Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andrej Oder

Avtomatsko prilagajanje tempa spremljave solistu

diplomsko delo
na interdisciplinarnem univerzitetnem študiju

Ljubljana, 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andrej Oder

Avtomatsko prilagajanje tempa spremljave solistu

diplomsko delo
na interdisciplinarnem univerzitetnem študiju

Mentor: doc. dr. Matija Marolt

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00054/2013
Datum: 01.09.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **ANDREJ ODER**

Naslov: **AVTOMATSKO PRILAGAJANJE TEMPA SPREMLJAVE SOLISTU**

AUTOMATIC ADJUSTMENT OF ACCOMPANIMENT TEMPO TO SOLOIST

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomski nalogi implementirajte sistem za avtomatsko prilagajanje tempa spremljave solistu. Pri tem razvijte algoritem, ki na podlagi avdio signala sledi kje v notnem zapisu se solist nahaja in glede na zaznano lokacijo prilagaja tempo spremljave. Sistem naj deluje v realnem času.

Mentor:

doc. dr. Matija Marolt

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 1

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andrej Oder, z vpisno številko **63070101**, sem avtor diplomskega dela z naslovom:

Avtomatsko prilagajanje tempa spremljave solistu

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matija Marolt,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne: 19. september 2013 Podpis avtorja:

Hvala družini in prijateljem za podporo. Hvala mentorju Matiji za konstruktivno pomoč pri diplomi.

Kazalo

Povzetek

V diplomski nalogi je predstavljen računalniški program, s katerim lahko aktivno zvočno spremljamo pevca. V tem primeru bi računalnik nadomestil različne inštrumente, kar bi solistu omogočilo petje brez prisotnosti inštrumentalistov. Diplomska naloga ima dva cilja. Prvi je ugotoviti, kje v notnem zapisu se pevec med izvajanjem nahaja, in drugi izvesti spremljavo, ki se po notnem zapisu ujema s solistom. Do rezultatov smo prišli s pomočjo skritega Markovksega modela, analize signala, algoritma za iskanje najbolj verjetnega stanja v verigi in mehanizmov za uspešno zvočno spremljavo. S testiranjem programa smo dosegli oba zastavljena cilja naloge, saj je implementiran program uspešno določal pozicijo pevca, prav tako pa je bila uspešno izvedena tudi aktivna spremljava, ki se je po notnem zapisala ujemala s solistom.

Ključne besede: aktivna zvočna spremljava, skriti Markovski model, procesiranje signala

Abstract

This thesis presents a computer programme which allows users to provide vocalists with active musical accompaniment. In this case, the computer would stand in for various instruments, which would allow the soloist to sing without being accompanied by instrumentalists. The thesis has two goals. The first goal is to discover at which point in a given piece the singer is located, and the second goal is to create and play an accompanying musical score which will match the soloist's piece. Results were obtained by using a hidden Markov model, analysing the signal, an algorithm used to search for the most probable state in the chain and mechanisms for ensuring a successful musical accompaniment. By testing this programme both goals were reached, since the programme was able to successfully determine the position of the singer, as well as successfully providing an active accompaniment, the score of which matched the score of the singer.

Key words: active musical accompaniment, hidden Markov model, signal processing

Poglavje 1

Uvod

Glasba je urejeno in oblikovano zaporedje tonov, zvenov in šumov. Po nekaterih definicijah se od nekega naključnega zaporedja not loči po svoji premišljenosti: note so ritmično razporejene in urejene v melodijo, ki je podprta s harmonijami. Zato v glasbi obstaja potreba po uskladitvi vsake posamezne komponente. Pred zborom in orkestrom stoji dirigent, ki kaže tempo skladbe, da se lahko glasbeniki uskladijo in tako glasba doseže željen učinek. Pevec in pianist se na odru sporazumevata z različnimi gestami, ki jih nekateri niti ne opazijo, in se vsem zdijo kot del skladbe, oziroma interpretacije.

Cilj naloge je nadomestiti pianista, oziroma spremljavo solista z računalnikom. Najlažji način za uskladitev računalnika s solistom bi bil, da bi računalnik predvajal spremljavo kot zvočno podlago, solist pa bi imel odgovornost, da med petjem sledi zvočni podlagi. V nadaljevanju bo predstaljeno, kako lahko dosežemo bolj realno spremljavo, kar pomeni, da je računalnik tisti, ki skrbi za uskladitev pevca, in ne obratno. Predpostavimo lahko, da poznamo notni zapis spremljave in solista.

Program je sestavljen iz treh delov. V prvem delu glede na podan notni zapis solista zgradimo Markovsko verigo, ki nekako najbolj intuitivno predstavi notni zapis. Prav tako notni zapis spremljave pretvorimo v ustrezno obliko, s katero lahko kasneje uspešno sledimo pevcu. Drugi del programa se nanaša na prepoznavanje pozicije pevca v notnem zapisu. S pomočjo osnovne frekvence

in energije signala, ki ga zajamemo s pomočjo mikrofona, lahko izračunamo najbolj verjetno noto v notnem zapisu. V tretjem delu pa glede na najbolj verjetno noto, ki smo jo izračunali v drugem delu, predvajamo spremljavo.

Podobno stukturo programa je razvil C. Raphael v [7]. Primer dobrega delovanja njegovega programa lahko najdemo na [8]. Prav tako lahko podobno strukturo z Markovskimi verigami najdemo v [6], kjer je uporabljena nekoliko drugačna struktura Markovske verige, in kjer je za najbolj verjetno pot v Markovski verigi uporabljen modificarn algoritem Viterbi.

Poglavje 2

Opis metod

Skriti Markovski model (HMM) je statistični Markovski model, pri katerem predpostavimo, da je modeliran sistem Markovska veriga s skritimi stanji. Razvil ga je L. E. Baum s sodelavci [1] in zaradi praktičnosti je model postal priljubljen v različnih sistemih za prepoznavanje govora, prepoznavanje pisave [3], notne spremljave, itd. Za dobro razumevanje HMM-a najprej poglejmo delovanje Markovske verige, ki je osnova za ta model.

2.1 Markovska veriga

Markovska veriga je stohastičen proces, pri katerem je prehajanje med končnim številom stanj določeno z verjetnostmi. Bistvena lastnost verige je, da je prehajanje iz nekega stanja v naslednjega odvisno samo od trenutnega stanja in ne od stanj iz prejšnjih korakov.
Predstavljajmo si sistem, ki se v vsakem časovnem koraku nahaja v enem izmed končnih stanj iz množice N različnih stanj $S_1, \ldots, S_N$. Pri vsaki spremembi se stanje časovnega koraka sistema spremeni glede na ustrezne verjetnosti prehodov med stanji. Če označimo s $t = 1, 2, \ldots$ spremenljivko časa in trenutno stanje v času t s q_t, potem velja lastnost:

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots q_0 = S_l) = P(q_t = S_j | q_{t-1} = S_i),$$

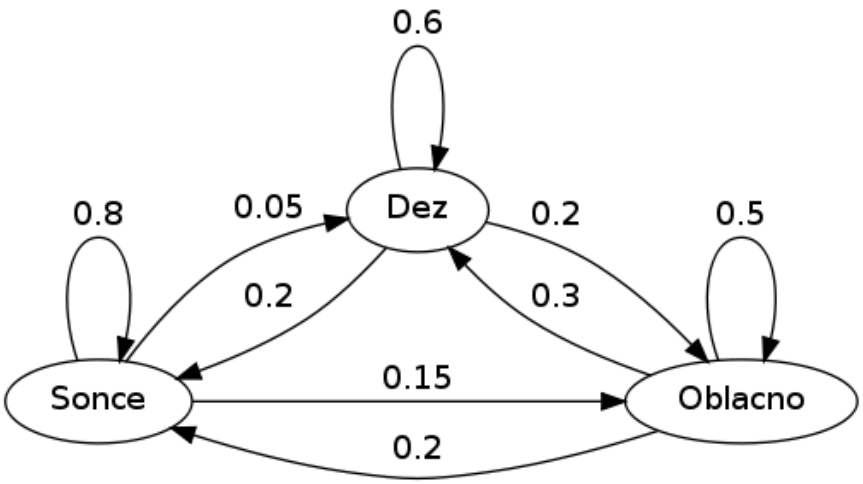
torej je verjetnost za prehod v stanje v naslednjem koraku odvisno samo od prejšnjega stanja. Verjetnost prehoda iz stanja i v stanje j v nekem koraku označimo z a_{ij}. Za te koeficiente morata veljati naslednji pravili:

$$a_{ij} \geq 0 \quad \text{ter} \quad \sum_{j=1}^{N} a_{ij} = 1, \quad \text{za vse } 1 \leq i, j \leq N.$$

Prav tako moramo za vsako stanje definirati verjetnost začetnega stanja, oziroma verjetnosti stanja v prvem koraku imenovanega π_i, ki ima enake lastnosti:

$$\pi_i \geq 0 \quad \text{ter} \quad \sum_{j=1}^{N} \pi_i = 1, \quad \text{za vse } 1 \leq i \leq N.$$

Verjetnosti a_{ij} lahko zapišemo v prehodno matriko A velikosti $N \times N$. Prav tako lahko vrednosti π_i zapišemo v vektor π velikosti N.



Slika 2.1: Stanja in verjetnosti prehodov Markovskega modela vremena [4]

Na sliki 2.1 imamo podan primer Markovskega modela vremena. Označimo stanja modela s $S_1 =$ sončno, $S_2 =$ oblačno in $S_3 =$ deževno. Prehodna matrika tega modela je torej:

$$A = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.2 & 0.5 & 0.3 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

Predpostavimo, da je danes vreme sončno in nas zanima, kakšna je verjetnost, da bo jutri prav tako sončno in dan za tem deževno vreme. Torej je:

$$\begin{aligned} p(q_3 = S_3, q_2 = S_2 | q_1 = S_1) &= p(q_3 = S_3 | q_2 = S_2, q_1 = S_1) p(q_2 = S_2 | q_1 = S_1) \\ &= p(q_3 = S_3 | q_2 = S_2) p(q_2 = S_2 | q_1 = S_1) \\ &= 0.05 * 0.8 \\ &= 0.04 \end{aligned}$$

Drugo vprašanje, ki bi si ga lahko zastavili je, kakšno stanje vremena je v času t najbolj verjetno glede na verjetnost začetnega stanja?

Preprosto lahko dokažemo, da velja rekurzivna enačba:

$$\pi(t) = \pi(t-1)A^T,$$

kjer je vektor $\pi(0)$ začetna porazdelitev in vektor $\pi(t)$ verjetnost stanj v času t.

2.2 Skriti Markovski model

Glavna razlika med Markovskimi verigami in HMM je, da so stanja pri skritem Markovskem modelu skrita. Stanja pri HMM opazujemo preko druge spremenljivke, ki indicira skrito stanje. Vzemimo kot primer modifikacijo Markovega modela vremena s primerom zapornika.

Jetnik je zaprt v sobi brez oken in ga zanima, kakšno je vreme. Ve kakšna je verjetnost, da je vreme sončno, oblačno in deževno. Prav tako pozna verjetnosti sprememb vremena. Vsak dan jetniku prinese hrano paznik, ki z neko verjetnostjo glede na vreme prinese s seboj dežnik. Edini način, kako lahko jetnik predpostavi, kakšno vreme je zunaj, je z opazovanjem.

HMM je torej določen z naslednjimi petimi parametri:

1. končna množica stanj $S = \{S_1, S_2, \ldots, S_N\}$,
2. prehodna matrika $A = \{a_{ij}\}$,

3. začetna porazdelitev stanj $\pi = \{\pi_i\}$,

4. množica možnih opažanj V velikosti M. V našem primeru je množica $V = \{V_1 = \text{dežnik}, V_2 = \text{ni dežnika}\}$,

5. pogojne verjetnosti $B = \{b_j(k)\}$ za vsak element množice opažanj:

$$b_j(k) = P(V_k \text{ v času } t | q_t = S_j) \quad 1 \leq j \leq N, 1 \leq k \leq M,$$

kar predstavlja verjetnost opažanja pri danem stanju.

Sedaj lahko definiramo tri osnovne probleme povezane s HMM:

1. Kako učinkovito izračunamo $P(O|\lambda)$, če imamo podano zaporedje opažanj $O = (o_1, o_2, \ldots, o_T)$ in model $\lambda = (A, B, \pi)$?

2. Kako izberemo parametre za λ, da bomo maksimizirali $P(O|\lambda)$?

3. Kako izračunamo optimalno zaporedje stanj $q = (q_1, q_2, \ldots, q_T)$, če imamo podano O in model λ?

Problem 1. lahko učinkovito rešimo s *Forward-Backward* algoritmom. S pomočjo algoritma *Baum-Welch* lahko učinkovito rešimo problem 2. Za reševanje problema 3. pa lahko uporabimo *Viterbi* algoritem. Delovanje algoritmov je opisano v [5].

Poglavje 3

Opis delovanja

3.1 Opredelitev problema

Cilj naloge je razviti program, s katerim poizkušamo nadomestiti pianista z računalnikom. Glavni namen programa je poenostaviti vadbo solista, saj lahko le tako doseže visoko stopnjo kvalitete. Zaenkrat si ne predstaljamo, da bi na koncertu lahko z računalnikom v celoti nadomestili pianista oziroma spremljavo, zato je glavni namen programa vadba solista. Solist lahko s takšno vadbo dobi občutek približne celote neke skladbe.

Tako kot pianist tudi računalnik pozna note, ki jih mora zaigrati in prav tako pozna note, ki jih bo pevec zapel. Pomembno je, da računalnik spremlja pevca, in ne obratno. Spremljeva naj ne bi nikoli prehitevala solista, saj s tem povzročamo dodaten pritisk na solista, ker bi se moral v tem primeru osredotočiti tudi na spremljavo in ji slediti, kar bi bilo v nasprotju s prej omnjeno predpostavko.

3.2 Opažanja

S pomočjo mikrofona zajamemo zvočni signal, ki ga obdelamo, in iz njega dobimo neka opažanja, ki so med seboj neodvisna. Opažanja so v našem primeru sestavljena iz dveh parametrov.

Prvi parameter je osnovna frekvenca tona. Za določanje osnovne frekvence tona je bila uporabljena knjižnica, ki implementira Yin pitch detector, ki je podrobno opisan v [2]. Velikost okna je bila 1024 s prekrivanjem 238 vzorcev. To pomeni, da smo v vsaki sekundi izračunali okoli 180 frekvenc. V našem primeru je bil uporabljen mikrofon z vzorčenjem 44100 Hz.

Drugi parameter je energija signala, s pomočjo katere lahko ugotovimo, ali je zvok prisoten ali ne. Izračunamo jo lahko po formuli:

$$E(s(n)) = \sum_{i=1}^{n} s(i)^2,$$

kjer je $s(n)$ naš vzorec signala dolžine 238. Če je $E(s(n)) \leq \tau$ pomeni, da na mikrofon ne pada zvok.

Če znamo izračunati frekvenco signala, potem je pri smiselno izbranem τ $E(s(n)) > \tau$, kar pomeni, da na mikrofon prihaja zvočno valovanje. Vseeno pa lahko zaznavamo prisotnost zvoka, čeprav ne moremo izračunati frekvence signala. Takšen primer so sičniki in šumniki.

Seveda bi bilo mogoče uporabiti drug način za določanje razlike med tišino in prisotnim zvokom, vendar je za trenutne potrebe algoritma to dovolj.

3.3 Struktura verige

Notni zapis je sestavljen iz not in pavz, ki imajo predpisano dolžino in predpisano višino. Ponavadi je v notnem zapisu podan tudi tempo, s pomočjo katerega lahko določimo trajanje posamezne note. Intuitivno lahko zapis torej preprosto pretvorimo v Markovsko verigo, kjer nam note in pavze predstavljajo stanja, povezava med sosednjima stanjema pa obstaja, če nota (pavza), ki ga opisuje stanje, sledi drugi noti (pavzi). Potrebno je dodati še povezavo iz vsakega stanja na isto stanje, saj je lahko pevec v času t in času $t + 1$ v istem stanju.

V poglavju 3.2 smo izpostavili tri možne primere za ločevanje opažanj:

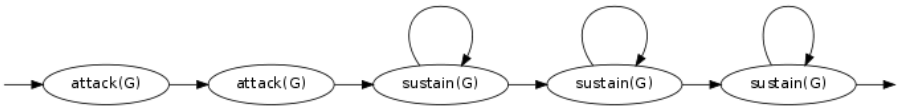
- ton ni prisoten,

- ton je prisoten, vendar mu ne znamo določiti osnovne frekvence,
- ton je prisoten in določimo mu lahko osnovno frekvenco.

Če upoštevamo še dejstvo, da lahko imamo v notnem zapisu note različnih frekvenc, potem so stanja našega HMM:

$$\{\text{rest}, \text{sustain}(f_1), \ldots, \text{sustain}(f_I), \text{attack}(f_1), \ldots, \text{attack}(f_I)\}.$$

Stanje rest predstavlja pavzo, sustain(f_t) predstavlja zadržani del note, oziroma tista opažanja, pri katerih lahko določimo frekvenco. Ta mora ustrezati frekvenci f_t. Attack(f_t) se nanaša na vse začetke not s frekvenco f_t, pri katerih bodisi še nimamo znane intonacije vokala, ali pa na tistem mestu pričakujemo konzonante brez intonacije. Stanje sustain(f_t) se od stanja attack(f_t) razlikuje le v tem, da je v stanju attack(f_t) lahko ton s frekvenco 1 prisoten ali pa ne, medtem ko je za stanje sustain(f_t) ton prisoten. Seveda pri obeh sprejmemo le ton s frekvenco f_t in ne drugih.



Slika 3.1: Veriga note G

Na sliki 3.1 je prikazan model note G, ki se nahaja nekje v notnem zapisu. Treba je določiti še verjetnosti prehodov. V poglavju 3.2 je bila predstavljena pogostost opažanj na sekundo. Iz notnega zapisa lahko predpostavimo trajanje posamezne note. Vzemimo za primer zgornjo sliko ter predpostavimo, da je predvideno trajanje te note ena sekunda. Torej želimo, da izmerimo na tej noti okoli 180 opažanj. Predpostavimo, da so opažanja O slučajne spremenljivke, ki so med seboj neodvisne. Torej velja:

$$\text{Ex}[O] = n/p,$$

kjer je Ex[O] pričakovano število opažanj, n število sustain(G) stanj za neko noto G in p verjetnost prehoda iz sustain(G) v naslednje stanje. Od pričakovanega števila opažanj je potrebno odšteti še opažanji za stanji attack(G) ter tako dobimo $p = \frac{3}{178}$.

Verjetnost prehodov za stanje rest najdemo na podoben način. Tako lahko celoten noten zapis predstavimo z Markovsko verigo.

Na tem mestu je treba izpostaviti tudi razlog za večje število stanj sustain(I) za posamezno noto. V primeru, da si v prihodnosti želimo algoritem izboljšati tako, da se bo program z vsakim izvajanjem izboljševal, bomo potrebovali za neko noto več stanj. Zgodi se lahko, da bodo parametri za vsa ta stanja nekoliko drugačni. Recimo, da se solist odloči in neko noto izvede *fortepiano* čeprav to ni označeno v notnem zapisu. *Fortepiano* predstavlja hitro spremembo jakosti zvoka. To pomeni, da bo vrednost energije zvoka na začetku note višja kot na koncu. Iz tega razloga je lahko verjetnostna porazdelitev za prvo stanje sustain note drugačna kot za zadnje stanje. To pa pripomore k boljšemu sistemu za spremljavo.

3.4 Verjetnostne porazdelitve

Za HMM je značilno, da stanja verige niso direktno vidna. Pri določanju vremena, kjer je jetnik zaprt v sobi, ta pozna verjetnosti, da paznik prinese dežnik ali ne v primeru, da je vreme sončno, oblačno ali dežuje. Zato moramo tudi v našem algoritmu določiti verjetnostne porazdelitve za posamezna stanja. V našem primeru sta naši opažanji intonacija ter energija signala. Energija signala je zvezna funkcija na intervalu večjem od nič, zato je tudi verjetnostna porazdelitev energije zvezna za vsako možno stanje. Vemo, da bo verjetnost stanja rest pri *nizki* vrednosti energije najvišja, zato predpostavimo verjetnostno porazdelitev:

$$P_r(E(s)) = \begin{cases} \frac{1}{\mu_r+\theta_r} e^{\frac{\theta_r - E(s)}{\mu_r}}, & \text{če } E(s) > \theta_r \\ \frac{1}{\mu_r+\theta_r}, & \text{če } 0 \leq E(s) \leq \theta_r, \end{cases} \tag{3.1}$$

kjer je $E(s)$ energija signala s, θ_r in μ_r pa sta povprečna vrednost ter standardna deviacija, ki ju lahko določimo na način opisan v poglavju 3.7.

Stanje attack smo dodali z razlogom, da se kar najhitreje premaknemo v naši verigi naprej, pa čeprav še ne poznamo frekvence tona. Predpostavimo,

da je pri našem signalu na prvi polovici tišina, na drugi polovici pa je prisoten zvok. V tem primeru bo $E(s)$ nižja, kot če bi bila v prvi četrtini tišina, nato isti signal ter v zadnji četrtini dodatno nek zvok. S testiranjem smo ugotovili, da je razlika podobna pri konzonantih, ki nimajo intonacije, ter vokalih. Kadar je v signalu vokal, ki mu znamo določiti osnovno frekvenco, je $E(s)$ tega signala večja. Želimo, da je verjetnost stanja sustain pri *visokih* vrednostih $E(s)$ visoka, pri *srednjih* vrednostih pa je najbolj verjetno stanje attack. Verjetnostno porazdelitev za stanje attack zapišimo kot normalno porazdelitev:

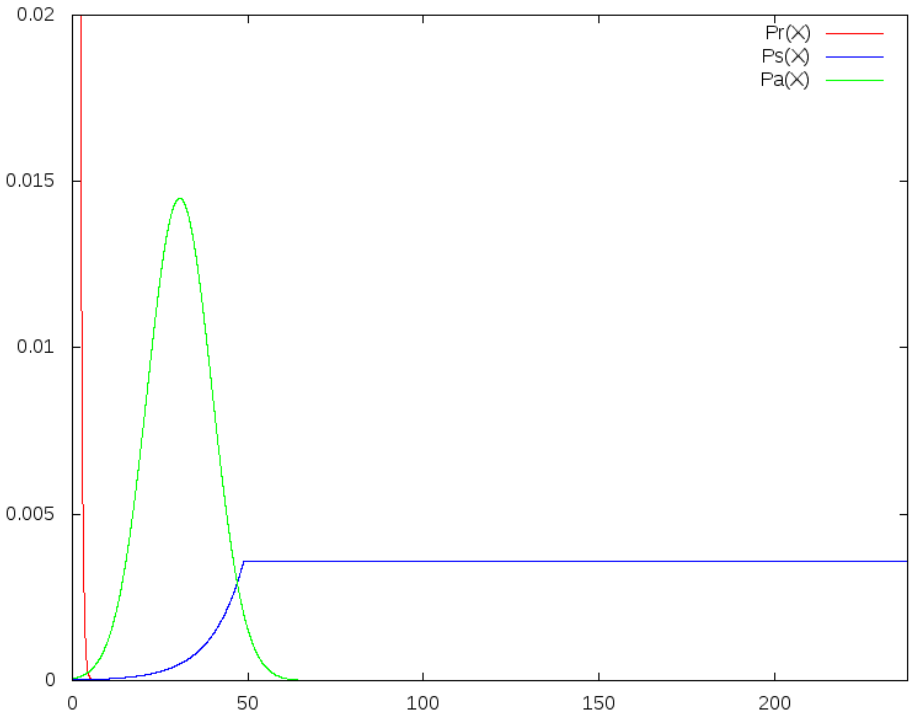
$$\frac{1}{\sigma_a\sqrt{2\pi}}e^{-\frac{(E(s)-\mu_a)^2}{\sigma_a^2}}$$

Verjetnostno porazdelitev za stanje sustain pa definiramo podobno kot za stanje rest:

$$P_s(E(s)) = \begin{cases} \frac{1}{238+\theta_s-\mu_s}e^{\frac{E(s)-\theta_s}{\mu_s}}, & \text{če } E(s) < \theta_s \\ \frac{1}{238+\theta_s-\mu_s}, & \text{če } \theta_s \leq E(s) \leq 238 \end{cases} \tag{3.2}$$

V zadnjem primeru je uporabljena naraščajoča eksponentna funkcija omejena s θ_s. Zaradi tega in zaradi potrebe po normalizaciji verjetnostne porazdelitve je v tem primeru treba omejiti velikost $E(s)$. Ker vemo, da je velikost okna 238, je torej maksimalna vrednost $E(s)$ največ toliko. Tako sicer res dobimo manjše verjetnosti, kar ni najbolje, ampak so te verjetnosti pri visokih vrednostih energije in pri ustrezni izbiri vseh parametrov še vedno večje od verjetnosti, ki jih vrneta ostali dve distribuciji. Posamezne funkcije so seveda odvisne od parametrov. Bolj kot bodo parametri blizu dejanskim vrednostim, bolje bo deloval naš algoritem.

Drugo opažanje, s katerim si zagotovo moramo pomagati pri opazovanju skritih stanj, je osnovna frekvenca tona oziroma intonacija. Vsaka nota v notnem zapisu ima predpisano višino, ki jo lahko izrazimo s frekvenco. Predpostavimo, da po notnem zapisu pričakujemo noto s frekvenco Hz. Želimo sestaviti verjetnostno porazdelitev, ki ima vrh pri frekvenci 440 Hz. Neupravičeno je pričakovati, da bo pevec zapel in vzdrževal ton predpisan čas s točno takšno frekvenco. Pomembno je tudi, da se zavedamo dodatnega pojava pri pevcih,



Slika 3.2: Slika prikazuje realen primer verjetnostnih porazdelitev za vsako stanje

ki ga imenujemo vibrato. Do naravnega vibrata pride zaradi sproščenosti grla ter zaradi konstantnega toka zraka, ki teče skozi glasilke. Tako lahko nastanejo pogoji, pri katerih glasilka niha naprej in nazaj, ter prav tako gor in dol, kar ustvari rahle turbulence v zračnem toku, ko ta zapusti grlo. Nihanje frekvence pri vibratu je približno pol tona s pet do sedem oscilacij na sekundo [9].

Pomembna pomanjklivost je tudi dejstvo, da se naš algoritem za prepoznavo osnovne frekvence tona lahko zmoti za približno oktavo in namesto frekvence 440 Hz porepozna frekvenco 880 Hz.

Program je namenjen pevcem, njihovo poznavanje računalništva ponavadi obsega le osnove. Nekateri imajo velike težave pri prepisovanju not v digitalno obliko. Zaradi tega je bilo predvideno, da bi uporabnik notni zapis prepisal v računalnik samo enkrat in bi te note uporabljal za več glasov. Pogosto se namreč notni zapis za glasova sopran in tenor razlikujeta le v tonaliteti, vse

ostalo pa je enako. Na primer visoki C pri tenorju ima frekvenco okoli 523 Hz, visoki C pri sopranu pa ima frekvenco 1046 Hz, kar je samo dvakratna vrednost frekvence C-ja pri tenorju.

Za verjetnostno porazdelitev frekvence je bila v algoritmu uporabljena vsota treh normalnih porazdelitev. Povprečne vrednosti porazdelitve so bile določene na podlagi pričakovane intonacije in sicer, če pričakujemo ton s frekvenco n, potem je: $\sigma_1 = n/2$, $\sigma_2 = n$ in $\sigma_3 = 2n$. μ_i določimo tako, da je verjetnost tona, ki je pol tona više od tona s frekvenco n, enaka $0,1$. Tako dobimo za stanje sustain:

$$I_s(x) = \frac{1}{3}\sum_{i=1}^{3} \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}.$$

Ker za stanje attack velja, da intonacija ni nujno znana, prištejemo še normalno porazdelitev s parametroma $\sigma_4 = -1$ ter $\mu_4 = 1$. Algoritem za določanje osnovne frekvence namreč vrne vrednost -1, če te ni bilo mogoče izračunati. Tako je verjetnostna porazdelitev enaka:

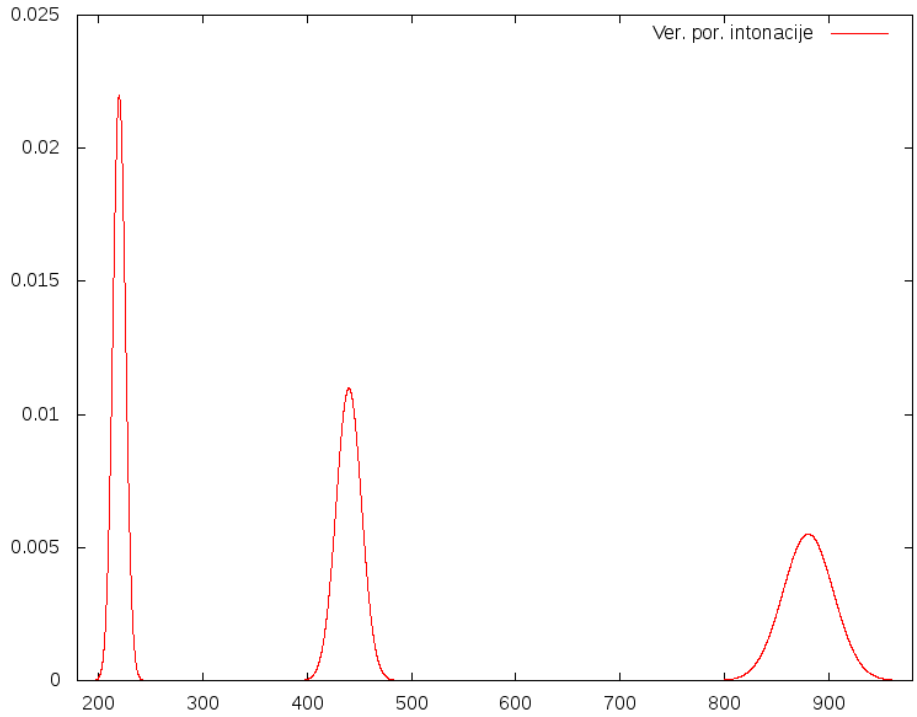
$$I_a(x) = \frac{1}{4}\sum_{i=1}^{4} \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}.$$

Tako imamo za vsako možno stanje definirano:

$$\begin{aligned} b_{\text{rest}}(E(s), x) &= P_r(E(s)) \\ b_{\text{attack}}(E(s), x) &= P_a(E(s))I_a(x) \\ b_{\text{sustain}}(E(s), x) &= P_s(E(s))I_s(x) \end{aligned} \tag{3.3}$$

3.5 Dekodiranje

Do sedaj smo spoznali, kako notni zapis pretvoriti v HMM. Naslednji del algoritma je prepoznavanje pozicije pevca v notnem zapisu v danem trenutku. Torej imamo podano neko zaporedje $O = (o_1, o_2, \ldots, o_N)$ in iščemo stanje x_k, ki je v času N najbolj verjetno. Število stanj v HMM je konstantno, medtem ko se dolžina vektorja O spreminja. Ker bomo morali zgornjo iskanje velikokrat



Slika 3.3: Slika prikazuje verjetnostno porazdelitev frekvence z osnovo 440 Hz

ponoviti je pomembno, da je število operacij majhno, in predvsem da ne računamo istih vrednosti večkrat. Pomagamo si lahko s pomočjo *forward-backward* algoritmom, ki nekaj težav odpravi, vendar ne vseh.

Za dobro spremljavo pevca nas zanima, kdaj je pevec zapel neko določeno noto. Predpostavimo, da je pevec začel skladbo na začetku in sledi notnemu zapisu. Torej vemo, katere note oziroma stanja smo že obiskali. Zanima nas, kdaj bomo zaznali naslednjo noto n, in kdaj se je zgodil začetek te note start(n). To lahko dosežemo z dvema fazama. Označimo z X vsa stanja, ki sledijo stanju start(n). Iščemo k, za katerega velja:

$$P(X_i \geq \text{start}(n) | O_1 = o_1, O_2 = o_2, \ldots, O_k = o_k) > \tau. \tag{3.4}$$

Levo stran neenačbe lahko izračunamo s pomočjo α-verjetnosti. Kadar je neenakost dosežena, lahko predpostavimo, da je začetek note n v preteklosti, kar pomeni, da se je začetek note zgodil nekje med opažanjem 1 in k oziroma,

če imamo znan začetek prejšnje note, potem lahko iskanje omejimo na $\hat{k}$. V naslednjem koraku nas zanima:

$$\arg\max_{\hat{k}\leq k} P(X_j = \text{start}(n)|O_1 = o_1, O_2 = o_1, \ldots, O_k = O_k),$$

kar pomeni, da iščemo $o_{\hat{k}}$, pri katerem je verjetnost za stanje start(n) največje. Zadnje verjetnosti izračunamo s *forward-backward* algoritmom.

α-verjetnosti računamo s pomočjo naslednje rekurzivne formule:

$$\alpha_k(x_i) = p(x_i, o_1, o_2, \ldots, o_k)$$

$$\alpha_{k+1}(x_{i+1}) = \sum_{x_i} \alpha_k(x_i)p(x_{i+1}|x_i)p(o_{k+1}|x_{k+a}).$$

Pri $k = 1$ je $\alpha_k(x_i) = p(x_i, o_1) = p(x_i)p(o_1|x_i)$. Tako zgornjo vrednost k izračunamo:

$$\min_k \alpha_k(X_i \geq \text{start}(n)) = P(X_i \geq \text{start}(n)|O_1 = o_1, \ldots, O_k = o_k)$$

Ker so vse posamezne verjetnosti in α_k manjše od nič, je vrednost $\alpha_{k+1}(x_i) < \alpha_k(x_i)$ za vsak i. Zaradi težav z aritmetiko računalnika zato raje uporabimo $\alpha_k(x_i) = \frac{\alpha_k(x_i)}{\sum_{i=1}^{N} \alpha_k(x_i)}$.

Podobno lahko definiramo:

$$\beta_k(x_i) = p(o_{k+1}, o_{k+2}, \ldots, o_{N-1}, x_i)$$

$$\beta_{k-1}(x_{i-1}) = \sum_{x_i} \beta_k(x_i)p(x_k|x_{k-1})p(o_k|x_i),$$

ki jih normaliziramo enako kot α-verjetnosti. Razlika med α in β verjetnostmi je v tem, da pri α začnemo s prvim opažanjem in nadaljujemo računanje naprej. Pri β verjetnostih pa začnemo pri zadnjem opažanju in nadaljujemo računanje nazaj. Odtod tudi ime *forward-backward* algoritem. Tako lahko izračunamo:

$$P(X_j = \text{start}(n)|O) = \frac{\alpha_k(X_j)\beta_k(X_j)}{\sum_{X_{j'}} \alpha_k(X_{j'})\beta_k(X_{j'})}.$$

α_k-verjetnosti so odvisne od opažanj. Ker se o_1 v času k in času $k+1$ ni spremenilo, je α_1 vedno enaka. Torej lahko z ustrezno podatkovno strukturo

zmanjšamo število računskih operacij tako, da si zapomnimo α_k za vsak k. Težava se pojavi pri β-verjetnostih. Število opažanj se povečuje in zato je potrebno vse verjetnosti računati ponovno. V programu so iz tega razloga bile uporabljene le α-verjetnosti. Izkazalo se je namreč, da računanje β porabi veliko časa. Prehodna matrika vsebuje veliko vrednosti nič. V prihodnosti lahko implementiramo algoritem, ki v takšnih primer zmanjša število množenj le na elemente, ki so različni od nič, ter tako dosežemo učinkovito računanje β-verjetnosti. Te so nujno potrebne za natančno določanje začetkov not.

Število operacij zmanjšamo tudi s številom primerjav, ki jih opravimo z vrednostjo τ. Predpostavljamo, da ko je solist na prvi noti, v naslednjem koraku ne bo zapel zadnje note ampak bo sledil notnemu zapisu. Iz tega razloga so bile uvedene dodatne podatkovne strukture, in sicer:

- vsako stanje kaže na noto, kateri pripada,
- vsaka nota kaže na prvo stanje v verigi HMM, ki pripada tej noti,
- vsaka nota kaže na vsako stanje naslednje note v notnem zapisu,
- vsako stanje, ki pripada neki noti, kaže na vsa naslednja stanja iste note.

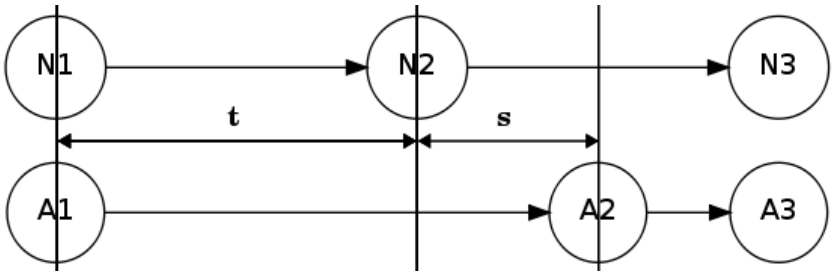
S takšnimi povezavami se lahko v verigi premaknemo za poljubno število not v notnem zapisu naprej in preverjamo le najmanjše število stanj, ki jih želimo. Prav tako lahko najprej preverimo, če je solist zapel naslednjo noto in šele nato preverimo, če je ostal na isti noti in se premaknemo le za kakšno stanje naprej. Ker ne računamo β, moramo poiskati najmanjši k, za katerega velja neenačba 3.4, saj le tako lahko določimo začetek naslednje note in v pravem času zaigramo ustrezno spremljavo.

3.6 Spremljava

Skozi prejšnja poglavja smo spoznali postopek, kako določiti, katero noto poje v danem trenutku solist in kdaj je to noto pričel peti. Prav tako vemo, kdaj je bil začetek vseh prejšnjih not ob predpostavki, da je solist vse note do tega časa

zapel. Naslednja naloga našega programa je, da v danem trenutku računalnik zaigra ustrezno spremljavo.

Naš notni zapis solista ter spremljave smo podali v obliki MIDI datoteke. V teh datotekah so note podane preko dogodkov, ki mu s pomočjo utripov določimo, kdaj naj bo ta nota zaigrana, ali kdaj naj bo note konec. Trajanje note lahko določamo tudi s pomočjo variacije tempa in ostalih funkcij. Ne glede na to lahko začetek, dolžino ter konec note izrazimo v milisekundah. Pomembno je, da če se v času t zgodi nek dogodek v MIDI datoteki solista in bi se po notnem zapisu moral zgoditi nek dogodek v MIDI zapisu spremljave, potem morata biti ta dva dogodka časovno usklajena.



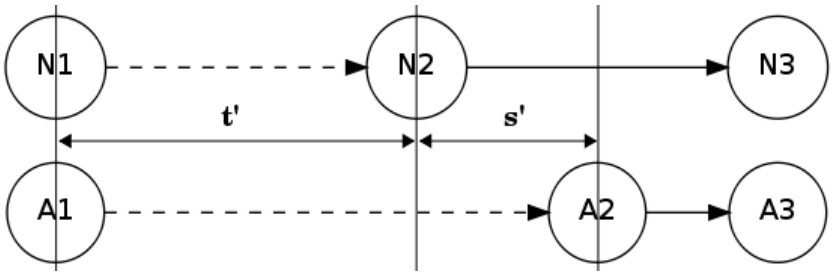
Slika 3.4: Model notnega zapisa solista in spremljave (idealni primer)

Od solista nikakor ne moremo pričakovati, da bo skladbo izvajal v konstantnem tempu, oziroma da bo dolžina not natanko ustrezala predpisani. Še več. Od solista pričakujemo, da bo zaradi interpretacije skladbe delal variacije v tempu. Iz tega razloga je cilj, da se solistu prilagajamo glede na njegov trenutni, tempo in ne glede na tempo neke note.

Na sliki 3.4 imamo podan primer, ki predstavlja notni zapis solista in spremljave. Stanja predstavljajo dogodke v notnem zapisu. Bodisi je to sprememba akorda, bodisi note, ali pavze. Dolžina povezave predstavlja trajanje tega dogodka. Prva veriga predstavlja zapis solista. Druga pa predstavlja zapis spremljave.

Želimo torej, da se v našem spremljevalnem sistemu zgodi dogodek $A1$ v istem času, kot je se zgodil dogodek $N1$, ozirma želimo, da je čas zamika majhen. Za lažje razumevanje si za začetek poglejmo primer, ko sta se zgodila oba dogodka $A1$ in $N1$, zanima pa nas, kdaj se mora zgoditi dogodek $A2$. V

idelanem primeru in pri konstantnem tempu bi se ta moral zgoditi v nekem razmerju med časom trajanja med dogodkoma $N2$ in $N3$ ter med dogodkoma $N2$ in $A3$. Ker v našem primeru nikoli ne bomo poznali trajanja med dogodkoma $N2$ in$N3$, in ker poznamo vse količine, je bolj primerno, da vzmamemo razmerje $\frac{t}{s}$, kjer je t čas med dogodkoma $N1$ in $N2$, ter s čas med dogodkoma $N2$ in $A2$.



Slika 3.5: Model notnega zapisa solista in spremljave (realen primer)

Realno (slika 3.5) pričakujemo, da čas trajanja med $N1$ in $N2$ označen s t' ne bo konstanten in se bo od izvedbe do izvedbe razlikoval. Dogodek $A2$ se mora zgoditi v času s' po dogodku $N2$. Veljati mora enakost:

$$\frac{s}{t} = \frac{s'}{t'},$$

kjer sta s in t konstanti določeni s pomočjo notnega zapisa. Za vsak dogodek, ki je bil v celoti izveden lahko določimo t' in tako izračunamo le s'. Ne glede na to, s kakšnim tempom solist izvaja skladbo, se spremljava prilagaja trenutnemu tempu. Če bo čas t' krajši, potem mora biti za enakost tudi s' krajši. Če je t' daljši, mora biti s' daljši.

Na tem mestu je potrebno rešiti dve težavi in sicer:

- kako rešimo primer usklajenosti, če se morata dogodka solista in spremljave zgoditi v istem času,
- koliko je vrednost t pred prvim dogodkom $N1$?

Odgovor na drugo vprašanje je preprost. Čas t je lahko katerakoli vrednost različna od nič. Pozorni moramo biti le, da tudi t' v tem primeru nastavimo na enako vrednost.

Odgovor na prvo vprašanje je nekoliko bolj zapleten. Vzemimo za primer dogodka $N3$ in $A3$ zgornje slike, ki bo dodatno razložil tudi določanje potrebnih količin. Imamo dva možna načina, kako določiti parametre t, s, in t'. t naj bo čas trajanja od $N1$ in $N2$, s naj bo čas trajanja od $N2$ in $A3$. Tako mora biti ustrezno tudi t' enak času, ki ga solist porabi iz $N1$ in $N2$. Na takšen način bomo izračunali vrednost s', ki predstavlja, koliko časa za dogodkom $N2$ se mora izvesti dogodek $A3$, ne glede na dogodek $N3$. S takšno izbiro bo torej solist sledil spremljavi in ne obratno. To nikakor ni slabo in je lahko v določenih primerih tudi zelo uporabno, če je v skladbi spremljava v danem trenutku pomembnejša kot solist.

V diplomi so bile uporabljene samo količine, ki spremljajo solsita in ne obratno. Ko se zgodi dogodek $N3$ se izračuna količina t', ki je čas od $N2$ do $N3$, s je za stanje $A3$ enak nič, t pa je idealni čas od $N2$ do $N3$.

3.7 Nastavitev parametrov

Med izdelavo programa se je izkazalo, da imajo za določanje pozicije v notnem zapisu največjo težo pravilno izbrani parametri za verjetnostne distribucije. Na energijo signala namreč vplivajo različni faktorji, kot so nastavitve mikrofona, oddaljenost mikrofona in prav tako, s kakšno dinamično stopnjo izvaja solist določeno skladbo. Pričakujemo lahko, da bo solist svojo dinamično stopnjo prilagajal poznavanju skladbe. Vendar v tej iteraciji programa ni predvideno, da bi se parametri med iteracijami ali tekom skladbe kakorkoli spreminjali oziroma prilagajali. V ta namen je bil implementiran dodatek za določanje parametrov verjetnostnih porazdelitev energije za vsako posamezno stanje - umerjanje.

Ker imamo tri možna stanja rest, attack in sustain, naredimo tri umeritve. Vsak traja pet sekund, kjer iz meritve odrežemo prvo in zadnjo sekundo ter naredimo analizo vzorca tako, da na enak način kot v poglavju 3.2 izračunamo energijo za posamezna okna vzorca. Za stanje rest je predviden posnetek tišine oziroma normalnega dihanja solista. Za stanje attack je predvidena črka S,

za katero vemo, da nima intonacije, za stanje sustain pa je predviden vokal *a*. Po analizi se preprosto izračunata standardna deviacija in povprečna vrednost podanega vzorca, ki nam predstavljata parametre za verjetnostno porazdelitev posameznega stanja.

3.8 Opis programa

Program je bil razvit v programskem jeziku *Java*, kjer sta bili dodatno uporabljeni knjižnici za Markovske verige ter za algoritem *Yin*. Notni zapis spremljave in solističnega dela se nahajata v dveh različnih MIDI datotekah. Najprej iz MIDI datoteke solista ustvarimo Markovsko verigo z zgoraj opisano strukturo, prehodo matriko ter verjetnostnimi porazdelitvami. Prav tako je potrebno obdelati datoteko spremljave, kjer določimo vrstni red ter čas začetkov in koncev dogodkov.

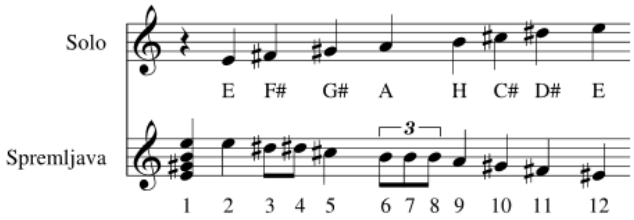
Signal mikrofona moramo najprej ustrezno pretvoriti v vrednosti tipa `float`. Te vrednosti nato uporabimo za analizo, kjer s pomočjo knjižnice *Yin* določimo osnovno frekvenco tona ter izračunamo energijo signala. Širino okna lahko seveda določamo s parametri, ki jih bo mogoče v prihodnosti vnašati s pomočjo grafičnega vmesnika. Iz frekvence in energije signala nato s pomočjo α-verjetnostmi izračunamo verjetnosti vseh stanj. Če je verjetnost katerega koli stanje naslednje note večja od τ, potem si zapomnimo to noto, njeno trajanje ter trajanje prejšnje note. Sledi zadnji del, ki je spremljava. Ta deluje popolnoma neodvisno od detekcije in analize. Pri spremljavi predvajamo MIDI datoteko tako, da se v MIDI datoteki premikamo na ustrezne *tick*-e, pri tem pa ohranjamo tempo zelo majhen.

Poglavje 4

Analiza in izboljšave

4.1 Analiza

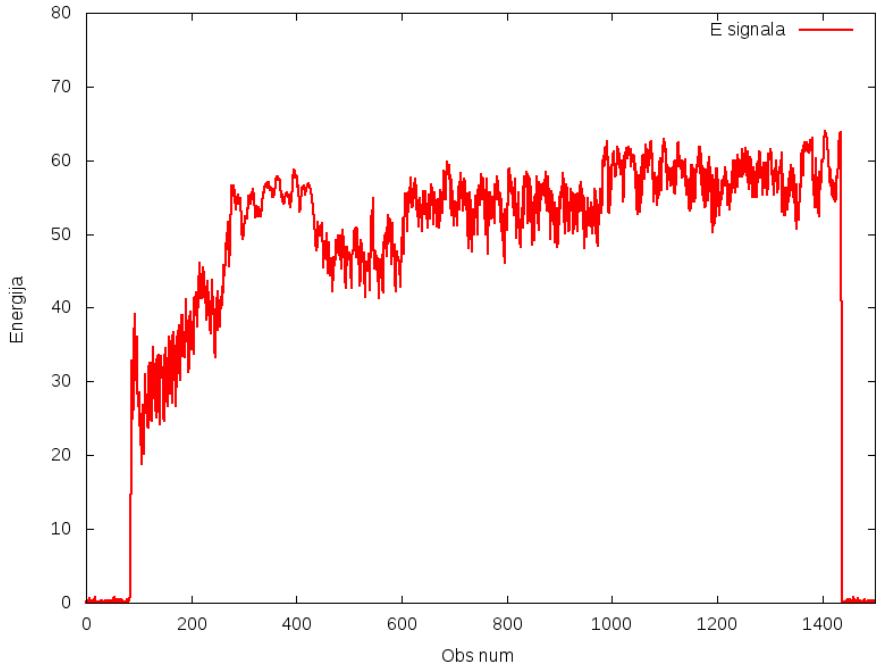
S testiranjem programa sta bila preverjena osnovna cilja diplomske naloge. Program je s pravilnimi nastavitvami parametrov posameznih verjetnostnih distribucij zadovoljivo določal pozicijo pevca. Prav tako je bila izvedena spremljava pevca z vsemi vmesnimi notami. Za testni primer smo vzeli E-dur lestvico s spremljavo prikazano na Sliki 4.1. V notnem zapisu solista je pod vsako noto zapisano ime note. V notnem zapisu spremljave pa je pod vsako noto zapisana zaporedna številka note. Te oznake so uporabljene v vseh spodnjih grafih. Solist je izvajal skladbo brez besedila, oziroma je uporabljal vokal a.



Slika 4.1: Primer spremljave in solističnega dela.

Na Sliki 4.2 lahko vidimo, kakšna je bila energija signala pri posameznem opažanju za podan primer notnega zapisa. Kot lahko vidimo, se največje

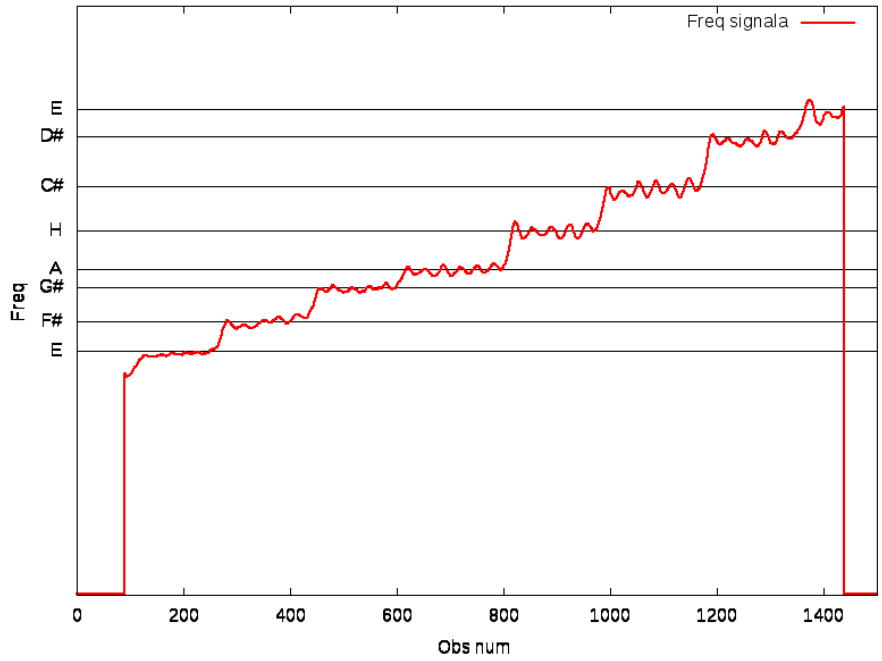
spremembe energije zgodijo na začetku in na koncu, torej takrat, ko pevec začne peti in takrat, ko preneha s petjem.



Slika 4.2: Energija spremljave v odvisnosti od opažanja.

Slika 4.3 prikazuje frekvenco signala pri določenem opažanju. Opazimo lahko, da so skoki med notami zvezni v smislu frekvence, in da je prisoten vibrato, ki ni večji od pol tona.

Na Sliki 4.4 so poleg frekvence z navpičnimi črtami označena tista opažanja, pri katerih je bila izveden začetek note. Vidimo lahko, da spremljava sledi solistu, kljub temu da solist nima konstantnega tempa. Prav tako so bile izvedene vse vmesne note, ki so napisane v spremljavi. Težava se pojavi pri začetkih not 6, 10 in 12. Vidimo lahko, da so bili začetki nekoliko prepozni in bi se morali zgoditi nekje v okolici opažanj označenih z zelenimi navpičnimi črtami. Posledično bi morali biti tudi noti 7 in 8 izvedeni prej, saj je čas med noto G# in noto A krajši. Opazimo lahko tudi, da je bila nota 2 izvedena preden je solist sploh dosegel pravo višino. V tistem trenutku je bila intonacija pevca neznana. Razlog za to je v energiji signala. Pri takratnem opažanju je



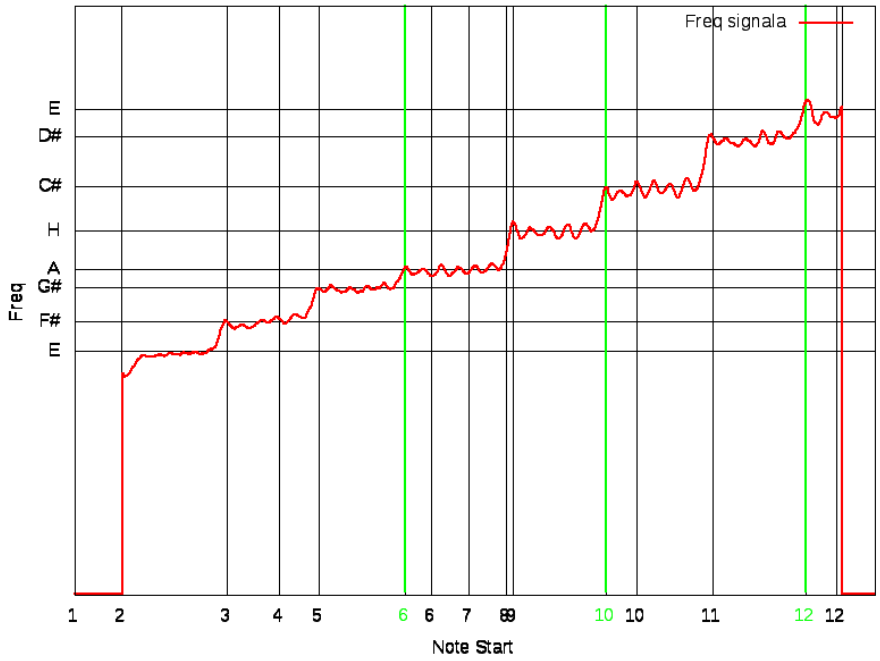
Slika 4.3: Frekvenca signala v odvisnosti od opažanja.

bila energija signala že dovolj velika, da je bilo stanje *attack* najbolj verjetno.

Težave pri testiranju so se pojavile, če je solist naredil napako. V tem primeru ni bilo več mogoče najti pozicije solista in tako se je spremljava ustavila na zadnjem znanem mestu. Enako se zgodi, če pevec naredi vdih na nepredvidenem mestu.

Velik problem predstavlja prehodna matrika Markovske verige. Zaradi velikega števila stanj je računanje β vrednosti zamudno in je bilo zaradi tega izpuščeno. To pa vpliva na določitev začetka note in posledično zmanjša natančnost spremljave. Tukaj govorimo o tistih dogodkih spremljave, ki naj bi se izvedli med dvema dogodkoma solističnega dela. Čeprav so razlike majhne, vse to vpliva na izvajanje solista in posledično se lahko tempo solista nezavedno zmanjšuje, saj se solist kljub temu ozira na spremljavo in ji nezavedno sledi.

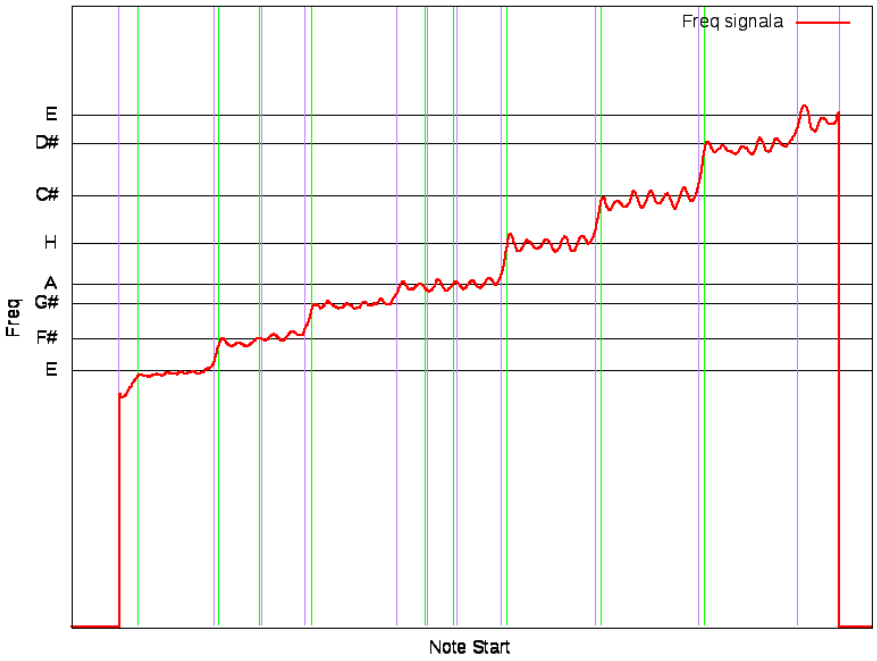
V tej iteraciji programa je bilo potrebno za poslušanje spremljave uporabiti slušalke, v nasprotnem primeru bi mikrofon zajemal tudi zvok, ki ga proizvaja program. Tako bi se lahko zgodilo, da program sledi sam sebi.



Slika 4.4: Frekvenca signala v odvisnosti od opažanja in začetki posameznih not spremljave.

4.2 Izboljšave

Na tem mestu najprej izpostavimo Sliko 4.5. Na sliki je prikazana frekvenca signala. Z zelenimi navpičnimi črtami so označeni začetki not. Začetek not je bil določen v prvem opažanju, pri katerem smo za manj kot polovico stran od razdalje do sosednje note. Z vijolično barvo pa so bili označeni konci teh not. Vidimo lahko, da je razlika v začetku in koncu neke note majhna, razen na začetku, kjer je razlika okoli 37 opažanj, kar nanese okoli 0.2 sekunde. Ker ne opazujemo le frekvence neke note, pač pa tudi energijo, se s tem ni treba posebej ukvarjati. Pomembno je, da lahko v nadaljnih implementacijah upoštevamo začetek naslednje note tudi takrat, ko se neka nota zaključi. Lahko se zgodi, da je pevec imel na tem mestu zelo velik vibrato, ali pa je bila napaka v meritvi, ampak na takšen način lahko izvajamo posamezne dele skladb, kjer ne želimo, da računalnik sledi pevcu, pač pa pevec računalniku.



Slika 4.5: Frekvenca signala v odvisnosti opažanja ter začetki (zeleno) in konci (vijolično) posameznih not.

S pomočjo uporabe Markovske verige lahko dosežemo večjo fleksibilnost za določanje strukture neke note. Če vemo, da bo solist na nekem mestu naredil vdih, potem lahko noto pred vdihom sestavimo tako, da bo imela na koncu dodano stanje *rest*. Še boljša rešitev je, da dodamo dodatna stanja, ki predstavljajo dodatne note. Iz vsake note/pavze se lahko premaknemo bodisi na naslednjo noto/pavzo, ali pa se premaknemo na dodatno noto. Iz te dodatne note se lahko premaknemo na katerokoli naslednjo noto. Na takšen način lahko dosežemo več različnih situacij kot so: napaka, dodajanje not/pavz, ali na primer variacija zapisa. Potrebno je izbrati smiselno število not, v katere se lahko premaknemo iz dodatne note. Prav tako je potrebno definirati, kaj naj se zgodi s spremljavo, če se pojavimo na noti, ki je ni v notnem zapisu. Podrobnejši opis lahko bralec najde v [6]. Prav tako je potrebno dodati več različnih definicij struktur posameznih not, saj notni zapis vsebuje veliko več znakov, kot jih v tem trenutku predvideva naš program.

Za matriko z veliko ničlami bi bilo dobro uporabiti drugačno podajanje matrike. V vsaki vrstici imamo le nekaj vrednosti različnih od nič. Tako si lahko zapomnimo pozicije in vrednosti ter tako omejimo računanje le na elemente, ki so različni od nič. Za uspešno vključitev te rešitve v naš program je potrebno spremeniti celotno knjižnico za Markovske verige ter spremeniti *Forward-Backward* algoritem.

Pričakujemo, da bo vsaka naslednja izvedba boljša, oziroma bo konvergirala proti končni izvedbi, zato bi bilo dobro implementirati učenje Markovske verige. Tako bomo lahko bolj natančno spremljali solista in celotna izvedba skladbe bo boljša.

Pogosto se v notnem zapisu pojavi del, kjer je v ospredju spremljava in solist nima svoje partiture. Takrat ne želimo, da spremljava nadaljuje v tempu, s katerim je zaključil solist pač pa želimo, da spremljava nadaljuje s svojim tempom. Zato je potrebno določiti mesta, kjer je tempo spremljave določen vnaprej in je popolnoma neodvisen od solista.

4.3 Zaključek

Program, implementiran v diplomski nalogi, je za dobro uporabo potrebno še dodelati, saj v trenutnem stanju program uspešno deluje za enostavne primere. Prav tako bi bilo za dobro uporabniško izkušnjo, potebno implementirati grafični vmesnik. Vse to pa v tem trenutku presega obsege, zastavljene v diplomski nalogi, in je prepuščeno nadaljnemu razvoju aplikacije.

Literatura

[1] L. E. Baum et al., "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics,* št. 41, str. 164, 1970.

[2] Alain de Cheveigne, Hideki Kawahara, "YIN, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America,* št. 111, zv. 4, str. 1917–1930, 2002.

[3] Marcus Liwicki and Horst Bunke, "HMM-Based On-Line Recognition of Handwritten Whiteboard Notes," v *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[4] Andres Meng, An Introduction to Markov and Hidden Markov Models, Oktober 2003.

[5] Mikael Nilsson and Marcus Ejnarsson, *Speech Recognition using Hidden Markov Model: magistrsko delo,* Karlskrona, 2002.

[6] Nicola Orio and Francois Déchelle, "Score Following Using Spectral Analysis and Hidden Markov Models," v *Proc. of the International Computer Music Conference (ICMC),* 2001.

[7] Christopher Raphael, "Automatic Segmentation of Acoustic Musical Signals Using Hidden Markov Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* št. 21, str. 360–370, 1998.

[8] Christopher Raphael. (2013). *Examples for ICML 2010, Haifa*[Online]. Dostopno na: http://www.music.informatics.indiana.edu/papers/icml10/.

[9] Johan Sundberg. (avg, 2013). "Acoustic and psychoacoustic aspects of vocal vibrato"[Online]. Dostopno na: http://www.speech.kth.se/prod/publications/files/qpsr/1994/1994_35_2-3_045-068.pdf