Univerzita Karlova v Praze
Matematicko–fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Vladimír Sadloň

Porovnávání hudby

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Bálek
Studijní program: Informatika

2007

Rád by som sa poďakoval všetkým, ktorí mi boli v priebehu písania bakalárskej práce nápomocní a bez ktorých by jej napísanie nebolo možné. Predovšetkým ďakujem svojim rodičom, ktorí mi poskytli potrebné znalosti z oblasti hudobnej teórie a terminológie a samozrejme vedúcemu práce Mgr. Martinovi Bálkovi za pomocnú ruku.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 08.08.2007 Vladimír Sadloň

Obsah

1 Úvod **5**

2 Použité algoritmy **7**
2.1 Výber vhodného typu algoritmov 7
2.2 Naivný algoritmus 8
2.3 Editačná vzdialenosť 9
2.4 Local suffix alignment 12
2.5 Efektivita algoritmov 16

3 Prevod hudobného diela na reťazec **18**
3.1 Logické členenie hudobného celku 18
3.2 Porovnávanie podľa základných stavebných jednotiek 20
3.3 Porovnávanie podľa väčších logických celkov 21
3.4 Metódy výberu reprezentanta 23
3.5 Oktáva a označenie tónov 24
3.6 Vplyv transpozície na porovnávanie 25

4 Ukážka výstupov ohodnocujúcej funkcie a algoritmov **27**

5 Implementácia **31**
5.1 Formát vstupných dát 31
5.2 Použité dátové štruktúry 32
5.2.1 Trieda Tone 32
5.2.2 Trieda Note 33
5.2.3 Trieda Beat 33
5.2.4 Trieda Track 33
5.2.5 Trieda All_Tracks 34
5.2.6 Ďalšie pomocné štruktúry 35
5.3 Návrh GUI a ovládania aplikácie 35

6 Záver **38**

Použitá literatúra **40**
Obsah priloženého CD **41**

Názov práce: Porovnávanie hudby
Autor: Vladimír Sadloň
Katedra (ústav): Katedra aplikovanej matematiky
Vedúci bakalárskej práce: Mgr. Martin Bálek
e-mail vedúceho: balek@kam.mff.cuni.cz

Abstrakt: Práca predkladá spôsob ako z hudobného diela vytvoriť štruktúru ktorá sa ďalej bude dať spracovávať algoritmami slúžiacimi na porovnávanie reťazcov. Pri tvorbe tejto štruktúry práca vychádza z poznatkov z teórie hudby a harmónie. Ďalej sa zameriava na výber vhodného typu algoritmov na samotné porovnávanie a ich podrobný popis. Práca osvetľuje spôsob implementácie jednotlivých častí a obsahuje ukážku vytvorenej štruktúry a výstupy porovnávacích algoritmov spolu s komentárom.

Kľúčové slová: porovnávanie hudby, podobnosť reťazcov, harmonická podobnosť, harmonizácia

Title: Music Comparison
Author: Vladimir Sadlon
Department: Department of Applied Mathematics
Supervisor: Mgr. Martin Balek
Supervisor's e-mail address: balek@kam.mff.cuni.cz

Abstract: The technique how to create a structure from a musical piece, which can be processed by the algorithms on strings, is presented in the work. In order to create such a structure the knowledge of music theory and harmony is needed. The work is also focused on selection of appropriate types of algorithms and their detailed description. The work explains the implementation of constituent elements and also contains the representative sample of created structure and output of the algorithms with detailed commentary.

Keywords: music comparison, similarity of strings, harmonic similarity, harmonization

Kapitola 1

Úvod

Idea porovnávania hudby pomocou reťazcových algoritmov vznikla po náhodnom následnom vypočutí si skladieb Imagine od J. Lennona a The Show Must Go On od skupiny Queen, ktoré majú z harmonického a melodického hľadiska rovnaký ústredný motív, aj keď obe skladby sú značne odlišné. Možnosť vedieť vyhľadávať, respektíve rozpoznávať takéto nie na prvý pohľad viditeľné podobnosti je lákavá najmä z hľadiska hudobnej teórie. Ponúka totiž náhľad na využívanie podobných melodických motívov v rámci jedného časového obdobia, využívanie rovnakých harmonických postupov v rôznych dielach toho istého autora, či podobnosť v rámci rovnakého štýlu a žánru. Zároveň ponúka náhľad na toto všetko v priereze rôznych časových období, rôznych autorov, žánrov a štýlov, kde je práve zaujímavé ako sa rovnaké, ba častokrát až identické motívy objavujú v barokových ako aj rockových, či metalových skladbách.

Práca si kladie za cieľ vytvoriť z hudobného diela štruktúru, ktorá sa bude dať spracovávať algoritmami na porovnávanie reťazcov. Notový zápis hudobného diela totiž svojim spôsobom môžeme pripodobniť napríklad textovým reťazcom – jednotlivými znakmi sú v tomto prípade noty, abeceda je pevne stanovená rozsahom oktávy a jednotlivé časti hudobného diela sú ekvivalentom slov a viet. V skutočnosti to nie je také jednoduché, pretože hudobné dielo je komplexnejšia, nie lineárna záležitosť, kde koexistujú viaceré hlasy a jednotlivé tóny sa navzájom prelínajú a znejú naraz. Ak by sme to mali prirovnať k textovým reťazcom, mohli by sme si predstaviť ako naraz vyjadruje viacero ľudí podobné myšlienky rôznymi slovami a my by sme sa mali snažiť zachytiť zmysel týchto monológov a nejakým spôsobom z nich vytvoriť jeden finálny reťazec. Toto prirovnanie azda nie je celkom presné,

pretože v hudbe máme predsa len obmedzenú abecedu a pomerne silný nástroj, harmóniu, ale dáva nám určitý náhľad na komplexnosť problému. V zásade by sa však v hovorenej reči bolo najjednoduchšie zamerať na jeden z monológov a výsledný reťazec koncipovať výlučne podľa neho. Ako sa ukáže, takýto prístup sa dá použiť aj v hudobných dielach, preto už počiatočná predstava notového zápisu ako reťazca je vo svojej podstate správna a pomerne názorná.

Vytvorená štruktúra by ďalej nemala iba slepo kopírovať základné stavebné jednoty hudobného diela, ale snažiť sa spájať tieto stavebné jednotky do väčších celkov za pomoci poznatkov z teórie harmónie a hudby a až z týchto zložitejších celkov vytvárať finálny reťazec určený na samotné porovnávanie. Takisto použité algoritmy by sa nemali obmedzovať na nájdenie prostej zhody, ale skôr sa pokúšať vyjadriť mieru podobnosti dvoch hudobných reťazcov. Oba tieto prístupy totiž dovoľujú sústrediť sa na melodickú a harmonickú stránku hudby, kde drobné odlišnosti nie sú podstatné pre posluchový dojem zo skladby, ale dôležité je ako pôsobí hudobné dielo ako celok. Nie je preto podstatná presná sekvencia tónov, ale to, v akej sú tieto tóny tónine, akým akordom sú harmonizované a náväznosť sekvencie na ostatné hudobné celky v diele.

Ďalším z cieľov práce bolo vyvinúť software, ktorý bude realizovať uvedené požiadavky a bude schopný jednoduchej a účelnej prezentácie výstupov jednotlivých porovnávaní. Dôležitým aspektom jeho návrhu bola predovšetkým jednoduchosť, aby ani pre bežného užívateľa nebol problém s aplikáciou pracovať a vyhodnocovať výstupy a porovnania. Taktiež bol kladený dôraz na to, aby bol tento software schopný hromadného spracovania súborov, t.j. aby bol schopný porovnať vybraný úsek s množstvom ďalších diel a výsledky tohto porovnania prehľadne prezentovať. Ako vstupné dáta boli vybrané súbory vo formáte MIDI najmä kvôli svojej podobnosti notovému zápisu.

Samozrejmou súčasťou práce je i jej voľná dostupnosť a otvorenosť, preto bude samotný program i zdrojové kódy voľne prístupné na internete tak, aby si mohli prípadní záujemcovia stiahnuť buď program alebo zdrojové súbory a upraviť si ich podľa vlastných potrieb.

Kapitola 2

Použité algoritmy

2.1 Výber vhodného typu algoritmov

Podobnosť dvoch hudobných diel, či úryvkov, je vecou skôr subjektívneho posúdenia ako deterministického prístupu. Vzhľadom k skutočnosti, že hudba je v konečnom dôsledku pre ľudské ucho iba vnem zvuku a ticha v priebehu času, určiť či jeden hudobný úryvok bude dvom rôznym ušiam znieť rovnako nie je možné. Veda o štúdiu subjektívneho ľudského vnímania zvukov, psychoakustika, mimo iné hovorí: „...počutie nie je iba čisto mechanický fenomén propagácie vlnenia, ale takisto aj zmyslová a vnemová udalosť...“ [5], a teda takúto udalosť nie je možné jednoznačne interpretovať: „...je výhodné brať do úvahy nielen mechaniku prostredia, ale aj fakt, že ucho i mozog sú súčasťou ľudského posluchového vnemu“ [5].

Základnou otázkou ktorú si teda pri porovnávaní hudobných reťazcov musíme položiť je, aký typ výsledkov od porovnania očakávame. Chceme presnú zhodu dvoch reťazcov, ktorá je daná jednoznačne a teda sa niečím ako posluchový vnem nemusíme vôbec zaoberať, alebo nám stačí zhoda iba čiastočná, kde budú určité nepresnosti ešte tolerované? Predstavme si dve rôzne melódie líšiace sa od seba len v jednom tóne – nakoľko sa zmení posluchový vnem z týchto dvoch melódií? Koľko tónov môže byť zmenených tak, aby bol posluchový vnem ešte rovnaký? Nakoľko je rôzny posluchový vnem dvoch hlasov jednej skladby?

Pretože sme chceli nájsť odpovede aj na tieto otázky – samozrejme v rámci našich ohodnocovacích funkcií (viď kap. 3) a vzhľadom k obmedzeniam daným algoritmami popísanými nižšie - rozhodli sme sa v práci zamerať najmä na algoritmy, ktoré sú schopné aj čiastočnú zhodu vyhodnotiť ako validný výsledok, a popritom je v nich zabudovaný určitý mechanizmus ohodnotenia skutočnosti, ako veľmi sa teda dva porovnávané úryvky líšia – túto skutočnosť budeme nazývať vzdialenosťou úryvkov. Tým pádom sa môžeme pýtať na presnú zhodu, kedy bude odlišnosť porovnávaných úryvkov nulová, a aj na zhodu čiastočnú, ktorej miera sa bude líšiť podľa skutočnej odlišnosti úryvkov – respektíve podľa odlišnosti ktorú vygenerujú naše ohodnocovacie funkcie.

2.2 Naivný algoritmus

Prvým použitým algoritmom je jednoduchý postup na porovnávanie jednotlivých nôt. V podstate ide o naivný algoritmus, ktorý k celkovej vzdialenosti pripočítava vzdialenosť jednotlivých nôt od seba tak, že porovná po dvojiciach všetky tóny obsiahnuté v oboch notách a túto vzdialenosť ďalej normuje počtom takýchto dvojíc. Vzdialenosť tónov od seba počíta podľa použitej vzdialenostnej tabuľky (viď nižšie). Problém tohto algoritmu však spočíva v nerovnomernom rozdelení nôt v dobách a skladbách, takže časovo rovnaké skladby môžu mať značne rozdielny počet nôt – napríklad dve rovnako dlhé skladby skladajúce sa jedna z polových a druhá zo šesnástinových nôt. Jedným z možných riešení je vynormovať túto vzdialenosť podľa počtu dôb. Problém takéhoto prístupu však spočíva v tom, že nebude odrážať skutočnú vzdialenosť podľa základných logických jednotiek, ale opäť by sme sa viac-menej dostali k počítaniu vzdialenosti podľa väčších logických celkov, v tomto prípade dôb. Keďže pri tomto algoritme vyžadujeme čo najväčšiu mieru detailnosti, je už otázka, či je vhodné normovať vzdialenosť počtom jednotlivých tónov obsiahnutých v notách – predsa len nota (v tónine C dur) ktorá obsahuje tóny c, e, g je od noty ktorá obsahuje jediný tón es vzdialená v podstate trojnásobne, pretože každý z uvedených tónov tvorí s tónom es iný interval. Na druhú stranu je z posluchového hľadiska vznik disharmónie týchto dvoch nôt podmienený všetkými znejúcimi tónmi, bez ohľadu na ich počet je výsledkom disharmonický akord. Takisto je diskutabilné aká je v tónine C dur vzdialenosť noty obsahujúcej tón c od

noty obsahujúcej tóny C, E, G, t.j. toniku tejto tóniny - môže byť braná ako nulová, alebo tiež môže byť opäť trojnásobná, t.j. budeme počítať vzdialenosti po dvojiciach. Ako vhodným riešením sa javí práve normovanie podľa počtu vzniknutých dvojíc, kde na jednej strane eliminujeme negatívny jav zvyšovania vzdialenosti s počtom tónov, na druhej strane zohľadníme vzdialenosti všetkých dvojíc tónov.

Vzhľadom k tomu, že vždy medzi sebou porovnávame iba rovnako dlhé úseky z hľadiska počtu dôb a na samotné porovnanie jednotlivých nôt potrebujeme zhruba konštantný počet operácií, časová zložitosť tohto algoritmu je O(n) kde n je počet dôb. Čo sa týka konštantného počtu operácií, tak to celkom pravda nie je, pretože táto konštanta by teoreticky mohla nadobúdať hodnoty vyššie ako n. Je to preto, že nikde explicitne neobmedzujeme počet nôt v dobe, ale predpokladá sa že nebude väčší ako 32, pričom priemerne máme jednu až štyri doby na notu. Takisto sa nepredpokladá, že počet tónov v jednej note presiahne 4 i keď tento je zhora obmedzený 12. Typicky máme v nosnej melódii cca. 1 – 4 noty na dobu, a každá nota obsahuje iba jeden tón.

2.3 Editačná vzdialenosť

Druhým použitým algoritmom je algoritmus nazývaný „Editačná vzdialenosť“. Ide o postup, ktorý v zásade počíta počet krokov, ktoré sú potrebné k pretvoreniu (editácií, odtiaľ názov) jedného reťazca na druhý. Povolené kroky môžu byť v zásade akékoľvek editačné úpravy (s konštantnou zložitosťou) my sa obmedzíme na vloženie znaku, zmazanie znaku, a nahradenie znaku iným znakom. Takže, pokiaľ máme ako vstup dva reťazce S a R, a dané editačné pravidlá, našou úlohou je nájsť sekvenciu operácií tak aby bola čo najkratšia (resp. aby mala čo najnižšiu cenu) a celkovú cenu tejto sekvencie. Pokúsime sa teraz nahliadnuť, ako nájsť riešenie tohto algoritmického problému. K tomu potrebujeme nasledujúcu definíciu:

Definícia: *Označme si S[1..i] podreťazec reťazca S obsahujúci znaky 1 až i, R[1..i] podreťazec reťazca R obsahujúci znaky 1..j. Potom D(i,j) označuje editačnú vzdialenosť dvoch reťazcov S[1..i] a R[1..j].*

Z uvedenej definície vyplýva, že pokiaľ je teda dĺžka reťazca S n znakov, a dĺžka reťazca R m znakov, potom editačná vzdialenosť týchto dvoch reťazcov je D(n,m). My sa pokúsime vypočítať túto vzdialenosť vyriešením všeobecnejšieho problému, konkrétne spočítaním vzdialenosti D(i,j) pre všetky i a j od 0 do n, respektíve m, za pomoci nástroja nazvaného dynamické programovanie. Najprv sa pokúsime stanoviť si rekurentnú rovnicu a jej okrajové (ukončujúce) podmienky, pre jednoduchosť budeme zatiaľ brať cenu každej editačnej operácie rovnú 1.

Okrajové podmienky sú stanovené ako D(i,0) = i a D(0,j) = j. Sú určite správne, pretože hovoria o tom, že pretvorenie reťazca S[1..i] resp. R[1..j] na prázdny reťazec nás stojí presne i, resp. j operácií – jediný spôsob ako toho docieliť je totiž zmazať všetky znaky daného reťazca. Keď máme stanovené okrajové podmienky, môžeme pristúpiť priamo k stanoveniu rekurentnej rovnice.

Veta: *D(i,j) = min [D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)], kde t(i,j) má hodnotu 0 pokiaľ sa i-tý znak reťazca S rovná j-tému znaku reťazca R, inak má hodnotu 1.*

Podrobný dôkaz nájdete v [1,str. 218], jeho hlavná idea je dokázať že D(i,j) sa nutne musí rovnať jednej z troch častí na pravej strane rovnice. Tieto tri časti reprezentujú totiž vloženie, zmazanie a nahradenie, resp. rovnosť znakov, podľa hodnoty t(i,j). Z predpokladu, že všetky tri sú spočítané korektne a teda označujú editačnú vzdialenosť príslušných podreťazcov vyplýva, že D(i,j) sa musí rovnať minimu z týchto troch, pretože používať môžeme iba tieto tri editačné operácie.

Naivný prístup by v tomto prípade bol použiť danú rekurentnú rovnicu priamo ako rekurzívnu funkciu, bolo by to však časovo veľmi náročné a neefektívne. Miesto toho pomocou dynamického programovania prenesieme časovú záťaž na pamäťovú a výpočet urobíme za pomoci tabuľky, ktorú nainicializujeme okrajovými podmienkami. Inicializácia tabuľky a priebeh výpočtu je znázornený v tabuľkách 1 a 2.

D(i,j)		C	E	A	H	C
	0	1	2	3	4	5
C	1					
G	2					
E	3					
A	4					
H	5					
Fis	6					

D(i,j)		C	E	A	H	C
	0	1	2	3	4	5
C	1	0	1	2	3	4
G	2	1	1	2	3	4
E	3	2	1	2	3	4
A	4	3	2	1	2	3
H	5	4	3	2	1	2
Fis	6	5	4	3	2	2

Tab. 1,2 Inicializácia a priebeh výpočtu editačnej vzdialenosti

Časová zložitosť algoritmu sa odvíja od skutočnosti, že musíme vyplniť všetky polia tabuľky a pri vypĺňaní jedného políčka sa urobí iba konštantné množstvo práce, pretože porovnávame hodnoty v troch rôznych bunkách. Polí tabuľky je dohromady n*m, celková zložitosť algoritmu je teda O(nm).

Vzhľadom k tomu že nie je potrebné zistiť presnú sekvenciu operácií, ktoré viedli k výsledku, ale podstatná je iba vypočítaná editačná vzdialenosť, nie je nutné uchovávať si v pamäti celú editačnú tabuľku. Stačí nám vždy uchovávať si iba riadok, ktorý počítame a riadok jemu predchádzajúci, čím ušetríme podstatnú časť pamäte spotrebovanej algoritmom. Označme S reťazec ktorý je v tabuľke v horizontálnej pozícií a R reťazec ktorý je v tabuľke vo vertikálnej pozícií. Nainicializujeme si dvojrozmerné pole o veľkosti 2*n, (kde n je dĺžka reťazca S) a aktuálny riadok ktorý práve v dvojrozmernom poli vypĺňame dopočítame vždy podľa momentálnej pozície v reťazci R modulo 2 (číslujeme od nuly), t.j ak počítame nultý riadok vypĺňame nultý riadok dvojrozmerného poľa, pri počítaní prvého riadku sa opierame o tieto hodnoty a vypĺňame prvý riadok poľa, a pri počítaní druhého riadku tabuľky v podstate prepisujeme položky nultého riadku a pozeráme sa na výsledky z riadku prvého.

Doteraz sme počítali so zjednodušeným modelom, kde cena každej operácie bola rovná jednej. Jedným zo zovšeobecnení tohto modelu je povoliť rôzne ceny jednotlivých operácií – tzn. ak označíme d cenu pre zmazanie resp. vloženie znaku, r cenu nahradenia znaku iným znakom a e cenu zhody, dostávame okrajové podmienky D(i,0) = i*d a a D (0,j) = j*d a rekurentnú rovnicu:

Veta: *D(i,j) = min [D(i-1,j)+d, D(i,j-1)+d, D(i-1,j-1) + t(i,j)], kde t(i,j) má hodnotu e pokiaľ sa i-tý znak reťazca S rovná j-tému znaku reťazca R, inak má hodnotu r.*

Dôkaz vety opäť viď [1].

Musíme ale dávať pozor na to, aby cena zmazania a pridania znaku bola vždy väčšia ako cena nahradenia znaku, pretože nahradenie sa dá ľahko zostrojiť ako zmazanie a pridanie.

Ďalším dôležitým zovšeobecnením je, aby váha substitúcie závisela presne na dvojici znakov ktoré nahradzujeme. To znamená, že by váha mala závisieť od toho, či zamieňame C za G alebo za Fis. Ak sa pohybujeme v tónine C dur, a chceme substituovať jej základný tón (t.j. C), znie menej rušivo ak ho substituujeme za tón ktorý povedzme patrí to toniky (základného akordu) tejto stupnice (E, G) alebo za tón ktorý do tejto stupnice ani nepatrí a vytvára značnú disharmóniu. Preto je v programe zaradená možnosť vybrať si distančnú tabuľku, predpripravená je okrem základnej tabuľky kde je každá substitúcia ohodnotená rovnako aj distančná tabuľka pre skladby v durovej tónine, a distančná tabuľka pre skladby v molovej tónine. Ďalej má užívateľ možnosť pripraviť si vlastnú distančnú tabuľku, v ktorej si zadefinuje ceny jednotlivých substitúcií podľa vlastného uváženia. Tým, že má naša abeceda iba trinásť prvkov (dvanásť jednotlivých poltónov a pomlčku) tak príprava tejto tabuľky nie je náročná, a takisto nie je problém udržiavať si túto tabuľku v pamäti. Tieto distančné tabuľky navyše môžeme používať aj v predchádzajúcom, naivnom algoritme na porovnávanie nôt.

2.4 Local suffix alignment

Ďalším použitým algoritmom je algoritmus, ktorý rieši tzv. „local alignment“ problém – najvhodnejší preklad by bol asi problém lokálneho zarovnania. K pochopeniu tohto problému si musíme najprv zadefinovať pojem (globálne) zarovnanie, a cena globálneho zarovnania. Znenia viet a definícií nasledujúcich v tejto kapitole sú prevzaté z [1] a mierne upravené tak aby aj bez kontextu bol jasný ich zmysel.

Definícia: *(Globálne) zarovnanie dvoch reťazcov S a R získame najskôr vložením medzier do, alebo na konce týchto reťazcov a potom umiestnením týchto reťazcov nad seba tak, že každý znak v každom reťazci sa nachádza oproti práve jednému znaku druhého reťazca.*

Definícia: *Pre dané zarovnanie A reťazcov S a R, nech S´ a R´ označujú reťazce po (nejakom) vybranom vložení medzier tak, aby sa ich dĺžky rovnali. Nech l označuje dĺžku týchto reťazcov. Potom cena zarovnania A je definovaná ako* $\sum_{i=1}^{l} s(S'(i), R'(i))$, *kde s(a,b) je cena zarovnania znakov a,b oproti sebe, a X(k) označuje k-tý znak reťazca X.*

Zarovnanie i jeho cenu si ukážme na príklade. Majme ohodnocovaciu tabuľku (tabuľka 3):

S	c	d	e	f	-
c	2	-1	-1	-2	0
d	-1	1	-2	-1	-1
e	-1	-2	1	-1	-1
f	-1	-1	-1	1	-1
-	0	-1	-1	-1	0

Tab. 3 Ohodnocovacia tabuľka

Potom zarovnanie: c – d c e - f f
c c f - e d c f

má celkovú cenu 2 + 0 – 1 + 0 + 1 -1 -1 + 1 = 1. Všimnime si, že na rozdiel od editačnej vzdialenosti sa tu nesnažíme dosiahnutú cenu minimalizovať, ale naopak maximalizovať. Ohodnocovaciu tabuľku musíme nastaviť tak, aby každá zhoda mala cenu vyššiu, nanajvýš rovnú nule, a každá zámena, respektíve medzera, by mala byť ohodnotená záporne. Tým pádom hľadáme zarovnanie ktoré má čo najvyššiu cenu a maximalizujeme ním počet zhôd.

Konečne sa teda dostávame k lokálnemu zarovnaniu:

Definícia: *Majme reťazce S a R, problém lokálneho zarovnania je nájsť podreťazec T reťazca S a podreťazec U reťazca R tak, aby globálne zarovnanie týchto dvoch podreťazcov malo najväčšiu hodnotu zo všetkých párov podreťazcov z S a R.*

Tento problém budeme riešiť za pomoci iného problému, ktorý sa na tento náš problém dá previesť – tzv. local suffix alignment problem. Predpokladajme, že cena zarovnania dvoch prázdnych reťazcov je nulová. Potom:

Definícia: *Buďte R,S reťazce, potom V(i,j) je definovaná ako cena optimálneho zarovnania predpôn S[1..i] a R[1..i].*

Definícia: *Buďte n,m dĺžky reťazcov S, R, a indexy i≤n a j≤m, potom local suffix alignment problem je problém nájdenia (prípadne prázdnej) prípony A podreťazca S[1..i] a (prípadne prázdnej) prípony B podreťazca R[1..j] tak, že V(A,B) je maximálna zo všetkých párov možných prípon. Túto cenu budeme označovať v(i,j).*

Veta: *Ak a,b je pár indexov ktorý maximalizuje v(a,b) vzhľadom k všetkým dvojiciam indexov i,j, tento pár rieši local suffix alignment problem. Navyše, tento pár rieši aj všeobecný problém lokálneho zarovnania.*

Dôkaz vety viď [1].

Teraz stanovíme, tak ako v prípade algoritmu riešiaceho problém editačnej vzdialenosti, rekurentnú rovnicu a okrajové podmienky tak, aby sme ich potom mohli využiť vo výpočte pomocou dynamického programovania. Okrajové podmienky sú určite v(i,0) = 0 a v (0,j) = 0, pretože si vždy môžeme vybrať prázdnu príponu. Ďalej:

Veta: *Majme reťazce S a R. Pre j > 0 a i > 0 je rekurentná rovnica pre v(i,j):*
v(i,j) = max [0, v(i-1,j-1) + s(S(i),R(j)), v(i-1,j) + s(S(i),_), v(i,j-1) + s(_,R(j)].

Dôkaz vety viď [1].

Algoritmus v tomto prípade postupuje tak, že zaplní celú tabuľku hodnotami maximálneho zarovnania pre indexy i, j. Spolu s touto hodnotou si ešte musíme ukladať informáciu, z ktorej bunky sa algoritmus dostal do aktuálnej, kvôli neskoršiemu spätnému vyhľadávaniu. Do aktuálnej bunky sa algoritmus mohol dostať jedným z troch smerov podľa rekurentnej rovnice, alebo mohlo byť maximum nulové, s tým že všetky tri ostatné hodnoty sú záporné – v tom prípade si zaznačíme že daná bunka nemá predchodcu. Ak sa maximum rovná nule, ale niektorá z týchto troch hodnôt záporná nie je, označíme ako predchodcu práve ju – je to preto že logicky vyhľadávame čo najdlhšie podobné úseky. Takisto nemajú predchodcu ani bunky, ktoré boli vyplnené okrajovými podmienkami. Vzhľadom k tomu, že tu neexistuje žiadna finálna vzdialenosť ako v prípade editačnej vzdialenosti kde sa stačilo pozrieť na poslednú bunku tabuľky, ale najlepším výsledkom môže byť vzdialenosť v ktorejkoľvek bunke, musíme prezrieť hodnoty vo všetkých takýchto bunkách. Užívateľ teda dostane možnosť obmedziť túto vzdialenosť podľa vlastného uváženia. Po vyplnení tabuľky hodnotami algoritmus prechádza všetky políčka tabuľky a ak narazí na hodnotu ktorá je vyššia ako obmedzenie, urobí spätný výpočet, ktorý končí v bunke, ktorá nemá žiadneho predchodcu. Tým pádom dostaneme dve dvojice indexov označujúce začiatok a koniec podreťazcov porovnávaných reťazcov, kde má lokálne zarovnanie hodnotu vyššiu ako obmedzenie.

Časová zložitosť celého algoritmu je n*m krokov na vyplnenie tabuľky a n+m krokov pri spätnom vyhľadávaní, celková časová zložitosť je teda O(nm).

Pri tomto algoritme musíme veľmi dbať na čo najlepšiu ohodnocovaciu tabuľku, pretože práve ona do veľkej miery ovplyvňuje výsledky ktoré dostaneme ako výstup. V programe sú implementované dve ohodnocovacie tabuľky – prvá, ktorá každú zhodu ohodnotí jednotkou a každú zámenu alebo medzeru nulou, dáva ako výstup najdlhšiu spoločnú podsekvenciu. Jej využitie vidíme najmä v porovnávaní skladieb, o ktorých podobnosti dopredu vieme a zaujíma nás podrobnejšie, respektíve v porovnávaní približne rovnako dlhých skladieb, kde bude mať dostatočne dlhá spoločná sekvencia už relevantnú výpovednú hodnotu z hľadiska podobnosti. Samozrejme je možné počítať aj s tým že budeme porovnávať

kratšiu skladbu s dlhšou a hľadať najpodobnejšie úseky, tu si okrem výsledného skóre musíme taktiež dávať pozor na dĺžku nájdeného úseku, pretože nájdenie oveľa dlhšieho úseku nebude mať veľkú výpovednú hodnotu, keďže sa jedná o subsekvenciu. Druhá implementovaná ohodnocovacia tabuľka vracia ako výsledok najdlhší spoločný podreťazec, kde opäť každá zhoda má cenu jedna a zámeny s medzerami majú veľké záporné hodnoty. Táto tabuľka sa hodí najmä na vyhľadávanie presných zhôd krátkych úryvkov. Samozrejme, opäť je tu možnosť nadefinovať si vlastné užívateľské tabuľky tak, aby boli výsledky ešte zmysluplnejšie – opäť je tu možnosť vážených tabuliek v závislosti na jednotlivých znakoch, alebo celkom iného prístupu. Je ale potrebné dať pozor na to, aby mala celá tabuľka záporný súčet všetkých buniek – v opačnom prípade totiž namiesto lokálneho zarovnania dostávame globálne.

2.5 Efektivita algoritmov

Vzhľadom k tomu, že k primárnym cieľom programu patrí aj to, aby bol schopný bežať aj dávkovo a hľadať podobnosť i porovnávať vybranú skladbu s väčším množstvom iných skladieb, je potrebné sa zamyslieť aj nad časovou zložitosťou algoritmov v takomto prípade.

Algoritmus na zistenie editačnej vzdialenosti sme sa preto rozhodli pri dávkovom spracovaní spúšťať iba na rovnako dlhých úsekoch. Ak by sme totiž chceli porovnať vybraný úsek so všetkými možnými úsekmi jednej stopy, dostali by sme sa k exponenciálnej zložitosti, čo nie je žiaduce. V prípade, že porovnávame iba rovnako dlhé úseky, tak ak dĺžku porovnávaného reťazca označíme n, a dĺžku reťazca v ktorom vyhľadávame m, ich počet je m-n. Ak si ďalej uvedomíme, že vo výpočte zložitosti samotnej editačnej vzdialenosti máme dĺžku úsekov rovnakú, celková časová zložitosť bude $O((m-n)n^2)$. Tak isto tomu je aj u naivného algoritmu na porovnávanie nôt, kde rovnako dlhé úseky porovnávame explicitne. Časová zložitosť v tomto prípade je O((m-n)n). U algoritmu na lokálne zarovnanie sa časová zložitosť nezmení, pretože porovnávame vždy celé úseky, takže máme stále O(nm).

Vzhľadom k dávkovému spracovaniu sa taktiež javí ako veľmi rozumná možnosť nejakým spôsobom obmedziť množstvo výsledkov, ktoré užívateľ dostane ako výstup. Pri editačnej vzdialenosti je najlepšou možnosťou obmedziť práve túto vzdialenosť maximálnou hodnotou, ktorú môže dosiahnuť, podobne u naivného algoritmu. U algoritmu na lokálne zarovnanie sa zasa obmedzuje zdola, t.j. hľadá sa podsekvencia, alebo podreťazec dlhý minimálne x znakov. Vo všetkých prípadoch sa tiež musí rátať s použitými ohodnocovacími tabuľkami. Od užívateľa sa pri zadávaní obmedzenia vyžaduje znalosť týchto tabuliek tak, aby vedel posúdiť čo je preňho ešte relevantný výsledok a podľa neho obmedzenie zadal.

Kapitola 3

Prevod hudobného diela na reťazec

3.1 Logické členenie hudobného celku

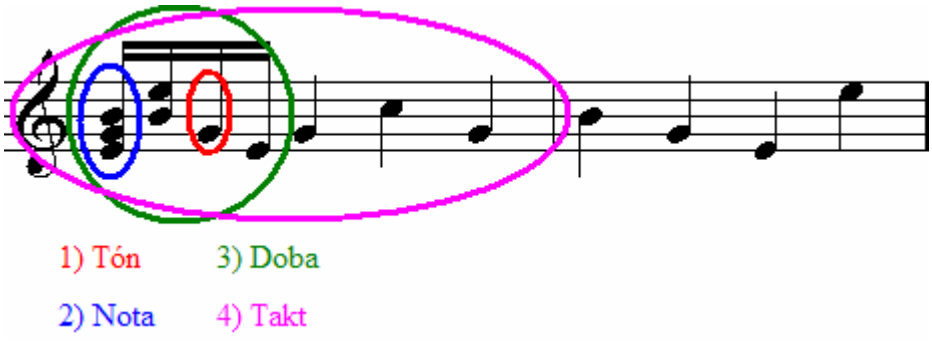
Hudbu samu osebe, a najmä jej notový zápis je možné chápať ako určitú formu reťazca – základnými elementmi sú v tomto prípade jednotlivé tóny noty, ktoré sa spájajú do väčších celkov, dôb či taktov, z ktorých je nakoniec formovaná celá skladba – analógia s akýmkoľvek iným reťazcom, napríklad písomným prejavom, t.j. písmenami a slovami, vetami a literárnym dielom je úplne zrejmá. Určitou odlišnosťou sa môže javiť akási nelinearita, kedy síce niekoľko tónov tvorí jednu notu, ale nota je v hudobnom reťazci chápaná ako nedeliteľná jednotka, a teda tóny tvoria notu paralelne – tzn. v podaní ľubovoľného hudobného nástroja takáto nota znie ako viachlas. Takisto skladby pre viac nástrojov (resp. pre hudobné nástroje schopné hrať viachlasne) sú tvorené jednotlivými hlasmi pre každý konkrétny nástroj, a všetky tieto potom odznievajú naraz, takže ich je možné opäť veľmi ťažko linearizovať. Pre názornosť uvádzame prehľad jednotlivých logických celkov. Grafické znázornenie celkov v notovom zápise je uvedené na obrázku 1.

1) Tón – najzákladnejšia jednotka , každý tón má priradenú samostatnú frekvenciu. Tón je reprezentovaný triedou Tone.
2) Nota – Jedna nota môže obsahovať viac (paralelných) tónov. Je časovo nedeliteľnou jednotkou, i keď jednotlivé noty môžu mať rôznu dĺžku. Základnou jednotkou je nota štvrťová, najväčšou je nota celá, ktorá má dobu trvania rovnakú ako štyri štvrťové noty. Najmenšia nie je explicitne určená, i keď bežne sa nepoužívajú noty kratšie ako stodvadsaťosminové –

v modernej hudbe však už boli vytvorené aj skladby ktoré experimentovali aj s kratšími notami. Je reprezentovaná triedou Note.

3) Doba – jedna doba je základnou časovou jednotkou hudobného diela – typicky pod dobou rozumieme čas trvania doby uvedenej v čitateli taktového označenia, alebo jednoducho čas trvania jednej štvrťovej noty, nezávisle na taktovom označení. Špecifikácia MIDI sa prikláňa k druhému z týchto spôsobov, (viď [3,55]) a tento spôsob je teda využitý aj v tejto práci. Je reprezentovaná triedou Beat.
4) Takt – jeden takt v sebe združuje viacero dôb, ich počet presne závisí od taktového označenia. Taktové označenie je zlomok, ktorého menovateľ označuje typ noty (napr. 4 – štvrťová, 8 – osminová) a čitateľ počet nôt daného typu v takte. Pre takt v práci nie je samostatná dátová štruktúra, ale pracuje sa s ním ako s logickým celkom.
5) Hlas (stopa) – je časť skladby pre jeden konkrétny nástroj (alebo hlas viachlasného nástroja) – jej ekvivalentom je jedna notová osnova v partitúre. Združuje jednotlivé doby (a takty), je reprezentovaná triedou Track.
6) Skladba – celá skladba, v ktorej sú obsiahnuté hlasy všetkých nástrojov. Je reprezentovaná triedou All_Tracks.

Možno trochu mätúce je označenie „tón“ a „nota“, keďže tón je označenie počuteľnej frekvencie, a nota je v podstate značka pre tón v partitúre, zápis daného tónu. Z nedostatku vhodnej terminológie sme sa však priklonili k tomu, že ako „tón“ označujeme notu samostatnú, a termín nota používame pre značku zloženú z jedného alebo viacerých tónov. Dokonca pomlčku, pauzu, chápeme v tomto kontexte ako tón.



Obr. 1 Grafické znázornenie logických celkov

3.2 Porovnávanie podľa základných stavebných jednotiek

Základnou otázkou, ktorú si musíme pri porovnávaní hudby pomocou reťazcových algoritmov položiť je, akým spôsobom budeme konštruovať finálny reťazec, ktorý chceme porovnávať. Ak vezmeme do úvahy jednoduchú, krátku, jednohlasnú melódiu, nemusíme ju na reťazec nijako prevádzať – je totiž reťazcom sama osebe. Abecedu tvoria jednotlivé tóny s pomlčkou (ekvivalent medzery v texte), každú notu tvorí práve jeden tón a tieto tóny sú uložené lineárne za sebou, čo nám dáva dokonalý reťazec. V skutočnosti sú však hudobné diela omnoho zložitejšie. Problém, s ktorým sa je potrebné vysporiadať na prvom mieste, je paralelizmus existujúci v hudobných dielach, t. j skutočnosť, že jednotlivé tóny nie sú v týchto dielach uložené lineárne, za sebou, ale paralelne, kde znie viacero tónov naraz na rôznych úrovniach. Dva najdôležitejšie paralelizmy sú paralelizmus tónov v notách, kde jedna nota môže obsahovať viacero tónov, a paralelizmus jednotlivých hlasov v skladbách, kde sa napríklad orchestrálna skladba môže skladať z teoreticky neobmedzeného počtu hlasov pre každý nástroj. Druhý menovaný problém vyriešime tak, že namiesto celých skladieb sa budeme snažiť porovnávať jednotlivé hlasy – vzhľadom k tomu, že typicky býva v skladbe jeden hlas ktorý nesie hlavnú melódiu (tzv. nosný) tak pri porovnávaní jednotlivých skladieb ako celkov bude najlepšie vyberať tento hlas. V harmonických skladbách sa navyše harmonizuje práve podľa nosného hlasu a všetky ostatné sú vedené v rovnakej harmónií, takže z tohto hľadiska rozdelením do jednotlivých stôp nestrácame vlastne žiadnu pre nás relevantnú informáciu. V skladbách disharmonických toto nutne platiť nemusí, ale keďže v týchto sú jednotlivé hlasy vedené na sebe nezávisle, respektíve tak aby práve dosahovali disharmóniu, bolo by tieto práve lepšie porovnávať po jednotlivých stopách.

Podľa nášho názoru by pokusy o nejaké zloženie jednotlivých hlasov do jedného reťazca viedli k zbytočnej väčšej zložitosti problému a neboli by z hľadiska porovnávania efektívne. Viedlo by to totiž k nutnosti vymýšľať ohodnocovaciu funkciu pre reťazec zložený z jednotlivých hlasov a vzhľadom k tomu, že v konečnom dôsledku by výstup takejto ohodnocovacej funkcie mal byť totožný, resp. maximálne podobný s výstupom ohodnocovacej funkcie pre nosný hlas, bolo

by takéto spájanie neefektívne. Tým, že budeme porovnávať iba jednu stopu sa dostávame k problému, akým spôsobom budeme túto stopu vyberať. V ideálnom prípade by sme mali vybrať stopu ktorej obsahom je nosný hlas skladby, ale štandard MIDI bohužiaľ nedefinuje, ktorý z jeho kanálov obsahuje nosný hlas, a teda je tento výber najlepšie nechať na užívateľovi. Tým sa mu poskytne aj väčšia voľnosť v prípade, že by chcel porovnávať iný ako nosný hlas skladby.

Paralelizmus tónov v notách sa v podstate dá vyriešiť dvomi spôsobmi. Prvý z nich je spočítanie vzdialeností jednotlivých tónov dvoch nôt po dvojiciach a ich následné vynormovanie podľa počtu takýchto dvojíc. Druhý možný spôsob je jednoducho podľa nejakého kritéria vybrať vhodný tón, ktorý bude túto notu reprezentovať. Oba spôsoby majú svoje výhody – prvý umožňuje detailnejší pohľad na skutočnú vzdialenosť daných dvoch nôt, druhý skutočne linearizuje reťazec a samotné porovnanie je výpočetne jednoduchšie. V práci je na porovnávanie reťazcov zložených z nôt využitý prvý z predchádzajúcich spôsobov. Je to najmä preto, že pokiaľ porovnávame samotné noty, chceme už pravdepodobne docieliť čo najväčšej detailnosti a pri výbere jedného reprezentanta spomedzi všetkých tónov obsiahnutých v note už strácame nejaké informácie, ktoré by mohli byť relevantné. Ak totiž vyberieme z napríklad z trojice tónov jedného reprezentanta, informácia o ďalších tónoch sa pre nás stane nedostupná. Druhý spôsob je využívaný v prípade, že nechceme porovnávať priamo noty, ale väčšie logické celky, kde už nám toľko nezáleží na detailnosti, ale práve na harmonickej podobnosti a efektívnosti.

3.3 Porovnávanie podľa väčších logických celkov

Skúsme sa teda zamyslieť, čo by sa stalo ak by sme neporovnávali reťazce zložené z nôt, ale z väčších logických celkov. Ak by sme sa pozreli na analógiu s písomným prejavom zo začiatku tejto kapitoly, tak si môžeme predstaviť že namiesto jednotlivých písmen budeme vnímať zmysel slov a viet, nebude teda dôležitá absolútna podobnosť jednotlivých hlások ale skôr celkový význam slov. Podobne je tomu aj v hudbe, kde namiesto každého jedného tónu obsiahnutého v dobe alebo takte sa budeme skôr snažiť vnímať celkovú harmóniu a myšlienku skladby, nepôjde

teda o porovnávanie tón po tóne, ale budeme sa skôr snažiť zachytiť harmonickú a melodickú podobnosť.

Predstavme si, že by sme mali zo všetkých nôt (respektíve tónov) obsiahnutých v jednej dobe vybrať práve jedného reprezentanta. Tým by nám opäť vznikol reťazec, pripravený na porovnávanie. Podobný proces sa v hudobnej náuke nazýva harmonizácia. Ide o tvorenie a rozvádzanie akordického napätia, čo v širšom slova zmysle môžeme chápať ako výber jedného vhodného akordu, ktorý sa ako reprezentant priradí danej dobe. Harmonizácia v hudbe však neprebieha vzhľadom k časovému celku, ale k celku melodickému – t.j. akordami podporujeme a rozvádzame hlavnú melódiu, nie nutne v závislosti na dobách a taktoch, ale v závislosti na melódií. Vybrať a správne interpretovať melodický celok je však v iných ako najjednoduchších skladbách veľmi zložitou záležitosťou – už len identifikovať melodický celok je náročné. Základom hudobného diela je jeden alebo viacero motívov, ktoré sa spájajú do jednej, prípadne viacerých tém, pričom tému už môžeme považovať za melodický celok. Z tém sa ďalej formujú jednotlivé myšlienky, ktorých takisto býva v diele viacero (napr. hlavná myšlienka, medziveta, vedľajšia myšlienka, druhá medziveta atď.) a ich usporiadanie sa líši podľa konkrétnej formy (piesňová, rondová, sonátová atď.). Navyše melodický celok nebýva v dielach opakovaný úplne presne, bežná je transpozícia, skrátenie, predĺženie i variácia. Zložitosť takejto detekcie nám teda zabraňuje pozerať sa na hudobné dielo z melodického hľadiska a hľadiska melodických celkov.

Zdalo by sa, že ak pozeráme na hudobný celok z hľadiska delenia do logických časových úsekov nemôžeme proces harmonizácie využiť. To ale celkom pravda nie je, pretože častokrát sa melodické celky zhodujú s časovými, a to najmä s taktami, typické bývajú štvor- či osemtaktia. Všeobecne to síce pravda nebýva, ale harmonizácia nám dáva aspoň aký - taký návod ako riešiť výber vhodného reprezentanta. Jednoducho využijeme niektoré aspoň základné poznatky z teórie harmónie a budeme sa ku každému logickému úseku správať ako k (takmer) samostatnému melodickému celku. Takisto nebude kladený až taký veľký dôraz na správne spájanie akordov, ako skôr na dĺžky a počet jednotlivých tónov v dobe. Problém so spájaním akordov je totiž ten, že neexistuje jednoznačné priradenie akordov daným úsekom – „...hudba, počnúc 17. storočím, vychádza z harmónie ako

zo základného princípu. V harmonickej stránke sa odrážajú celkom jasne všetky slohové zmeny a dá sa dokonca povedať, že veľké umelecké osobnosti majú svoj vlastný vyjadrovací spôsob, ktorý sa prejavuje aj v oblasti harmónie niekedy dosť svojrázne.“. [2] Znamená to teda, že jedna skladba môže byť zharmonizovaná viacerými spôsobmi, čo ešte viac vyniklo nástupom modernej hudby, kedy sa začali používať do tej doby zakázané (de facto, nie de iure) spojenia.

3.4 Metódy výberu reprezentanta

V práci sú použité dve metódy výberu reprezentanta, ktorého budeme nazývať signifikantným tónom. Obe tieto metódy používame ako na logickú jednotku doba, tak aj na logickú jednotku takt. Prvá je metóda jednoduchá, kedy iba spočítame dĺžku trvania jednotlivých tónov v dobe, resp. takte, takže výsledkom je pole dvanástich tónov, ktorého obsahom sú spočítané dĺžky. Za signifikantný tón označíme ten, ktorý má túto dĺžku maximálnu. Určité poznatky z hudobnej teórie tu využívame iba v prípade zhody maximálnych dĺžok, v tom prípade sa snažíme najskôr vybrať tóny, ktoré majú väčšiu dôležitosť v rámci tóniny skladby, alebo konkrétnej doby, t.j. prvý tón toniky, dominanty a subdominanty, tóny dominantného septakordu. Ak ani jeden z týchto tónov nezdieľa spoločnú maximálnu dĺžku, vyskúšame všetky ostatné tóny ktoré do tóniny patria, tých je vždy sedem (aj s predchádzajúcimi tónmi). V prípade že ani jeden z týchto tónov spoločné maximum nezdieľa (čo sa stáva zriedkakedy, resp. v prípade že sa to stane by sme mali skontrolovať či má skladba správne nastavenú tóninu) vyskúšame jeden zo zostávajúcich piatich mimotonálnych tónov. Samozrejme, pokiaľ sme napríklad v tónine C dur a v dôsledku zlyhania všetkých predchádzajúcich kritérií sa rozhodujeme povedzme medzi tónmi Cis a Es, s informáciami ktoré sú pre nás dostupné je v podstate jedno, ktorého z nich označíme ako reprezentanta – oba sú totiž mimotonálne a je pre nás nemožné určiť, ktorý je vhodnejší, preto u nich už nezáleží na poradí.

Druhá metóda už z teórie harmónie využíva viac poznatkov, pozostáva z dvoch častí. V prvej časti opäť sčítame dĺžky jednotlivých tónov, ale namiesto jednoduchého výberu maxima sa uplatnia zložitejšie kritériá. Tón sa prehlási za signifikantný iba v jednom z nasledujúcich prípadov:

- zaberá tri štvrtiny celkovej dĺžky všetkých tónov v danom logickom celku
- zaberá viac ako polovicu s tým že každý iný tón zaberá menej ako štvrtinu
- je jedným z primárnych tónov (za primárne označujem prvý tón toniky, dominanty a subdominanty) a zaberá viac ako tri štvrtiny noty – ale iba v prípade, že žiadny iný signifikantný tón nezaberá takisto viac ako tri štvrtiny
- pokiaľ je to prvý tón toniky a zaberá viac ako polovicu dĺžky noty.

V prípade, že sa ani jedným z týchto kritérií nepodarí určiť signifikantný tón, skúsime za pomoci rovnakých kritérií určiť tón nadchádzajúci a ak sa to podarí, vrátime sa späť a skúsime vybrať reprezentanta s informáciou o nasledujúcom a predchádzajúcom signifikantnom tóne. Ak sa nám nadchádzajúci tón určiť nepodarí, nepokračujeme ďalej, ale opäť sa vrátime a vyberieme reprezentanta aj bez pomoci tohto nadchádzajúceho signifikantného tónu, tentoraz iba s využitím predchádzajúceho. To všetko za pomoci podobných, i keď o niečo podrobnejších kritérií ako v prvej časti metódy. Ak zlyhajú aj tieto kritéria a s ich využitím sa reprezentanta zvoliť nepodarí, využijeme jednoduchý výber maxima a v prípade viacerých maximálnych dĺžok sa rozhodujeme pomocou rovnakých kritérií ako v jednoduchej metóde spomínanej na začiatku tejto podkapitoly.

Pomlčku vyberáme ako reprezentanta iba v prípade že v celom logickom celku nie sú žiadne ďalšie tóny. Znamená to teda, že ak sú v logickom celku spolu ľubovoľné tóny a pomlčka, dĺžku pomlčky ignorujeme a zameriavame sa iba na znejúce tóny a ich dĺžky.

3.5 Oktáva a označenie tónov

Oktáva je interval medzi vybraným tónom a tónom, ktorý má polovičnú, alebo dvojnásobnú frekvenciu ako tón vybraný. Ľudské ucho má tendenciu počuť dva tóny

ktoré sú v oktáve ako rovnaké, preto sa takéto tóny v hudobnej terminológií označujú rovnako – tzn. tón oktávu pod tónom A sa bude opäť označovať A, takisto tón oktávu nad. V západnej hudbe (tzn. prakticky v každom hudobnom diele s ktorým sa dá prísť do styku) existuje označenie pre 12 základných frekvencií (hovoríme o delení do poltónov), ostatné tóny sú v oktáve s nejakým z týchto dvanástich. Vzhľadom k tomu že ľudské ucho počuje tóny vzdialené od seba o oktávu prakticky rovnako, je informácia o oktáve nepodstatná aj z harmonického hľadiska. Pri porovnaní nás teda bude zaujímať najmä to, či sa rovnajú označenia jednotlivých tónov, ktoré porovnávame bez ohľadu na príslušnosť k oktáve. Takéto tóny prehlásime za rovnaké aj keď ich frekvencie sú de facto rôzne. Takisto pri výbere signifikantných tónov, t.j. reprezentantov jednotlivých logických celkov, nebudeme dbať na oktávu ale práve na tónové označenie.

3.6 Vplyv transpozície na porovnávanie

Ďalšou dôležitou vecou, ktorú si musíme uvedomiť, je vplyv tóniny na porovnávané reťazce. Sekvencia tónov C, E, G v tónine C dur je totiž úplne rovnaká ako sekvencia D, Fis, A v tónine D dur, táto je akurát posunutá o tóninu vyššie. Tento jav, nazývaný transpozícia, musíme brať do úvahy, pretože v opačnom prípade by nám dve rovnaké, iba transponované skladby vo výsledku dávali obrovský rozdiel. Jedným z možných riešení by bolo transponovať porovnávané skladby do rovnakej tóniny, napríklad už pri vstupe. To by však znamenalo zbytočný priebeh celého hudobného reťazca a vzhľadom k tomu, že tento musíme aj tak pri samotnom porovnávaní prebiehať, naskytá sa elegantnejšie riešenie a to dopočítavať prípadný vplyv transpozície až keď je to naozaj potrebné. Veľmi dôležitá je v tomto prípade z hľadiska rýchlosti výpočtu absencia logických podmienok, takže sa musíme danú transpozíciu snažiť dopočítať nezávisle na konkrétnych tónoch a tóninách.

Keďže máme iba dvanásť (pol)tónov, a trinástu pomlčku, a navyše sú tieto poltóny za sebou zoradené tak isto ako tóniny, dá sa celkom ľahko zostaviť všeobecná rovnica pre dva porovnávané tóny. Majme dva porovnávané ľubovoľné tóny T a S, ktoré sú v tóninách A, respektíve B. Majme tóny (a tým pádom i tóniny) očíslované od 0 po 12 tak, že tón C má číslo 0, tón H číslo 11 a pomlčka číslo 12.

Nech operácia div označuje celočíselné delenie, a operácia mod zvyšok po celočíselnom delení. Nech

$$P_1 = T \text{ div } 12$$
$$P_2 = S \text{ div } 12.$$

Tieto dva indexy označujú, či je daný tón pomlčka alebo nie. V prípade že je tón pomlčkou nadobudne index hodnotu 1, v opačnom hodnotu 0. Ďalej, definujme

$$I_1 = (12 - A + T) \bmod 12$$
$$I_2 = (12 - B + S) \bmod 12,$$

kde I_1 a I_2 nadobúdajú hodnoty od 0 do 11 v základnej tónine (C dur alebo a mol) v závislosti na konkrétnom tóne a transpozícií – tzn. pokiaľ je tón v poradí desiaty vo svojej stupnici, dostane číslo desať. Pomlčka tu ale dostáva číslo vždy rovnaké ako číslo tóniny v ktorej sa nachádza, definujme preto:

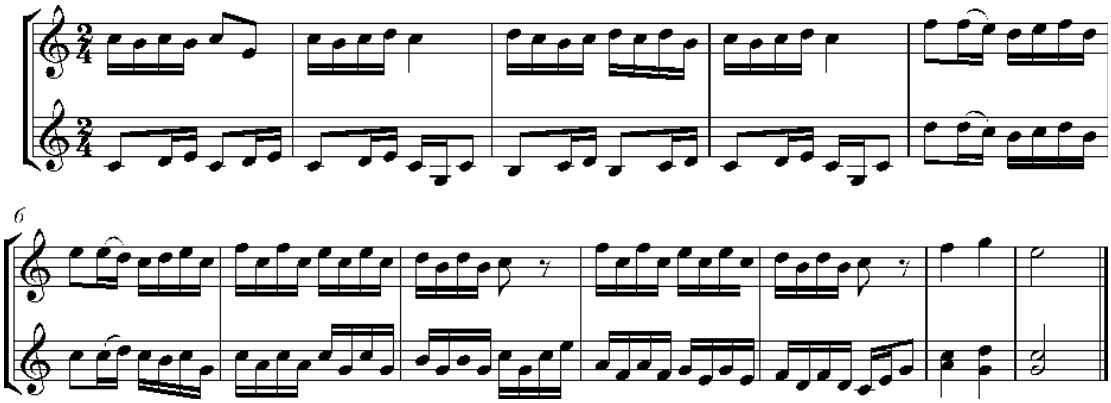
$$V_1 = (I_1 + P_1*(12-I_1)) \bmod 13$$
$$V_2 = (I_2 + P_2*(12-I_2)) \bmod 13,$$

kde výraz $P_1*(12-I_1)$ je nenulový iba v prípade že je tón T pomlčka a teda v prípade, že je tón T pomlčka sa V_1 rovná 12, teda skutočne číslu pomlčky, inak sa rovná I_1. Z uvedeného teda vyplýva, že V_1 a V_2 sú čísla tónov T a S transponované do základnej tóniny a môžeme ich teda medzi sebou priamo porovnávať.

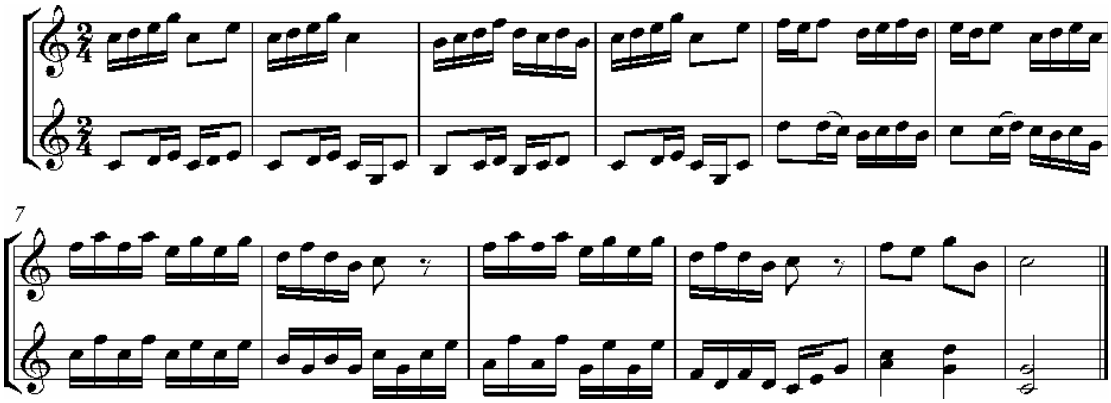
Kapitola 4

Ukážka výstupov ohodnocujúcej funkcie a algoritmov

Skúsme sa teraz pozrieť na dve konkrétne harmonicky úplne rovnaké skladby s podstatne odlišnou melódiou – najprv ich uvedieme v notovom zápise a pod ne uvedieme výstupy (zložitejšej) ohodnocovacej funkcie pre tieto skladby. Obe skladby sa nachádzajú na priloženom CD pod názvami „skladba2_1.mid“ a „skladba2_2.mid“.



Obr. 2 Prvá skladba



Obr. 3 Druhá skladba

Prvá skladba prvá stopa:

Doby: C C C C C D C C F D E C C C D C C C D C F G E E
Takty: C C C C F E C C C C G E

Prvá skladba druhá stopa:

Doby: C C C C H H C C D H C C C C C G C E G E G C G C C
Takty: C C H C D C C G G G C C

Druhá skladba prvá stopa:

Doby: C C C C C D C C C F D E C F G D F G D C F G C C
Takty: C C D C F E E C G C G C

Druhá skladba druhá stopa:

Doby: C E C C H D C C D H C C C C C G C F G F G C G C C
Takty: C C D C D C C G G G C C

Z uvedených výstupov ohodnocovacích funkcií vidíme, že funkcia dáva i pre dve pomerne odlišné skladby až na drobné odlišnosti značne podobné výstupy. Pozrime sa ešte na tabuľku výsledkov ktoré dávajú jednotlivé algoritmy v závislosti na ohodnocovacích tabuľkách.

Algoritmus	Editačná vzdialenosť								Local suffix alignment							
Tabuľka	Základná				Durová				Podsekvencia				Podreťazec			
Skladba/ Stopa	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2
1/1	0	130	60	130	0	97	36	93	24	14	18	14	24	4	12	4
1/2	130	0	110	20	97	0	87	14	16	24	17	22	4	24	4	18
2/1	60	110	0	110	36	87	0	83	18	17	24	17	12	4	24	4
2/2	130	20	110	0	93	14	83	0	14	22	17	24	4	18	4	24

Tab. 4 Výsledky algoritmov podľa dôb

Algoritmus	Editačná vzdialenosť								Local suffix alignment							
Tabuľka	Základná				Durová				Podsekvencia				Podreťazec			
Skladba/ Stopa	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2	1/1	1/2	2/1	2/2
1/1	0	80	40	80	0	50	25	50	12	7	8	7	12	3	3	3
1/2	80	0	70	10	50	0	44	9	7	12	8	11	3	12	3	9
2/1	40	70	0	60	25	44	0	35	8	8	12	8	3	2	12	3
2/2	80	10	60	0	50	9	35	0	7	11	8	12	3	9	3	12

Tab. 5 Výsledky algoritmov podľa taktov

V riadkoch tabuľky sú uvedené čísla stôp a skladieb, kde napr. 1/2 znamená že je to druhá stopa prvej skladby. Tak isto sú čísla skladieb a stôp uvedené aj v jednotlivých stĺpcoch tabuľky. Samotné výsledky sú uvedené pre každý algoritmus a tabuľku zvlášť, takže ak hľadáme napríklad výsledok editačnej vzdialenosti druhej stopy prvej skladby a druhej stopy druhej skladby za použitia ohodnocovacej tabuľky pre durovú tóninu, vyberieme si stĺpec s názvom „Durová“ a v ňom sa pozrieme na hodnou danú stĺpcom s označením 1/2 a riadkom s označením 2/2, prípadne vice versa. Takisto si musíme uvedomiť čo jednotlivé hodnoty znamenajú. U editačnej vzdialenosti je v základnej tabuľke penalizácia 10 za každú nezhodu, zhoda je ohodnotená ako 0. Podobne je tomu aj v tabuľke pre durovú tóninu, kde je väčšina nezhôd tiež ohodnotená 10, ale niektoré príbuzné tóny (napr. C,G, keďže sa pohybujeme v tónine C dur) majú túto penalizáciu nižšiu. Takže dve úplne odlišné skladby trvajúce 24 dôb by mali mať teoreticky vzdialenosť 240 – takže už

vzdialenosť 130 znamená že naše dve stopy sú takmer spolovice rovnaké, no a vzdialenosť 20 znamená veľmi veľkú podobnosť. V tabuľke pre najdlhšiu spoločnú podsekvenciu u algoritmu Local suffix alignment je každá zhoda ohodnotená ako 1, nezhoda ako 0. Tu majú teda pre nás relevantné výsledky úplne iné hodnoty, dostatočne dlhá podsekvencia by mohla nadobúdať hodnoty okolo 20. Presné hodnoty ohodnocovacích tabuliek sú uvedené na priloženom CD.

Vidíme teda, že výsledky porovnania najmä druhých stôp sú veľmi podobné, čo je však do značnej miery ovplyvnené ohodnocujúcou funkciou. Takisto je veľmi badateľný vplyv ohodnocovacej tabuľky najmä u editačnej vzdialenosti, kde na niektorých miestach klesla výsledná vzdialenosť až na polovicu. Výsledky sú z hľadiska podobnosti veľmi uspokojujúce, ak sa pozrieme na editačnú vzdialenosť 36 (so základnou jednotkou vzdialenosti 10) v prvých hlasoch oboch skladieb, je dosť malá na to, aby aj v prípade, že by sme o oboch skladbách dopredu nič nevedeli, upútala našu pozornosť ak by sa objavila vo výsledkoch napríklad dávkového spracovania a poukázala na značnú podobnosť týchto dvoch skladieb. Výsledok 14 u druhých hlasov týchto skladieb je takisto slušný, tu si ale musíme uvedomiť značnú podobnosť oboch skladieb už v notovom zápise.

U algoritmu local suffix alignment výsledky takisto zodpovedajú očakávaniam. Najdlhšia spoločná sekvencia 18 v prvých hlasoch je dostatočne dlhá, o číslo 22 v hlasoch druhých naznačuje vysokú mieru podobnosti. Najdlhší spoločný podreťazec algoritmus takisto vyhľadal správne.

Výsledky porovnania zvolených dvoch skladbičiek sú teda uspokojivé, musíme si ale uvedomiť že je to sčasti dané tým že sú si pomerne podobné. Ak by sme napríklad pustili algoritmus editačná vzdialenosť na dve dlhé skladby (rádovo stovky dôb) ktoré by mali spoločné podobné úseky dlhé cca. 15-30 dôb, algoritmus by nám túto zhodu nebol schopný vyhodnotiť. V takomto prípade by bol správny postup najskôr pustiť algoritmus Local sufix alignment kde by sme našli nejaké spoločné podreťazce alebo podsekvencie rozumnej dĺžky a až potom aplikovali editačnú vzdialenosť na úseky určené týmito celkami. Na zistenie existencie rovnakého alebo podobného vzorku v skladbách ale už môžeme spustiť akýkoľvek z algoritmov.

Kapitola 5

Implementácia

5.1 Formát vstupných dát

Ako vstupné dáta sa pre túto prácu javia byť najvhodnejšie MIDI súbory, ktoré sa už samé najviac blížia notovému zápisu. Používať akýkoľvek audio formát, kde sú dáta uložené ako nasamplované vlnenie by so sebou prinieslo veľké problémy už pri konverzií takýchto dát do notového zápisu (respektíve použitých dátových štruktúr) a preto sa pre naše účely nehodí. MIDI je skratka pre Musical Instrument Digital Interface, primárnym účelom tohto štandardu je poskytnúť jednotný interface pre elektrické nástroje, počítače a iné vybavenie, cez ktoré tieto môžu komunikovať a synchronizovať sa v reálnom čase. To sa deje pomocou tzv. MIDI správ. Štandard MIDI definuje 16 samostatných kanálov (ekvivalent stopy), kde každý kanál slúži pre samostatný nástroj.

MIDI súbory sú sekvenčné súbory, ktoré pozostávajú z tzv. „chunks“ – existujú dva typy a to riadiaci, tzv. „header“ chunk, ktorý obsahuje informácie týkajúce sa celého súboru, tzn. správy týkajúce sa zmeny tempa, taktového označenia či predznamenania používajú typicky tento kanál. Za ním nasledujú kanály týkajúce sa jednotlivých stôp. Správy, v sekvenčnom MIDI súbore nazývané udalosti, pozostávajú z „delta-time“, čiže času ktorý ubehol od poslednej udalosti, a samotnej MIDI správy. Pre nás sú dôležité udalosti týkajúce sa začiatku a konca jednotlivých tónov - Note On/Off správy. Tieto správy obsahujú okrem delta-time ešte číslo noty, ktorá má byť začatá resp. ukončená (od 0 po 127) a tzv. „velocity“, silu, s ktorou bola nota zahraná. (pre kompletnú tabuľku jednotlivých tónov spolu s frekvenciami

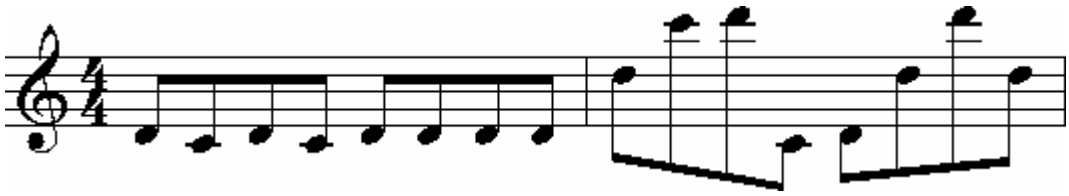
viď [3,str. 31]). Ďalej sú dôležité správy týkajúce sa taktového označenia a to z hľadiska rozdelenia jednotlivých dôb do taktov, správy týkajúce sa predznamenania, a správy týkajúce sa tempa, t.j. času trvania jednej doby, ktoré sú dôležité z hľadiska rozdelenia nôt do jednotlivých dôb. Ostatné správy ktoré MIDI formát definuje nie sú z hľadiska tejto práce dôležité.

Na prácu so sekvenčnými MIDI súbormi, ich parsovanie a prehrávanie sme využili knižnicu pre jazyk C++ libjdkimidi dostupnú na stránke [4].

5.2 Použité dátové štruktúry

5.2.1 Trieda Tone

Použité dátové štruktúry v zásade reflektujú logické členenie hudobného celku. Základnou je trieda Tone, ktorá je ekvivalentom jedného tónu. Táto v sebe obsahuje číslo daného tónu podľa formátu MIDI, a tónové označenie tohto tónu, kde vynechávame informáciu o oktáve (viď kapitola 3.5). MIDI formát definuje 128 tónov v celkovo 11 oktávach. Je teda výhodné mať uložené aj tónové označenie, ktoré sa bude vo veľkej miere využívať v ďalších výpočtoch. Možno by dokonca stačilo mať uložené iba tónové označenie, ale to by zasa zabránilo detailnejšiemu porovnávaniu, kde by mohla informácie o oktáve hrať významnú úlohu – viď príklad na obrázku 4, kde sú oba takty tónovo rovnaké, jednotlivé tóny sa od seba odlišujú iba v oktávach.



Obr. 4 Rozdiel medzi tónmi v rôznych oktávach

5.2.2 Trieda Note

Trieda Note je ekvivalentom jednej noty, obsahuje v sebe zoznam jednotlivých tónov. Ďalej v sebe obsahuje svoju dĺžku, čoho je využívané pri porovnávaní jednotlivých nôt medzi sebou.

5.2.3 Trieda Beat

Trieda Beat je ekvivalentom jednej doby, okrem zoznamu nôt z ktorých sa skladá obsahuje aj položku označujúcu tóninu tejto doby. Je to preto, že v hudobných dielach nie je nezvyčajné zmeniť tóninu uprostred skladby, (viď napr. tangá), alebo transponovať iba určitú hudobnú frázu (typicky napr. 8 taktov) a potom sa vrátiť naspäť do pôvodnej tóniny. Vzhľadom k tomu že harmonizácia ja viazaná na ústredný tón určitej stupnice, nestačí nám, v prípade že bola nejaká fráza transponovaná, iba informácia o tónine (predznamenaní) celej skladby, ale potrebujeme si túto informáciu uchovávať vzhľadom na aktuálny kus skladby v ktorom sa práve nachádzame.

Určitou nepresnosťou vo vzťahu Doba – Nota je fakt, že vzhľadom k tomu že štandard MIDI vníma dobu ako čas trvania jednej štvrťovej noty, tak noty dlhšie ako táto, t.j. polová a celá nemôžu byť v jednej dobe obsiahnuté. Riešením je rozdeliť tieto dlhšie doby do príslušného počtu dôb, pretože z melodického a harmonického hľadiska tým žiadna zmena nenastane. Ďalšou nepresnosťou je absencia značiek pre spájanie tónov, t.j. legát a ligatúr. Tieto sú pre účely práce ignorované, pretože opäť neprinášajú žiadnu zmenu z melodického a harmonického pohľadu. (štandard MIDI ale definuje správu ktorá spôsobuje efekt spájania tónov, takže by bolo možné zachytiť aj tieto značky)

5.2.4 Trieda Track

Trieda Track je ekvivalentom jedného hlasu, respektíve stopy. Obsahuje zoznam jednotlivých dôb, vektor ohodnotenia týchto dôb a takisto vektor ohodnotenia

jednotlivých taktov. Ďalšími položkami sú tónina skladby, taktové označenie skladby, tempo skladby a položky ktoré zaisťujú prehrávanie skladby.

Za zmienku stojí absencia logického celku takt v použitých dátových štruktúrach. Je to preto, že takt ako logický celok iba združuje jednotlivé doby, a teda nám stačí vedieť koľko dôb do taktu patrí, a držať si vektor jeho ohodnotení. Snáď by bolo možné podobne ako u jednej doby mať ešte položku pre aktuálnu tóninu, ale keďže sa takt skladá z dôb, stačí sa pozrieť na tóniny dôb vnútri taktu, respektíve na tóninu prvej doby, keďže sa tónina v prostriedku taktu nemení (skoro) nikdy. Ako riešenie by sa v tomto prípade mohlo javiť to, že namiesto logického celku doba, si teda budeme držať iba celok takt, ktorý bude už priamo obsahovať jednotlivé noty. To však nie je celkom vhodné z hľadiska toho, že najmä v klasickej hudbe je často harmonizovaná každá doba osobitne a tento spôsob je tiež jedným z použitých v práci. Absencia logického celku takt má aj tú výhodu, že môžu byť použité rovnaké funkcie na ohodnocovanie ako taktov, tak aj dôb, keďže oboje sú vektory v triede Track.

5.2.5 Trieda All_Tracks

Trieda All_Tracks je ekvivalentom celej skladby. Okrem zoznamu jednotlivých stôp obsahuje aj položky pre tempo, taktové označenie a tóninu, a položky ktoré zaisťujú prehrávanie skladby. Duplicita jednotlivých položiek v tejto triede aj v triede Tracks je spôsobená tým, že najskôr sa prečíta header chunk a až potom sa spracovávajú ostatné kanály, kde sa táto informácia potom distribuuje do jednotlivých stôp (trieda Track), pretože častokrát potrebujeme pracovať so samostatnou stopou a museli by sme zakaždým zaistiť prenos týchto informácií inou cestou. Ďalším dôvodom je ten, že v modernej hudbe nie sú nezvyčajné, či už skladby v ktorých sú paralelne vedené melódie v rôznych tóninách, alebo dokonca skladby, kde súčasne znejú melódie hrané v rôznych tempách.

5.2.6 Ďalšie pomocné štruktúry

Medzi ďalšie pomocné štruktúry patrí trieda Harmony, ktorá uchováva harmóniu (predznamenanie). Jej položkami sú počet krížikov/béčok a informácia o type tóniny. Pre jednoduchosť sú uvažované iba dva základné typy – dur a mol, i keď správnejšie by kvôli harmonizácií, resp. ohodnocovacej funkcií bolo rozdeliť molovú tóninu ešte na prirodzenú, harmonickú a melodickú, ale MIDI formát podporuje bohužiaľ iba základné rozdelenie. Ďalšou je štruktúra Measure, ktorej položkami sú čitateľ a menovateľ taktového označenia. Patria sem aj jednotlivé triedy pre formuláre v implementácií užívateľského prostredia.

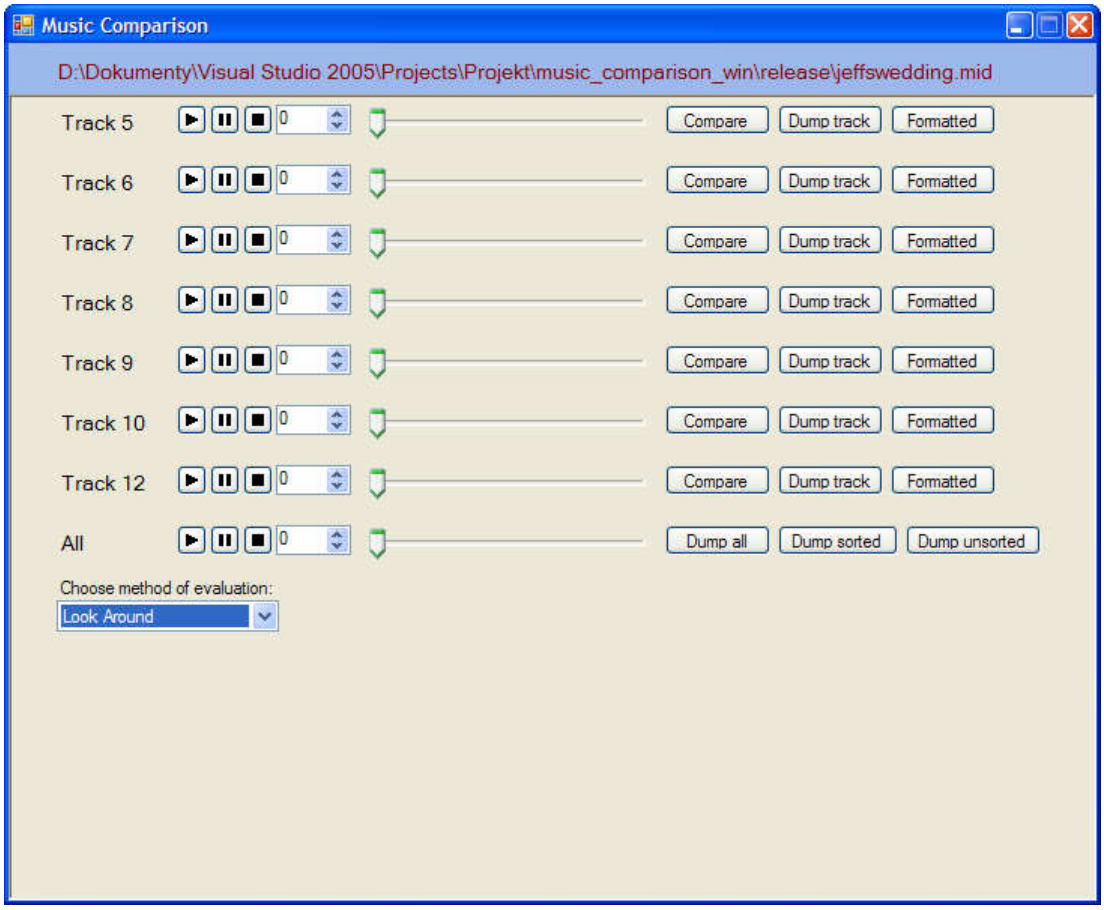
Ďalšie pomocné dátové štruktúry sa týkajú samotného porovnávania hudby, sem patria triedy Table_Cell, čo je položka dynamickej tabuľky pri algoritme Local Suffix Alignment a LSA_Result ktorá slúži na prezentáciu výsledkov tohto algoritmu. (Samotný algoritmus viď kapitola 2.) Trieda Value_Table plní funkciu distančnej tabuľkou pre jednotlivé algoritmy.

5.3 Návrh GUI a ovládania aplikácie

Súčasťou a jedným z cieľov práce je i vytvorenie aplikačného prostredia tak, aby bolo čo najjednoduchšie získavať dáta a vyhodnocovať dosiahnuté výsledky. Na tento účel bolo využité prostredie Microsoft .NET Framework, a to najmä kvôli výhode množstva predpripravených riešení v oblasti užívateľského rozhrania, ktoré užívateľ dobre pozná, pretože sa s nimi denne stretáva pri práci s počítačom a teda si nemusí zvykať na nové nezvyklé riešenia a zložito sa učiť novú aplikáciu používať. Skúsime si teraz ukázať vzhľad užívateľského rozhrania a jednotlivých dialógov. Podrobný popis ovládacích prvkov a práce s nimi sa nachádza v užívateľskej dokumentácií na priloženom CD.

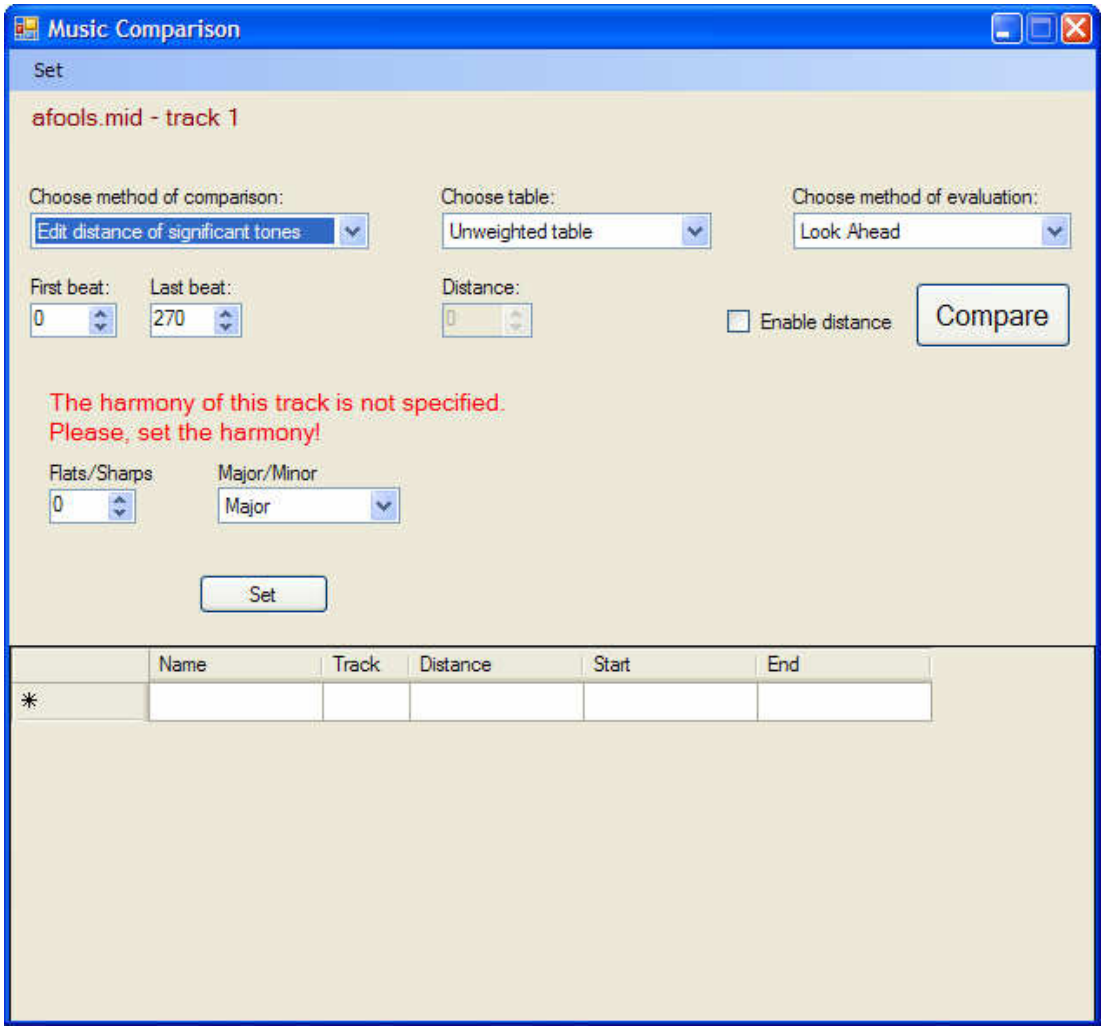
Po spustení sa objaví úvodne okno, kde je možnosť vybrať si pieseň ktorú chceme porovnávať. Po tomto výbere sa nám zobrazí ovládacie okno tejto piesne (obr. 5), kde si môžeme prehrať jednotlivé stopy, prípadne celú skladbu, a rozhodnúť sa, ktorú z nich chceme porovnávať. Takisto je tu možnosť textového výstupu

jednotlivých stôp, či už vo formáte MIDI alebo formátovaný výstup aj s výsledkami ohodnocujúcej funkcie.



Obr. 5 Ovládacie okno aplikácie

Ak sa rozhodneme porovnávať stopu, otvorí sa nám porovnávacie okno (obr.6), kde môžeme nastaviť spôsob porovnávania a ohodnocovaciu tabuľku, takisto ako môžeme vybrať úsek skladby ktorý chceme porovnávať a obmedzenie výsledkov ak požadujeme výsledky napr. do maximálnej editačnej vzdialenosti 50. Pokiaľ sa programu nepodarilo zistiť tóninu, alebo taktové označenie skladby, budeme na to upozornení a nútení tieto položky nastaviť. Po spustení porovnávania bude možnosť vybrať jednu alebo viacero skladieb s ktorými chceme porovnávať a začne samotné porovnávanie. Počas neho, pokiaľ opäť jednotlivé skladby nebudú mať nastavenú tóninu alebo taktové označenie, program nás na to upozorní, a my budeme mať možnosť tieto položky nastaviť, prípadne celú skladbu preskočiť.



Obr. 6 Porovnávacie okno aplikácie

Výstupy porovnávacích algoritmov sa zapisujú do tabuľky (obr. 7), ktorej položky môžeme zoradiť podľa jednotlivých stĺpcov, aby sme mohli ľahko získať najrelevantnejšie výsledky. Jednotlivé skladby je možné otvárať priamo z tejto tabuľky a ich stopy si prehrať, aby sme mohli ľahko overiť podobnosť daných skladieb.

	Name	Track	Distance	Start	End
Open:	skladba2_2.mid	1	60	0	23
Open:	skladba2_2.mid	2	130	0	23
Open:	piesen1_1.mid	2	180	1	24
Open:	piesen1_2.mid	2	180	2	25
Open:	piesen1_transposed.mid	2	180	3	26
Open:	piesen1_2.mid	2	180	3	26
Open:	piesen1_transposed.mid	2	180	1	24
Open:	piesen1_1.mid	2	180	3	26
Open:	piesen1_transposed.mid	1	180	8	31

Obr. 7 Tabuľka výsledkov

Kapitola 6

Záver

Napriek komplexnosti a nelinearite hudobného diela sa podarilo vytvoriť nástroj, ktorý z tohto diela vytvorí štruktúru vhodnú na porovnávanie pomocou reťazcových algoritmov. S využitím poznatkov z teórie harmónie sme boli schopní zamerať sa popri porovnávaní podľa základných stavebných jednotiek aj na porovnávanie podľa väčších logických celkov, takže popri exaktnom porovnávaní tón po tóne sme boli určitým spôsobom schopní zachytiť aj harmonickú a melodickú podobnosť. Voľba algoritmov určených na porovnávanie samotných reťazcov sa tiež ukázala ako správna, najmä kvôli možnosti zamerať sa nielen na presné zhody dvoch reťazcov, ale aj na situácie kedy tieto neboli úplne ekvivalentné. Ako dôležité, najmä u editačnej vzdialenosti, sa ukázalo zavedenie distančných tabuliek, kde môžeme ešte presnejšie popísať harmonickú a tonálnu príbuznosť jednotlivých tónov.

Výstupy ohodnocovacích funkcií a porovnávacích algoritmov sú taktiež veľmi uspokojivé, kde podobné skladby majú očakávanú nízku editačnú vzdialenosť, resp. dlhé spoločné podsekvencie a podreťazce. Tu je však kladený dôraz na to, aby bol užívateľ dopredu oboznámený z použitými distančnými tabuľkami a hodnotami ktoré môže očakávať, pretože bez tejto znalosti nie je možná správna interpretácia dosiahnutých výsledkov.

Prínos práce vidíme najmä v oblasti hudobnej teórie, kde môže pomôcť k lepšiemu pochopeniu previazanosti jednotlivých období či žánrov, takisto ako aj k náhľadu na využívanie rovnakých, respektíve podobných, harmonických a melodických postupov, kde znalosť ktoré postupy sú najpoužívanejšie, resp.

naopak neveľmi používané môže viesť k pochopeniu úspechu, či naopak práve neúspechu rôznych hudobných diel, a tým k lepšiemu pochopeniu zásad kompozície.

Ak by sme uvažovali o budúcnosti práce, možné vylepšenia by sa pravdepodobne týkali najmä ohodnocovacej funkcie, kde by sa dalo využiť ešte viac poznatkov z teórie harmónie, najmä z oblasti vedenia hlasov a spájania jednotlivých akordov, ako aj príbuznosti kvintakordov, čo by umožnilo viac sa priblížiť k harmonizácií v pravom slova zmysle. Tým by sme dosiahli možnosť porovnávať skladby z čisto harmonického hľadiska, kde by sme nemuseli brať melodické hľadisko vôbec do úvahy (resp. melódia by sa zohľadňovala iba pri samotnom procese harmonizácie), čo by mohlo priniesť ešte iný pohľad na podobnosť hudobných skladieb.

Takisto sa naskytá možnosť implementácie viacerých algoritmov primárne určených na porovnávanie reťazcov. Jedným zo zaujímavých by mohol byť napríklad algoritmus uvedený v [1], ktorý je založený na skutočnosti že v reťazcoch vykazujúcich podobnosť sa striedajú úseky, ktoré sú si veľmi podobné, až identické, s úsekmi, ktoré sú navzájom absolútne odlišné. Snaha je teda zamerať sa čo najviac na tieto podobné úseky a nájsť ich najlepšie zarovnanie, pričom rozdielne úseky, ktoré ich oddeľujú budeme jednoducho brať ako medzery, resp. prázdne miesto. Ďalšími možnými vylepšeniami sú napríklad práca na distančných tabuľkách či vlastný parser MIDI súborov.

Použitá literatúra

[1] Dan Gusfeld: Algorithms on strings, trees and sequences, University of Cambridge Press, 1997

[2] Jaroslav Kofroň: Učebnice Harmonie, Supraphon, 1961

[3] Pete Goodliffe: MIDI documentation, 1999

[4] C++ MIDI Library, http://www.jdkoftinoff.com/main/Free_Projects/C++_MIDI_Library

[5] Článok o psychoakustike, http://en.wikipedia.org/wiki/Psychoacoustics

Dodatok A

Obsah priloženého CD

Na priloženom CD nájdete:

1) Samotnú aplikáciu – adresár Music_Comparison
2) Projekt pripravený na preloženie v Microsoft Visual Studio .NET a zdrojové súbory - adresár Project
3) Programátorskú dokumentáciu – adresár Development
4) Užívateľskú dokumentáciu – adresár Help
5) Použité ohodnocovacie tabuľky – adresár Tables
6) Vzorové MIDI súbory s popisom – adresár Samples
7) Text tejto práce – adresár Bakalarka