaza (tetszőleges hosszúságú). A nyelv Σ^* egy lehetséges részhalmaza, amit definiálhatjuk felsorolással, nyelvtannal, illetve automatával. Mi itt most csak a nyelvtannal való definiálást vizsgáljuk meg.

A nyelvtan olyan szabályrendszer, mely segítségével a nyelv egyes mondatai levezethetőek. Kompakt módon leírva a nyelvet a $G(N,\Sigma,P,S)$ négyes határozza meg, ahol:

- N: nemterminálisok (nyelvtani kategóriák)
- Σ: terminálisok (karakterkészlet)
- P: helyettesítési szabályok
- S: kiinduló szimbólum, mondatszimbólum ($S \in N$)

A nyelv egyes elemeit a mondatszimbólumból kiindulva a helyettesítési szabályok felhasználásával vezethetjük le. Ennek során az egyik mondatszerű formából egy másikba megyünk át. A mondatszerű forma terminálisokból és nemterminálisokból áll. Sikeres levezetés esetén végül csak terminálisokból álló karaktersorozatot kapunk, mely a G grammatika egy értelmes mondta. Megjegyzendő, hogy terminálisokat általában kisbetűvel, míg a nemterminálisokat nagybetűvel jelöljük.

Chomsky a helyettesítési szabályok bonyolultsága alapján négy nyelvosztályt határozott meg, melyek növekvő komplexitási sorrendben a következőek: reguláris nyelvek, környezetfüggetlen nyelvek, környezetfüggő nyelvek, rekurzívan felsorolható nyelvek. Keller és Morrison által készített zenei improvizációs algoritmus környezetfüggetlen (CFL = context-free grammar) nyelvtant használ. A CFL nyelvosztályba tartozó nyelvekre általánosan az $A \rightarrow \alpha$ típusú levezetési szabály érvényes, ahol a bal oldalon egy nemterminális áll, míg a jobboldalon egy tetszőleges terminálisokból és nemterminálisokból álló sorozat [30][28].

8.1.3 Formális nyelvek használata zenei improvizáció készítéséhez

Az Impro-Visor program először a zene ritmusának az improvizációját készíti el, majd csak később rendel az egyes ritmusértékekhez hangokat. Ahhoz, hogy megértsük a formális nyelvek szerepét, vizsgáljuk meg a következő példát.

Ritmusszekvenciákat szeretnénk generálni, ami tetszőleges hosszúságú, illetve csak negyed, és félhangokból áll. Ezenkívül nem fogadunk el bármilyen ritmust, ehhez szabályokat használunk fel. Matematikailag ez a következőképpen írható fel:

- A karakterkészlet: $\Sigma = \{\mathrm{h}, q\}$ (fél és negyedhangok)
- A nemterminálisok: $N = \{S, M, H\}$ (start szimbólum, egész ütem, fél ütem)

- A helyettesítési szabályok:
  - $S \rightarrow M$
  - $S \rightarrow MS$
  - $M \rightarrow HH$
  - $H \rightarrow h$
  - $H \rightarrow qq$

Most nézzünk néhány példát, ahol konkrét ritmusszekvenciákat vezetünk le.

- S → M → HH → hH → hh
- S → M → HH → hH → hqq
- S → M → HH → qqH → qqh
- S → MS → HHS → qqHS → qqhS → qqhHH → qqhqqH → qqhqqqq

Az Impro-Visor ennél természetesen gazdagabb ritmuskészletet biztosít, beleértve a szünetet is, illetve a lehetséges ritmusképletek gyakorisága is konfigurálható.

Most nézzük meg, hogy egy ritmusszekvenciához az Impro-Visor hogyan rendel konkrét hangokat. A program a hangokat adott akkordokhoz viszonyítva négy csoportra osztja. Ezek a következőek:

- *akkordhangok*: az adott akkord hangjai
- *színezőhangok*: konszonáns az akkordhangokkal
- *átvezetőhangok*: nem akkordhang, ami jó átvezetést biztosít akkordhangokhoz vagy színezőhangokhoz
- *egyéb hangok*: olyan hangok, amelyek nem tartoznak semelyik előző kategóriába

A program a hangszekvencia generálásához az arra vonatkozó nyelvtant használja fel. Ezenkívül az improvizációhoz előzetes megkötéseket is használhatunk. Megmondhatjuk a minimális és maximális hangmagasságot, a legnagyobb, és legkisebb hangközt az improvizációban két szomszédos hangnál, illetve a felhasznált hangközök gyakoriságát. Egyéb megkötések jövőbeli munkaként szerepelnek.

A program eredményének az értékelése szubjektív. Amivel le tudjuk mérni az eredményességét, az esetleg lehetne az, hogy az emberek hány százalékának tetszik a generált dallam [28].

8.2 Saját improvizációs algoritmus offline tesztelése

Ezek után a szakirodalmi kutatásból ötleteket merítve, elkészítettem a saját improvizációs algoritmusomat. A célom egy valós idejű algoritmus elkészítése, így a tervezésekor alapvetően a valós idejű követelményeknek való megfelelést kellett szem előtt tartanom. Először az improvizáció készítést offline módon, Matlab környezetben teszteltem.

Az improvizáció készítéséhez a zenéről a következő információkat kell ismernünk:

- a zene tempója
- a zene hangneme
- a zene akkordmenete

Akkordfelismeréssel és tempódetektálással már foglalkoztunk, viszont szükségünk lesz a zene hangnemének a felismerésére is. A hangnemfelismerésre az általam használt algoritmus a felismert akkordokat használja fel, és viszonylag egyszerű. A hangnemfelismerés csak a dúr és a természetes moll skálát képes felismerni. Minden dúr skálának van egy párhuzamos moll skálája, amivel a hangkészletük megegyezik. Ebben az alkalmazásban az egyszerűség kedvéért ezeket nem különböztetjük meg. Ez a két skála hét hangot tartalmaz. Az egyszerűbb zenék általában ezt a két skálát használják.

8.2.1 A hangnemfelismerés

1. A felismert akkordokat típusai szerint összeszámoljuk.
2. Készítünk egy 12 elemű vektort, amibe az egyes elemek sorba (C-dúr, Cisz-dúr,…, H-dúr) azt reprezentálják, hogy mennyire valószínű az adott skála
3. Zeneelméletből ismert, hogy a dúr skála (vagy a hozzá tartozó párhuzamos moll) egyes fokaira (a 7-et kivéve) olyan megfelelő dúr vagy moll hármashangzatokat építhetünk, amiknek a hangjai ugyanazon a skálán belül maradnak. Nézzünk meg ezt a C-dúr skála példáján:
    a. I. fokra: C-dúr
    b. II. fokra: d-moll
    c. III. fokra: e-moll
    d. IV. fokra: F-dúr
    e. V. fokra: G-dúr
    f. VI. fokra: a-moll

    Tehát minden skálához létezik hat ilyen akkord, amit megkaphatunk, ha a példában szereplő akkordokat annak megfelelően transzponáljuk. A 12 elemű vektor adott elemébe az fog bekerülni, hogy az annak megfelelő skálához tartozó 6 akkord milyen gyakorisággal fordult elő.

    Ismerve az akkordokat, a tempót és a hangnemet, zenei improvizációként egy egyszerű, egyszólamú dallamot fogunk generálni.

8.2.2 A generált dallam hangkészlete

A hangkészletünk kétféle elemekből fog állni:

1. Pentaton skála hangjai: a felismert hangnemből származtatható öt fokú skála, amit úgy kapunk, ha a skálában lévő dó-ré-mi-szó-lá hangokat használjuk csak fel. Ezen hangkészletet a zene egész időtartama alatt használhatjuk
2. Akkordhangok: ez háromféle hang, csak a megfelelő időpillanatban használhatjuk fel.

A szólók természetéből fakadóan a hangok kiválasztásában egyéb megkötések is szerepelnek:

- A pentaton és akkordhangokat csak a midi kód szerinti 3, 4, 5-ös oktávokból szedjük, így a pentaton esetben 15, az akkordhang esetében 9 darab hangunk lesz.
- Két egymás utáni hang nem lehet oktáv távolságnál messzebb. Ha megengednénk, akkor a nagy ugrások zavaróak lennének.

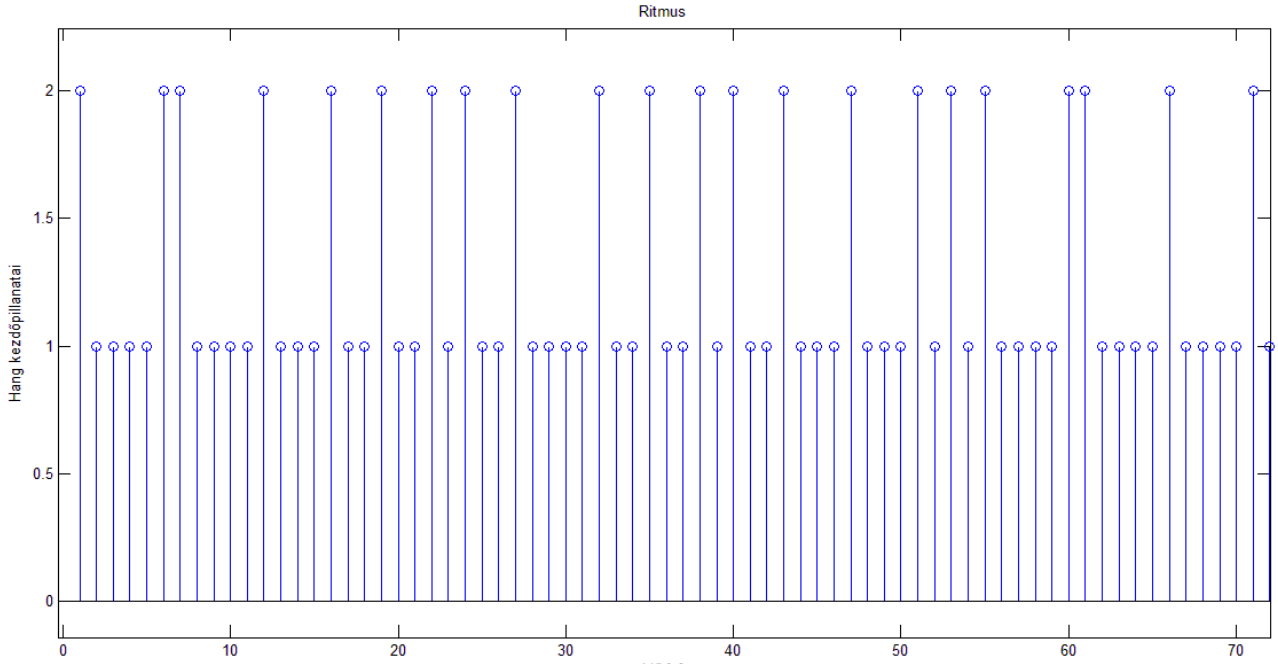
A megfelelő hangkészlet kiválasztása után az algoritmus véletlenszerűen választja ki a hangok sorozatát, figyelembe véve az időpillanatot is.

Készítettem olyan improvizációt, ahol a dallam csak pentaton hangokból állt, olyat, ahol csak a megfelelő akkordokból, és olyat, ahol ezt vegyesen használtam. A tapasztalatom az, hogy még véletlenszerű ritmus mellett is, az akkordhangokkal már egészen jó minőségű dallamot lehet generálni.

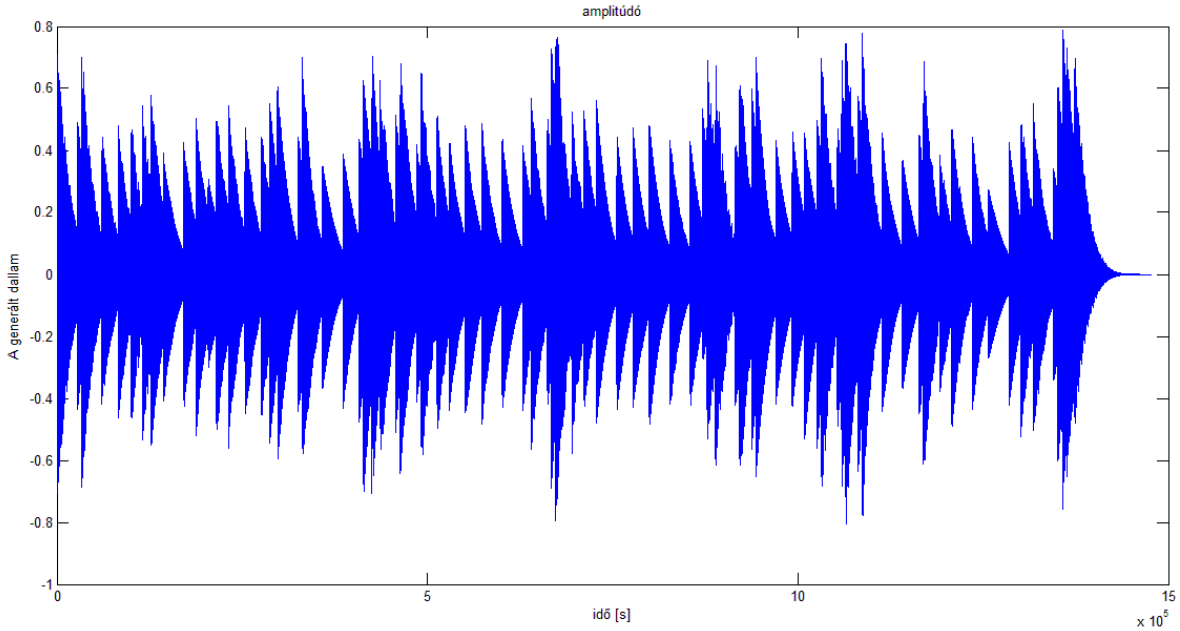
8.2.3 A generált dallam ritmusa

Ahhoz, hogy az alapdallamra illeszkedjen az általunk generált zene, ahhoz ritmusban követnie kell azt. Ehhez az alapdallam időtartamát felosztjuk a tempó szerint kicsi, egyenlő részekre. Ezen periódusidőnek az 1, 2^1, 2^2, 2^3, 2^4, 2^5-szeresét vesszük, ezek lesznek a ritmusértékek.

Ezután véletlenszerűen választunk ritmusértékeket, majd az eltároljuk őket egymás után. A 36. ábrán látható egy ilyen generált sorozat. Ahol 2 értéket látunk, onnan fog indulni egy adott hang. Az adott ritmusérték kezdőpillanatában lejátszuk az oda választott hangot. Egyelőre ez egy egyszerű szinuszjel, megfelelő frekvenciával, és exponenciálisan lecsengő amplitúdóval.



36. ábra: Ritmus generálása



37. ábra: a generált dallam

8.3 Valós idejű zenei improvizáció készítése

A cél egy olyan alkalmazás készítése volt, ami valós időben felismeri a zene tempóját, rászinkronizálódik, és az éppen elhangzott akkordok alapján zenei improvizációt készít. Ehhez a feladathoz a JUCE keretrendszert használtam.

Az ábrák készítéséhez, valamint a hibakereséshez a *Visual Studio*-ba beépíthető *ArrayPlotter* kiegészítő programot használtam fel [33].

8.3.1 JUCE

A JUCE egy platformfüggetlen keretrendszer, ahol C++ kódot tudunk készíteni, és *Windows, Mac OS, Linux, iOS* és *Android* operációs rendszerre fordítható állapotra tudjuk könnyen beállítani. A kód fordításához külön fejlesztő környezet szükséges. A JUCE ebből is többfélét támogat. Én a *Microsoft Visual Studio 2012*-t használtam. A JUCE folyamatosan fejlődik, jönnek újabb és újabb frissítések. A munkámhoz a *Grapefruit JUCE 4.2* verziót használtam [31].

A JUCE-al többfajta programot tudunk készíteni: *GUI Application, Animated Application, Console Application, Audio Application, Audio Plug-In, Static Library, Dynamic Library*.

A JUCE számunkra fontos tulajdonsága, hogy az audio stream kezelése, audio fájl írása és olvasása beépített függvényekkel könnyen megoldható. Egyszerűen lehet vele *VST, VST3, AU, RTAS* és *AAX* formátumú plugin-okat készíteni, ha megadjuk a hozzájuk tartozó *SDK* elérési útvonalát. Ezenkívül teljes körű MIDI szolgáltatást nyújt.

A zenei improvizációt készítő programom kiindulási alapjának a JUCE demó kódok közül a *SimpleFFTExample* nevű audio alkalmazást választottam. Ez a kód a mikrofonbemenetre érkező jelet 1024 mintából Fourier- transzformálja, majd az eredményvektort minden 1024 mintára jobbról balra úsztatva folyamatosan megjeleníti egy vonalban, az intenzitások méretét színekkel jelölve.

A JUCE-ban audio alkalmazás készítéséhez létre kell hozni egy *AudioAppComponent* osztályt. Ennek három fontos tagfüggvénye van, amit felül tudunk definiálni [31]:

- ***prepareToPlay*:** az audio stream-et előkészíti a lejátszás előtt, és *„prepared”* állapotba helyezi
- ***getNextAudioBlock*:** Az audio hardver callback függvénye fogja meghívni periódikusan. A beállított fix bufferméretnek megfelelő mennyiségű adatot játszunk le, megadott mintavisszajátszási frekvenciával. Ezen értékeket a *samplesPerBlockExpected sampleRate* változókból tudjuk kiolvasni.
- ***releaseResources*:** lejátszás végén az audio stream-et *„unprepared”* állapotba helyezi

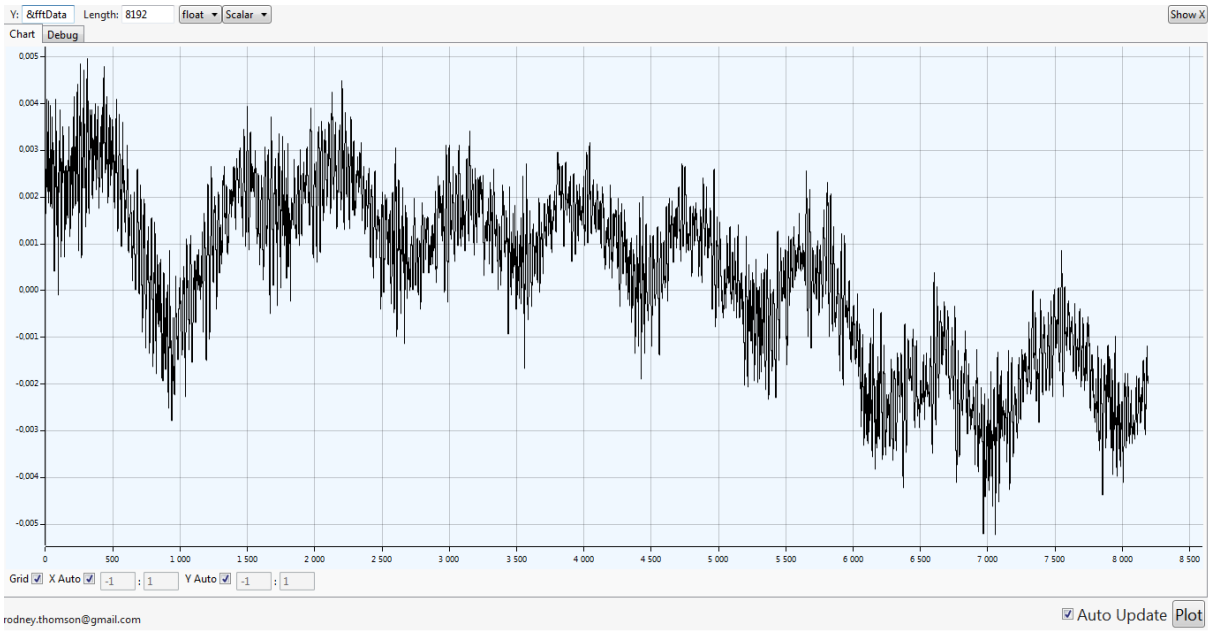
Ezenkívül az *AudioAppComponent* osztályhoz interfészeket definiálhatunk, mint például nyomógombok, csúszkák, címkék, egér. A programnak egy tetszőleges méretű ablakot adhatunk meg, amire az egyes GUI elemeket helyezhetjük [32].

8.3.2 Akkordfelismerés

A valós idejű programban használt akkordfelismerés egyszerűbb algoritmust használ, a 6. fejezetben bemutatotthoz képest. Ezt fogjuk végignézni lépésről lépésre.

8.3.2.1 Adatgyűjtés

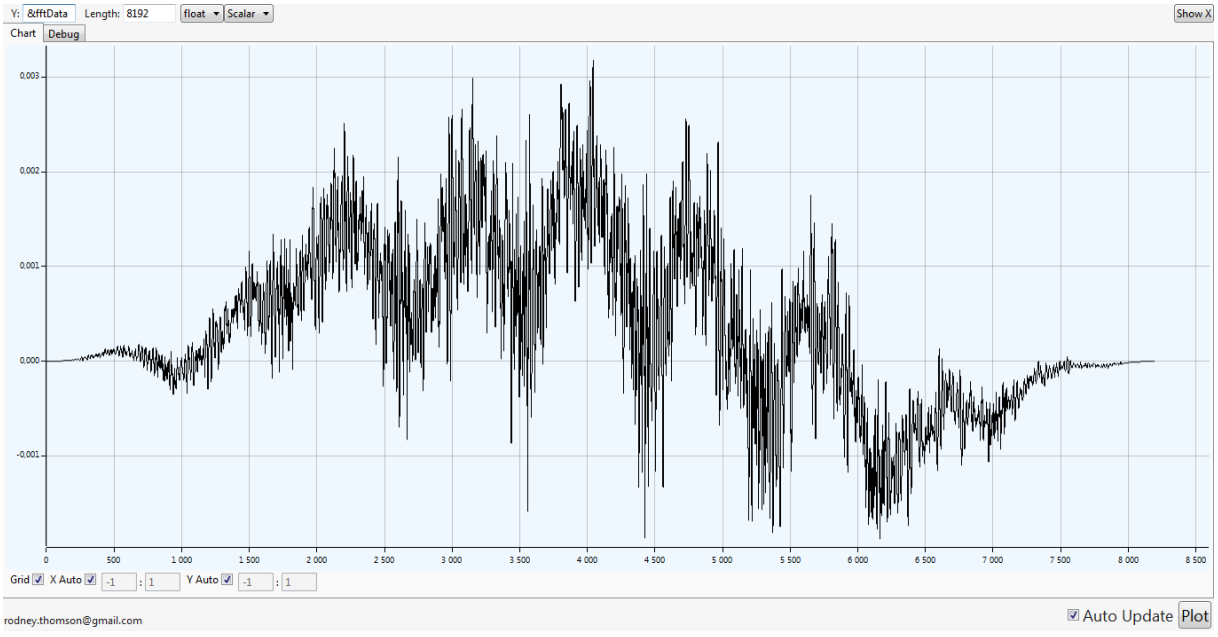
A számítógép a mikrofonbemeneten kapja zenei jelet, 44100 Hz mintavételi frekvenciával. A *getNextAudioBlock* függvényt a hangkártya periódikusan hívja meg, minden 448. beérkező minta után. Mivel a 44100 Hz mintavételi frekvenciával a 448 minta kevés megfelelően pontos FFT előállításához, ezért definiálunk egy *fifo* nevű változót, amit a *getNextAudioBlock* fog tölteni egyesével a *pushNextSampleIntoFifo* segítségével. A *fifo* méretének a megválasztása kompromisszum kérdése, ugyanis minél nagyobbnak választjuk, annál pontosabb lesz az FFT felbontása, viszont a valós idejű működés miatt nem engedhetünk meg túl nagy méretet. Végül a kódgyorsítást is figyelembe véve 2^13=8192 méretű buffert választottam. Ez 5,34 Hz-es frekvenciafelbontást eredményez. Ehhez az alkalmazáshoz ez a pontosság jónak bizonyult. Ha megtelt a *fifo*, akkor a *pushNextSampleIntoFifo* függvény átmásolja a *fifo* adatait az *fftData* bufferbe. Ezzel a kettős bufferelléssel tudjuk megoldani azt, hogy addig se veszítsünk adatot, amíg a bejövő adatok feldolgozását végezzük. A 38. ábrán látható az *fftData* buffer tartalma egy adott pillanatban. Az buffer tartalmát az ArrayPlotter segítségével rajzoltam ki [33].



38. ábra: audio input, 8192 minta

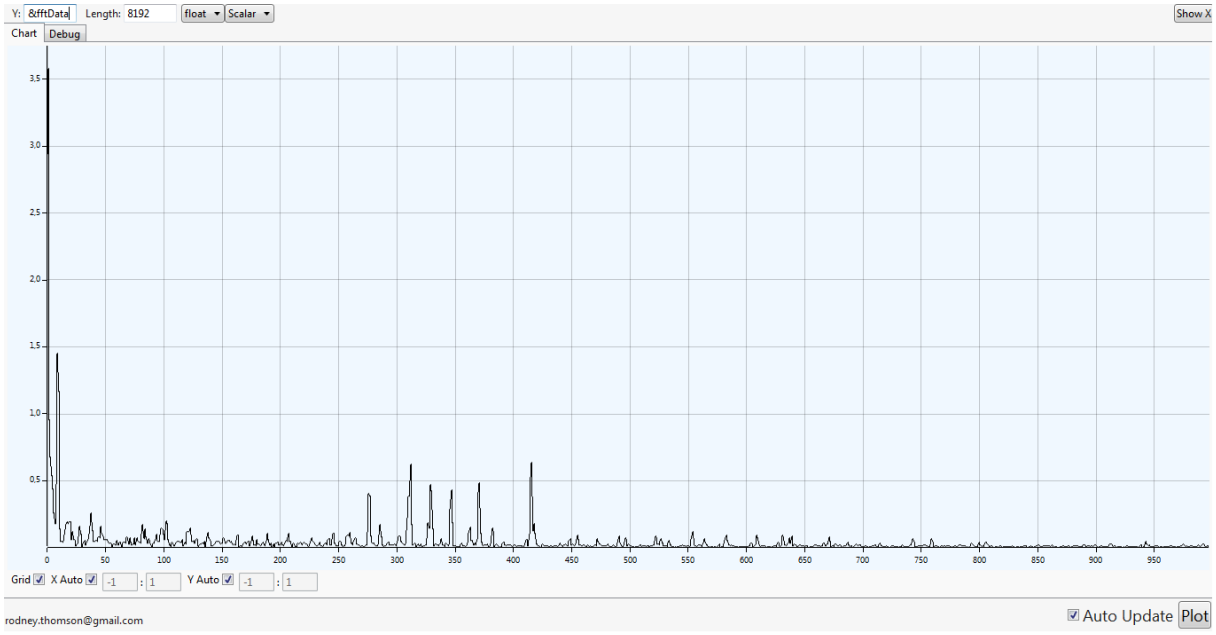
8.3.2.2 Jelfeldolgozás

A bejövő zenei jel 8192/44100 = 0.186 másodperces szakasza tehát az *fftData* bufferbe kerül. Ezután a spektrumszivárgás csökkentés érdekében ablakozunk. Ehhez a feladathoz a Hann-ablakot választottam. Az ablakozott jel a 39. ábrán látható.



39. ábra: bejövő audio jel ablakozva (8192 minta)

Az ablakozott jelet Fourier-transzformáljuk. A JUCE-ban van beépített FFT osztály, amivel könnyen el lehet végezni egy bemeneti vektor Fourier-transzformációját. A transzformáció eredményét a program visszaírja a bemeneti vektorba, esetünkben az *fftData* bufferbe. A bemeneti jel Fourier-transzformáltjának a számunkra érdekes részlete a 40. ábrán látható.



40. ábra: bemeneti jel Fourier-transzormáltja

Ahhoz, hogy meghatározzuk, milyen akkord szólalhatott meg, elkészítjük a chroma-vektort. Ez egy 12 elemű vektor, aminek az egyes elemei az egyes zenei hang típusokat jelölik a következő indexelés szerint:

Chroma-vektor											
0	1	2	3	4	5	6	7	8	9	10	11
C	C#	D	D#	E	F	F#	G	G#	A	A#	H

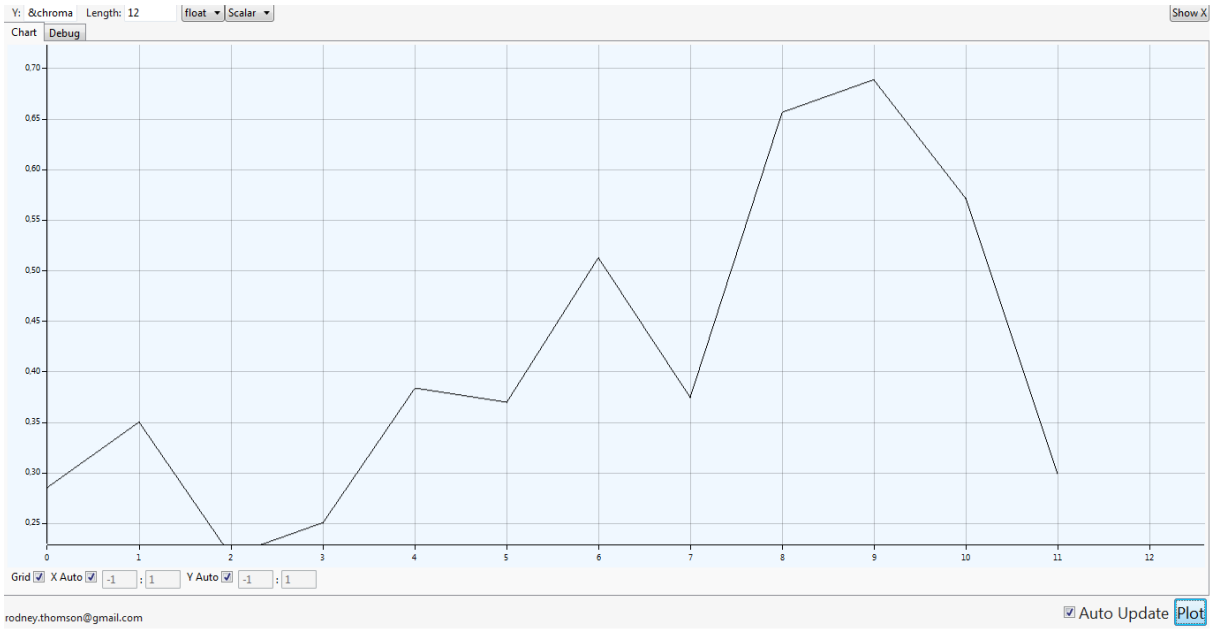
9. táblázat: Chroma-vektor elemei

Készítésének a C kódja a következő:

```
int noteType = 0;
float chroma[12] = {0};
for(int i=36; i<97; i++){        //C2-C7
   chroma[noteType] += fftData[midiToIndex(i)];
         noteType = (noteType+1)%12;
}
```

A kódból látható, hogy csak a 36-os midi kódtól a 96-osig vizsgáljuk a zenei hangokat. Ezek a C2 és a C7 közötti zenei hangok. Végigmegyünk a spektrum azon binjein, amik az adott zenei hangnak felelnek meg. Itt megjegyzendő, hogy a zenei hang frekvenciájához legközelebbi frekvenciát választottam az n*8192/44100 Hz (n=1,…,8192/2+1) frekvenciák közül. A zenei hanghoz legközelebbi frekvenciabinhez

tartozó intenzitás értéket hozzáadjuk a chroma megfelelő eleméhez. Egy ilyen elkészült chromát láthatunk a 41. ábrán. A lejátszott akkord egy F#-moll volt.



41. ábra: chroma-vektor, F#-dúr

Ezután minden akkordra kiszámítjuk, hogy a chromában mekkora az akkord hangaihoz tartozó intenzitások összege. A 24 lehetséges érték közül a maximálisat választjuk ki, és azt tároljuk el az akkordfelismerés eredményeképpen.

A 41. ábrán látható chromában az F#-dúrnak megfelelő 1-es, 6-os és 9-es indexek összege lesz a legnagyobb, így a program felismeri azt.

8.3.3 Tempódetektálás és a zenei improvizáció szinkronizálása

8.3.3.1 Tempódetektálás

A tempódetektálást alapvetően 6.2 fejezet szerint valósítottam meg itt is, viszont egy-két dolgot módosítani kellett, annak érdekében, hogy a program képes legyen valós idejű működésre.

A tempódetektáláshoz korábban ismertetett okokból körülbelül 10 másodperces zenei részre van szükségünk. Ehhez létrehozzuk a *tempoBuffer* tömböt. Egyszerűség kedvéért 448.000 bejövő minta után végezzük el a tempódetektálást (a *getNextAudioBlock* minden 448. beérkező minta után hívódik meg). A számítás csökkentése érdekében a bejövő jelet 5-el decimáljuk, így a *tempoBuffer* vektort elég 89600 eleműnek deklarálnunk. A *tempoBuffer*-t a *getNextAudioBlock* fogja feltölteni.

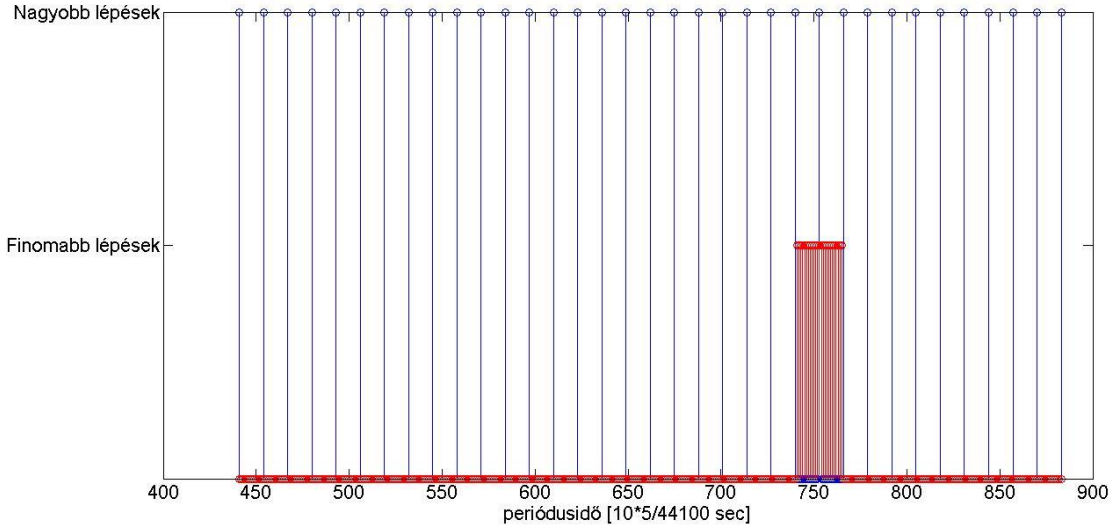
A tempófelismerés periódikusan történik, ehhez szükséges definiálnunk egy timer-t, amihez ugyancsak van előre megírt JUCE osztály. A timer előre definiált idő alatt fut le. Ha lefutott, akkor reseteli magát, és meghívja a *timerCallback* függvényt, ahol a tempófelismerés történik. A timert 1000/6 = 166.7 ms-ra állítjuk be. Választás oka egyrészt itt is a valós idejű követelmények teljesítése. Ezt azért szükséges, mert a *timerCallback* függvény önmagában nagyjából 120 ms alatt fut le, illetve ennél sokkal

nagyobb idő már nagyobb lenne az *fftData* feltöltéséhez szükséges időnél (8192/44100 = 0.1857 sec).

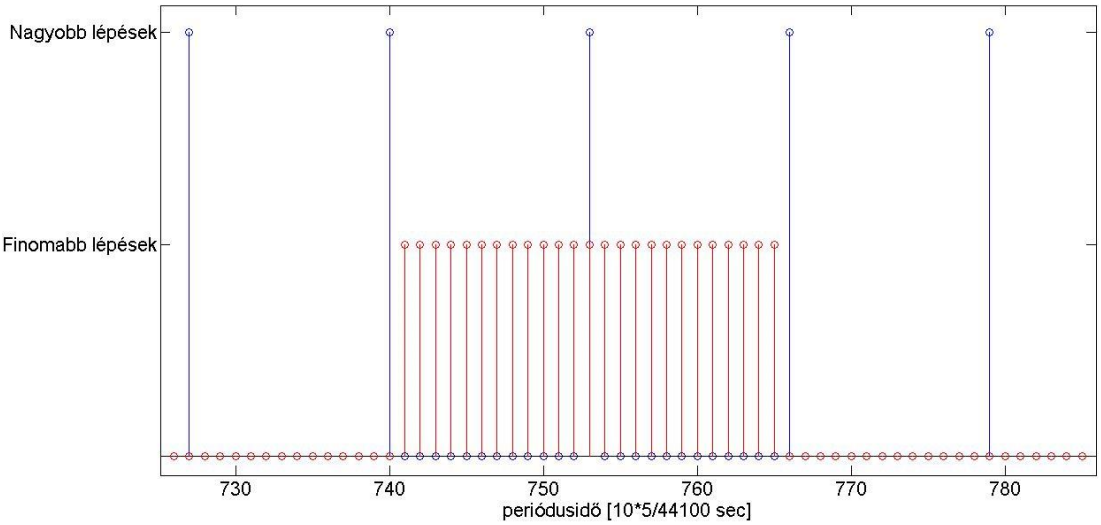
A *timerCallback* függvényben történik a *tempoBuffer*-ben levő nagyjából 10 másodperces szakasz feldolgozása. Itt is hasonlóan, mint a nem valós idejű algoritmusban, először a jel abszolútértékét vesszük, aluláteresztő szűrővel szűrjük, majd elkészítjük a differenciált jelet. A választott szűrő JUCE beépített IIR szűrője, 30 Hz törésponti frekvenciával.

Az igazán lényeges különbség a konvolúció esetén van. Itt a következő módosításokat végeztem a programgyorsítás érdekében:

- A tempót csak a 60 bpm és a 120 bpm között keressük. Így ha például egy 170 bpm-es tempójú zenét vizsgálunk, akkor így 85 bpm-et fogunk felismerni. Mivel a hangok lehetséges időtartamai 2 hatványaival vannak definiálva, ez nem jelent különösebb problémát.
- A fésűszűrővel való konvolúció leegyszerűsíthető legfeljebb (N+M-1)*F darab összeadásra, ahol N a mintavett jel hossza, M a fésűszűrő hossza, F pedig a fésűszűrőben található egységimpulzusok száma.
- A konvolúció effektív értékének a meghatározásához a négyzetre emelést kiírjuk két tag szorzataként, és nem a *pow()* C++ függvényt használjuk.
- A tempófelismerésnél a 60 és a 120 bpm között először nagyobb léptékben keressük a maximális tempót, majd a maximum környezetében újra keressük, nagyobb felbontással. Erre láthatunk egy példát a 42. és 43. ábrán.



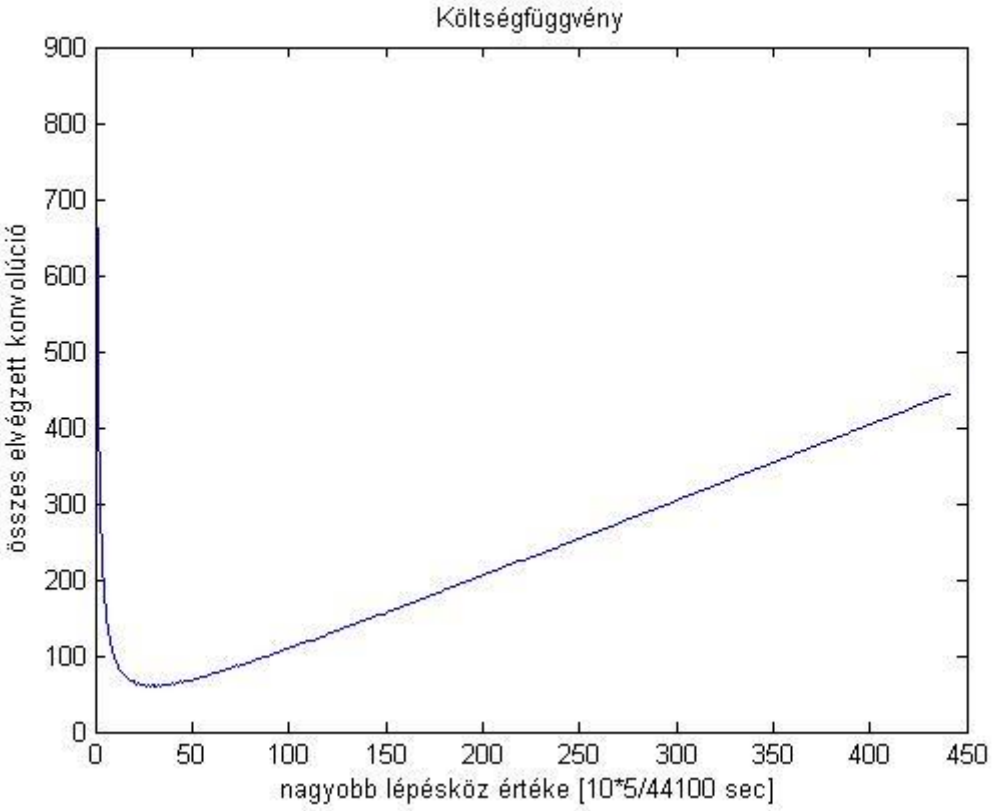
42. ábra: tempó keresése nagyobb és finomabb periódusidő lépésekkel



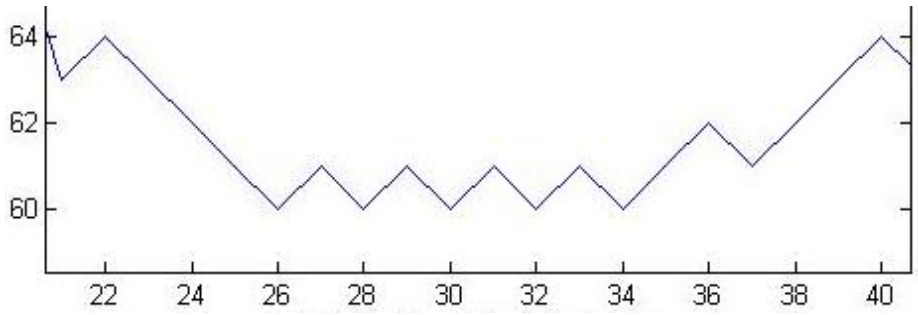
43. ábra: tempó keresése finomabb periódusidő lépésekkel

A finomabb felbontásnál a decimált jelben 10-esével növelem a fésűszűrő távolságát. Ez azt jelenti, hogy a felismert periódusidő maximális tévedése 50/44100 = 0.0011 másodperc lehet. Tekintve, hogy 10 darab 5/44100 másodperc pontos periódusidő mérés szórása 3.9441*5/44100 másodperc volt, nem is érdemes ennél pontosabban mérni. Ezenkívül a nagyobb léptéket kell még definiálni. Ennek a kiválasztása az alapján történt, hogy az összes elvégzett konvolúció száma a lehető legkevesebb legyen, a következőket figyelembe véve. 120-tól 60 vagy annál kicsit kisebb bpm-ig keresünk. Ez utóbbi azért kell, hogy a tartománynak osztója legyen a nagyobb lépésköz. Ezenkívül a nagyobb felbontásnál a megtalált maximum két szomszédos, már megvizsgált tempója között nézzük át újra, 10-esével lépkedve. Az optimumkereséshez használt kód:

```
x=1:1:441;
range=882-441;
a=ceil(range./x);
possibleRanges=a.*x;
costFunc=x+2*possibleRanges./x;
find(costFunc==min(costFunc))
```



44. ábra: elvégzendő konvolúciók száma a nagyobb lépésköz függvényében



45. ábra: a költségfüggvény minimumai

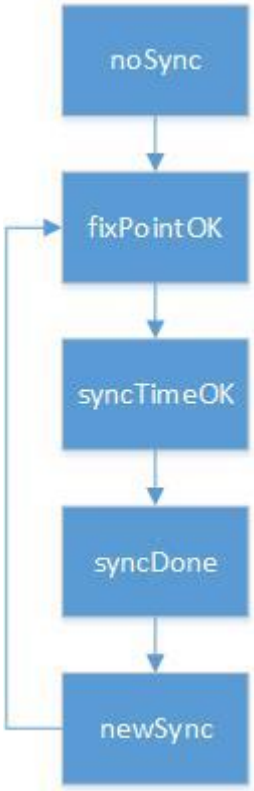
A 44. ábrán látható az elkészített költségfüggvény. Ennek a minimumpontjai láthatóak a 45. ábrán. A 26, 28, 30, 32, 34 lehetséges lépésszámok közül a 34-et választottam. Ezen választások mellett 60 darab konvolúciót kell elvégezni.

A tempó felismerése után a megtalált tempóval ismét elvégezzük a konvolúciót a differenciált jellel, majd megkeressük annak a maximumát. Az ennek megfelelő időpont lesz egy fix beütés helye. Az időt a program indítása óta mérjük méghozzá úgy, hogy az *absoluteTime* változót folyamatosan inkrementáljuk a *getNextAudioBlock* függvényben, minden beérkező mintára eggyel. A fix beütés idejét az *absoluteTime* változó segítségével

eltároljuk a *fixPoint* nevű változóban. A zenei improvizációt ehhez a ponthoz fogjuk igazítani.

8.3.3.2 Szinkronizáció

A tempó a zene közben némileg elcsúszhat, illetve a tempódetektálás sem teljesen pontos. Ezt kompenzálhatjuk azzal, ha a tempót bizonyos időközönként újraszámítjuk a legkorábbi információkból, és az alapján újra szinkronizálódunk a zenére. Ezt programtechnikailag egy állapotgéppel oldottam meg, aminek a folyamatábráját a 46. ábrán láthatjuk.



46. ábra: a szinkronizáció folyamata

A program az elindulásakor *noSync* állapotban van, ekkor az audio kimenet le van tiltva. Elkezdjük gyűjteni az adatokat a *tempoBuffer*-be, majd ha beérkezett a 89600. minta is, akkor a timer callback függvényében lefut a tempódetektálás, és a fix pont megkeresése. Ennek hatására a rendszer a *fixPointOK* állapotba kerül. Ezzel egyidejűleg elkezdjük a *tempoBuffer*-t tölteni az új adatokkal. Ezután amikor újra meghívódik a *getNextAudioBlock* függvény, akkor beállítjuk a fix pontra szinkronizált *syncTime* változót, és egyuttal *syncTimeOK* állapotba kerülünk. Ekkor a *getNextAudioBlock*-ba várunk addig, amíg a *syncTime* nem ér el egy olyan értéket, ami a felismert periódusidő egész számú többszöröse. Ha elértük, akkor *syncDone* állapotba ugrunk, és eltároljuk, hogy frissen beérkezett 448 elem közül hol járunk. Még ugyanebben a *getNextAudioBlock* függvényhívásban az audio output engedélyezve lesz, és az eltárolt érték elérése után meghívjuk először a *getNextNote* függvényt. Az első új hang keresése a tartomány közepéről indul. Ezzel egyidejűleg *newSync* állapotba kerülünk. Ezen állapotban az audio

kimenet engedélyezve van, és arra várunk, hogy a *tempoBuffer* megteljen. Ha megtelt, akkor újra tempót és fix pontot számol, majd megint a *fixPointOK* állapotba kerül. Innentől az algoritmus ciklikusan működik.

8.3.4 Zenei improvizáció készítése

Amint megtörtént a szinkronizáció, a program elkezd egyszólamú dallamot lejátszani. A következő hang generálását a *getNextNote* függvény végzi el.

8.3.4.1 Ritmus

Az új hang ritmusértékét véletlenszerűen választunk ki az 1/16, 1/8, 1/4, 1/2 értékek közül. Van egy osztályváltozónk, az *actualNoteLength*, amibe a kiválasztással egyidejűleg beírjuk a tempóból és a ritmusértékből számított időtartamot, ameddig ezt a hangot fogjuk lejátszani. Ezt a változót minden, a kimeneti csatornán kiadott minta után eggyel csökkentjük. Ha eléri a nullát, akkor újra meghívjuk a *getNextNote* függvényt.

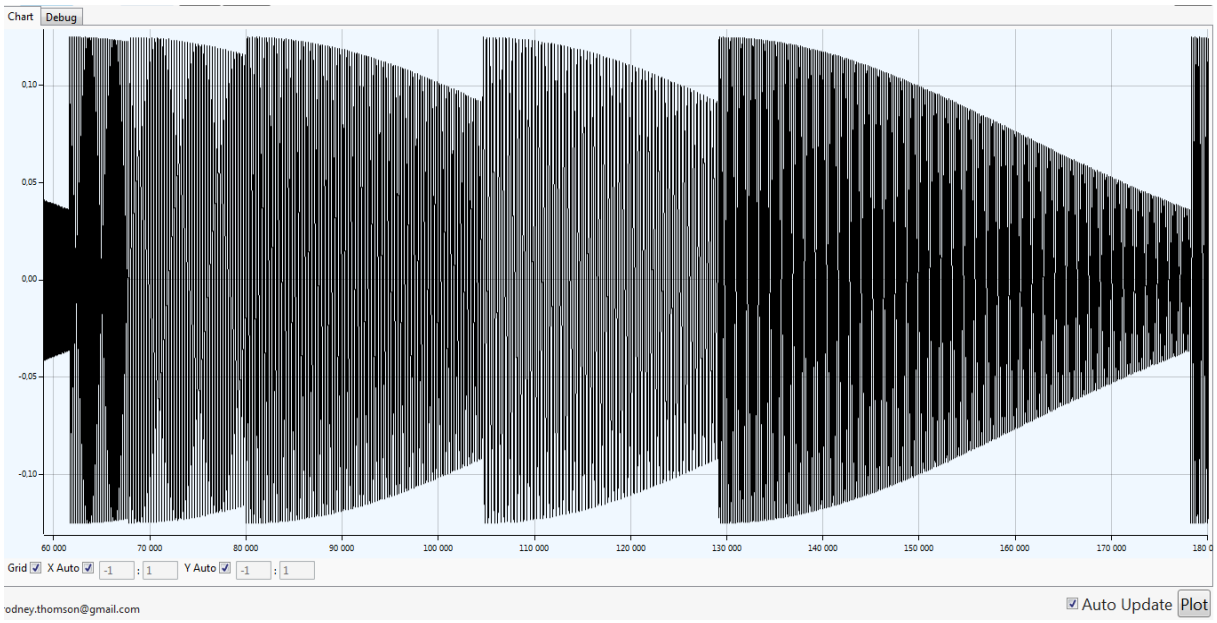
8.3.4.2 Hangok

Ebben az alkalmazásban csak a 3, 4. és az 5. oktávban szereplő hangok közül választunk. Ezenkívül itt most csak az aktuálisan felismert akkord hangjait játszuk le, és az új kiválasztott hang semmi esetben sem lehet az előzőnél nagyobb távolságra, mint egy oktáv.

A *getNextNote* függvényt a *getNextAudioBlock* függvényben hívjuk meg, aminek gyorsan kell lefutni. Ezért az aktuális akkordhoz a következő lejátszandó hangot az alábbi módon határozzuk meg. A program elején feltöltünk egy *chordNoteLUT* mátrixot, ami tartalmazza, hogy melyik akkordhoz milyen hangok tartoznak a 3, 4. és az 5. oktávból. Ez egy 24x36-os mátrix, aminek az sorai a 24 féle akkordot jelentik, sorban először a dúrokkal (0-11.), utána a mollokkal (12-24), oszlopai pedig az egyes hangokat. Ha az egyik akkordhoz hozzátartozik az egyik hang, akkor a mátrix megfelelő eleme 1-es, egyébként nulla. Egy új hang kereséséhez az aktuális hang indexétől 12-re lefelé és felfelé lévő hang indexeket keressük meg. Ha kiszámított index negatív, akkor helyette 0-t írunk, ha pedig nagyobb, mint 36, akkor 36-ot. Így megkapjuk azt a tartományt, amiben benne vannak a kiválasztható hangok. A mátrixban, a legutóbb felismert akkord által kijelölt sorban megvizsgáljuk ezt a tartományt. Megkeressük benne az 1-eseket. Az ezeknek megfelelő hangok közül véletlenszerűen választunk egy újat.

Például, ha az előző hang a 27-es indexű (D#5 hang, zölddel jelölve) volt, és legutóbb egy C-dúr akkordot ismertünk fel, akkor a tartomány, amiből választhatunk a 15-ös indextől a 36-osig tart (kékkel és zölddel jelölt hangok). A lenti táblázatban látható, hogy ebben a tartományban 5 darab 1-es van. Az ezeknek megfelelő hangok közül fogjuk kiválasztani a következő lejátszandó hangot.

	3. oktáv												4. oktáv												5. oktáv											
C	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0



47. ábra: zenei improvizáció, audio kimenet

A 47. ábrán láthatunk egy, a program által készített valósidejű zenei improvizációt. A dallamot egyszerű, exponenciálisan lecsengő szinusz jelek adják.

8.3.5 Szekvenciafelismerés és a következő akkord jóslása

A zenében, főleg a közös örömzenélésekben (jam-elés), gyakran előfordul, hogy egy rövidebb akkordsorozatot többször is megismételnek. Ezt itt most akkordszekvenciának fogjuk nevezni. A közös, előre nem megbeszélt struktúrájú zenélés során például elképzelhető egy olyan szituáció, hogy a gitáros elkezd pengetni négy akkordot egymás után. A második akkordkör befejeztével a többi zenész sejti, hogy a könnyebb összhang érdekében ez a szekvencia még ismétlődni fog többször is, így a többi zenész gyakorlatilag jóslást tesz, és így nagy valószínűséggel előre tudja, hogy az egyes ütemekben melyik akkord fog következni. Abban az esetben, ha a gitáros váltani akar, akkor kell némi idő, hogy ezt a váltást a többi zenész le tudja követni.

A saját valós idejű zenei improvizációt készítő szoftveremben ezt a jelenséget akartam leprogramozni annak érdekében, hogy a gép is előre tudhassa, hogy milyen akkord jön, és ne az egy időszelettel korábban felismert akkord alapján játssza a dallamot. Ehhez szükséges egy akkordszekvencia-felismerő algoritmus.

Az algoritmust a következő feltételezésekkel készítettem. Egy akkord minimum 1 másodpercig tart. Egy akkordszekvencia minimum 2 akkordból áll, így annak minimális hossza 2 másodperc, míg a maximális hosszát 10 másodpercre korlátoztam. A programom egy cirkuláris bufferbe gyűjti össze az utolsó 120 felismert akkordot, ami 20 másodpercnyi információnak felel meg. Ha már összegyűjtött 120 darabot, onnantól lép életbe a lehetséges akkordszekvenciát is figyelembe vevő improvizáció készítése. Ha az algoritmus talál szekvenciát, akkor a szekvencia legrégebbi eleme lesz a következő hang

generálásához felhasznált akkord. Ha nem talál, akkor marad az egy időszelettel korábban felismert akkord erre a célra. Tehát ha a program felismer egy szekvenciát, akkor az éppen felismert akkordot lecserélem a szekvencia legrégebben lejátszott elemére. A szekvenciafelismerést úgy végezzük, hogy a bufferben lévő különböző hosszú akkordsorozatokat összehasonlítjuk az időben azt megelőző ugyanolyan hosszú akkordsorozattal. Ha 80%-ban megegyezik, akkor úgy minősítjük, hogy találtunk egy szekvenciát. A megengedett 20% hiba segít, hogy néhol pontatlan akkordfelismerés mellett is felismerhessünk egy szekvenciát. A szekvenciakeresés minden új, felismert akkord után valós időben megtörténik, ezzel a hibás szekvenciafelismerések számát csökkenthetjük.

Megjegyzendő, hogy az akkordfelismerés nem tempó szerint van, hanem fix bufferhosszal. A buffer 8192 méretű. A választás egyrészt azért esett erre a számra, mert 2 hatvány, és ezáltal az fft gyorsabb, másrészt mert ez a méret egy kompromisszum a valósidejűség és az fft frekvenciafelbontása között. 8192 alá méretben már nem érdemes menni, mert akkor a basszushangok frekvenciái az fft-ben már nem választhatóak el, fölé pedig azért nem, mert az már jelentős időkésést visz be a rendszerbe. Így viszont a zene negyedhangjainak a periódusideje és a fix buffermintavétel frekvenciája általában nem koherens. Ez nem minden esetben jelent problémát, de azokban az esetekben, ahol igen, ott segíti a szekvencia felismerését az, hogy csak 80%-ig kell, hogy megegyezzen két akkordsorozat.

8.3.6 Felhasználói felület

A felhasználói felület egyszerű kialakítású. A *Start* gombbal elindíthatjuk a programot, ami ezek után kicsit több, mint 10 másodperccel felismeri az aktuálisan játszott zene tempóját. Ezek után nem sokkal rászinkronizálódik a zene ütemére, és elkezd egyszólamú dallamot játszani. A tempót 10 másodpercenként újra kiszámítjuk, illetve újra szinkronizálunk. A program a tempó aktuális értékét bpm egységben az ablak közepén jeleníti meg. A programot a *Stop* gombbal állíthatjuk le. Következő elindításkor mindent alapállapotba helyezünk, így újra ki kell várni a 10 másodpercet, hogy a számítógép improvizáljon.

48. ábra: a zenei improvizációt készítő program GUI-ja

8.3.7 Tesztelés, eredmények

Az elkészült program forráskódját, és az abból lefordított futtatható fájlt (*RT_Impro.exe*) a mellékletben közzétettem. Ezenkívül két *mp3* kiterjesztésű audio demót is mellékeltem. Ebből az egyikben a programom számítógép által lejátszott akkordokra készít improvizációt, a másikban pedig az általam zongorán játszott akkordokra. A felvételben megfigyelhető, hogy a generált improvizáció hangjai legtöbbször konszonánsak. Az egyes hangok véletlenszerűen megválasztott időtartama még néhol generál furcsa ritmusképleteket. A generált dallam minősége végső soron szubjektív.

9 Összefoglalás és továbbfejlesztési lehetőségek

Az algoritmusaim több területen is továbbfejleszthetőek.

- A kutatásaimat nagyban nehezíti, hogy nem rendelkezek olyan akkordadatbázissal, amelyben kellő mennyiségű és minőségű felcímkézett akkord szólalna meg. Az interneten csak C. Harte felcímkézett PCP adatbázisát [23], illetve J. Osmalskyj egyhangszeres, felvételenként egyakkordos felcímkézett mintáit találtam meg [25]. Az igazán hasznos az olyan adatbázis lenne, amelyben konkrét dalok rövid zenerészletei lennének felcímkézve a megfelelő akkordokkal. Ez a feladat extrém módon sok időt vesz igénybe, valószínűleg ezért nem találtam ennek megfelelőt az interneten. A további munkámban egy ilyen adatbázis elkészítésével tudnám fejleszteni az algoritmust, illetve ez a tesztelésben is nagyon hasznos lenne.
- A popzenében a leggyakoribb az egyszerű dúr és moll hármashangzatok használata, ezenkívül ritkábban, de előfordulnak más jellegű hármashangzatok (szűkített, bővített), illetve négyeshangzatok is. Hasznos lenne a programot kiterjeszteni olyan módon, hogy ne csak hármashangzatokat legyen képes felismerni, ugyanis ez a plusz információn kívül a hibás találati arány értékét is csökkentené.
- Az időszeletek szeparációját tempódetektálás, és szinkronizáció segítségével oldottuk meg. Ez az algoritmus egészen pontosan működik, viszont ez sem tökéletes. Ennek a további fejlesztése is javíthatná az akkordfelismerés pontosságát.
- Lehetne gondolkodni olyan másfajta jelfeldolgozási módszeren, mellyel PCP vektort készítünk. Egy újfajta PCP vektorral bővítve az MPCP vektort akár jobb leíróját kaphatnának a rövid zenerészleteknek.
- Az érzékelőmodellnél gyakori megközelítés az egyszerű Gauss modell. További munkám során ki lehetne próbálni az ennél komplexebb, Gauss-keverékek módszerét is.
- Az akkordfelismerésre ki lehetne dolgozni egy mély neurális hálózaton alapuló módszert is.
- A gépi improvizációt színesebbé tehetnénk, ha komplexebb dallamokat játszanánk le. Erre az egyik lehetséges ötlet, hogy nem csak az adott harmónia hangjait használnánk fel, hanem skálafelismerés alapján ezt egy szélesebb körbe terjesztenénk ki, és emellett az egyes hangok gyakoriságát is előre meghatároznánk. A másik ötlet, hogy improvizatív módon vagy egy előre legyártott adatbázisból tempó és dallamszekvenciákat használhatnánk fel.
- Az improvizált dallam minőségét javíthatná egy zenei stílus felismerését megvalósító algoritmus. Ezt felhasználva a gépi improvizációt végző szoftver felismerhetné a zene stílusát, és arra jellemző, előre megírt dallamszekvenciákat, vagy automatikusan generált hangokat improvizálhatna rá.

- A gépi improvizációnál a szekvenciafelismerés pontosabb lehetne, ha a háttérben történne egy tempó alapján történő akkordfelismerés is. Ez amiatt, hogy a zene tempója általában kicsit ingadozik, illetve hirtelen meg is változhat, nem könnyű feladat.
- Összetettebb gépi improvizációt készíthetnénk, ha szimultán többféle hangszínt használnánk.
- Végül a zeneelmélet alaposabb tanulmányozása is segíthetne az mind az akkordfelismerő, mind a gépi improvizációs algoritmus fejlesztésében. Ennek segítségével több, a zene természetét leíró információt lehetne felhasználni a RMM-ben, illetve ennek segítségével komplexebb zenei improvizációt lehetne programozottan készíteni.

10 Összegzés

A munkámnak tehát a célja automatikus akkordfelismerés és gépi improvizáció készítése volt. A dolgozatom első fele az akkordfelismerést tárgyalja, itt célnak egy olyan szoftver készítését tűztem ki, amely dúr és moll hármashangzatokat képes felismerni. Egy ilyen alkalmazás hasznos lehet többek között zenetanulásra, a zeneelmélet alaposabb megértésére, illetve gépi tanításhoz alkalmas információk szerzésére. A szakirodalomban megismert irányokat háromféle csoportra osztottam: mintamegfeleltetéses, rejtett Markov-modelles és neurális hálózatot felhasználó algoritmusok. Több kísérlet után a rejtett Markov-modelles megközelítést választottam, és a Bello és Pickens által készített algoritmust vettem alapul. Az általam készített algoritmus a bementként megkapott zenei jelet a tempója alapján darabolja fel rövid, időben átlapolódó szakaszokra, majd abból készíti el a DFT spektrumot, mint az adott diszkrét időszelethez tartozó megfigyelést. Az általam készített rejtett Markov-modell állapotátmenetvalószínűségi mátrixát zenék akkordmenetéből készített statisztika alapján inicializáltam, a megfigyelési modellt pedig négy különböző módszer eredményének az összegeként állítottam elő. A modell a kezdeti paraméterekkel és a megfigyelések alapján kiindul egy állapotból, amit az elvárásmaximalizációs algoritmussal tudunk iteratív módon az adott megfigyeléssorozathoz igazítani. Végül a legvalószínűbb akkordsorozatot az RMM új paraméterei alapján a Viterbi-algoritmussal tudjuk meghatározni. Az algoritmus tesztelése során több dolgot állapítottam meg. Elsőként lemértem, hogy a saját ötletként bevezetett MPCP vektor jobb zenei leíró, mint a szakirodalomban gyakran használt társai. Másodrészt láthattam, hogy az általam készített tempódetektáló algoritmus viszonylag pontosan működik. Továbbá összehasonlítottam a saját programom kettő, a szakirodalomból ismert módszerrel, illetve a piacon található Chordify programmal. Az összehasonlítás alapjául azt választottam, hogy a szoftver az akkordváltásokat milyen jól ismeri fel. A tesztek során láthattunk, hogy az EPCP-s módszer önmagában korlátozott módon használható csak akkordfelismerésre. Megfigyelhettük, hogy a RMM nagyban javítja az akkordfelismerés eredményességét. Láttuk azt is, hogy a legismertebb online akkordfelismerő alkalmazás (Chordify) pontosságát még nem sikerült elérni. Ehhez további fejlesztés és kifinomultabb tesztelési eljárások szükségesek.

A dolgozatom második fele gépi improvizációval foglalkozik. A cél egy olyan valós idejű szoftver készítése volt, ami a számítógép mikrofonbemenetére érkező audió jelet feldolgozza, és a felismert akkordok és tempó információ alapján egyszólamú dallamot generál. Az elkészült programot a *JUCE* keretrendszer felhasználásával írtam *C++* nyelven. A program valós időben 10 másodperces gyakorisággal ismeri fel a zene tempóját, majd rászinkronizálódik, és egy egyszerűbb akkordfelismerő alkalmazás segítségével az aktuális akkordhangok felhasználásával, véletlenszerű ritmussal egyszólamú zenei improvizációt készít. Ezenkívül a program a felismert akkordok sorozatának ismétlődését is figyeli. Ha talál egy akkordszekvenciát, akkor az alapján jóslást tesz az éppen következő akkordra, és a annak hangjait felhasználva folytatja az improvizációt. Egy ilyen program végső eredményének, a generált dallamnak a minősítése szubjektív. A programot érdemes lenne továbbfejleszteni a kitekintésben leírtak alapján.

Köszönetnyilvánítás

Ezúton szeretném megköszönni Dr. Bank Balázsnak, a konzulensemnek (BME-VIK, Méréstechnika és Információs Rendszerek Tanszék) a munkám elkészítéséhez nyújtott tanácsokat, segítségeket és mindent, amit tanított nekem.

11 Ábrajegyzék

12 Irodalomjegyzék

[1] T. Fujishima, „*Realtime chord recognition of musical sound: A system using Common Lisp Music*”. In Proc. Int. Comput. Music Conf. (ICMC), pp. 464–467, Beijing, China, 1999.

[2] Kyogu Lee, „*Automatic Chord Recognition from Audio Using Enhanced Pitch Class Profile*”. Proc. of the International Computer Music Conference, pp. 306–313, New Orleans, LA, 2006.

[3] J. P. Bello and J. Pickens, „*A robust mid-level representation for harmonic content in music signals*”. In Proceedings of the International Symposium on Music Information Retrieval, pp. 304–311, London, UK, 2005.

[4] J. C. Brown, „*Calculation of a constant q spectral transform*”. Journal of the Acoustical Society of America 89 (1), pages 425–434, 1990.

[5] A. M. Noll, „*Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate*”. In Proceedings of the Symposium on Computer Processing ing Communications, pp. 779–797, New York, 1969.

[6] C. A. Harte and M. B. Sandler, „*Automatic chord identification using a quantised chromagram*”. In Proceedings of the 118th Convention of the Audio Engineering Society, paper 6412, Barcelona, Spain, May 28-31, 2005.

[7] A. Sheh and D.P.W. Ellis, „*Chord segmentation and recognition using EM-trained hidden markov models*”. In Proceedings of the 4th ISMIR, pages 183–189, Baltimore, Maryland, October 2003.

[8] J. Osmalskyj, J.-J. Embrechts, S. Piérard and M. Van Droogenbroeck, „*Neural networks for musical chords recognition*”. INTELSIG Laboratory, University of Liège, Departement EECS. In]ournees d'informatique musicale, 2, pp. 9- 11. Mons, Belgium, 2012.

[9] W. Michael Lai, David H. Rubin, David Rubin, Erhard Krempl, „*Introduction to Continuum Mechanics*”. Elsevier, 4th edition, September 3, 2009.

[10] T. Hastie, R. Tibshirani, and J. Friedman, „*The elements of statistical learning: data mining, inference, and prediction*”. Springer Series in Statistics. Springer, second edition, September 2009.

[11] M. Deza and E. Deza. „*Encyclopedia of Distances*”. Springer, 16 April, 2009.

[12] Horváth G. (szerk.), Altrichter M., Horváth G., Pataki B., Strausz Gy., Takács G., Valyon J., „*Neurális hálózatok*”, Budapest, Panem Kiadó, 2006.

[13] Fülöp Tibor, „*Zeneszámok hangnemének automatikus felismerése*”, Szakdolgozat, BME-MIT, 2012. május.

[14] Márkus Tibor, „*Akkordok I. – Hármashangzatok*”, NSZFI Zenész alapmodul, 1436-06

[15] Jan Vanek, Lukas Machlica, Josef Psutka, „*Estimation of Single-Gaussian and Gaussian Mixture Models for Pattern Recognition*”. Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 18th Iberoamerican Congress,

CIARP 2013, Havana, Cuba, pages 49-56, University of West Bohemia in Pilsen, Faculty of Applied Sciences, Department of Cybernetics, November 2013.

[16] M. E. P. Davies and M. D. Plumbley, „*Beat tracking with a two state model*". In Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 241–244, Philadelphia, Penn., USA, 2005.

[17] L. Oudre, Y. Grenier, and C. Fevotte. „*Template-based chord recognition: Influence of the chord types*". In Proceedings of the 10th International Society for Music Information Retrieval Conference pages 153–158, (ISMIR 2009), 2009.

[18] T.S. Caetano, S.D. Olabarriaga, D.A.C. Barone, „*Performance evaluation of single and multiple-Gaussian models for skin-color modeling*". Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on, SIBGRAPI02, pp. 275–282, 10 Oct. 2002.

[19] Stuart Russel – Peter Norvig, „*Mesterséges Intelligencia Modern megközelítésben*", Panem kiadó – 2005, ISBN: 9789635454112

[20] Balázs János: „*Beat detection and correction for djing applications*", Diplomaterv, BME-MIT, 2013.

[21] Kovásznai Gergely: „*Párbeszédes rendszerek, 3. fejezet: A beszédfelismerés alapjai*", Digitális Tankönyvtár http://www.tankonyvtar.hu/en/tartalom/tamop425/0038_informatika_Kovasznai_Gergely-Parbeszedes_rendszerek/ch03.html#id471232 (2016.10.10.)

[22] C. Harte, M. Sandler, S. Abdallah, and E. Gomez, "*Symbolic representation of musical chords: A proposed syntax for text annotations*". In Proceedings of the International Conference on Music Information Retrieval, the 6th International Conference on Music Information Retrieval, pp. 66–71, London: Queen Mary, University of London, 2005.

[23] Chris Harte adatbázisa: http://labrosa.ee.columbia.edu/projects/chords/ (2016.10.21.)

[24] Lawrence R. Rabiner, „*A tutorial on hidden Markov models and selected applications in speech recognition*". IEEE Acoust., Speech, Signal Processing Mag., pp. 4–16, AT&T Bell Lab., Murray Hill, NJ, USA, Jan. 1986.

[25] Osmalskyj akkordadatbázisa: http://www.montefiore.ulg.ac.be/services/acous/STSI/file/jim2012Chords.zip (2016.10.16.)

[26] Chordify, online akkordfelismerő program: https://chordify.net/ (2016.10.27.)

[27] Impro-Visor: https://www.cs.hmc.edu/~keller/jazz/improvisor/ (2016.12.03.)

[28] R. M. Keller, & D. R. Morrison, „*A grammatical approach to automatic improvisation*". In Proceedings of the Sound and Music Computing Conference, pp. 330–337, Lefkada, Greece, 2007.

[29] MusicXml: https://www.musicxml.com/ (2016.12.05.)

[30] Vajk István: „*Automaták és nyelvek*", Informatika 2 jegyzet, BME, 2010. március.

[31] JUCE hivatalos honlap: https://www.juce.com/ (2016.12.05.)

[32] Martin Robinson: „*Getting started with JUCE*", Packt Publishing Ltd, Birmingham, 2013 Október, ISBN 978-1-78328-331-6

[33] ArrayPlotter tool: https://visualstudiogallery.msdn.microsoft.com/2fde2c3c-5b83-4d2a-a71e-5fdd83ce6b96 (2016.12.05.)

[34] Michigan tech honlapja: „*Physics of notes*”, http://www.phy.mtu.edu/~suits/NoteFreqCalcs.html (2016.12.08.)