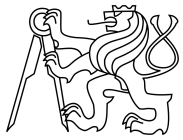
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Podobnostní vyhledávání klavírních skladeb
Student:	Duc Anh Mai
Vedoucí:	Ing. Jiří Novák, Ph.D.
Studijní program:	Informatika
Studijní obor:	Web a multimédia
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

1) Nastudujte problematiku podobnostního vyhledávání v audiu.
2) Proveďte rešerši existujících aplikací, které lze využít pro podobnostní vyhledávání klavírních skladeb.
3) Proveďte rešerši existujících knihoven pro extrakci deskriptorů z audia (např. MPEG-7 audio encoder, jAudio). Stručně popište jednotlivé deskriptory a vysvětlete jejich význam. Popište podobnostní funkce používané pro porovnávání dvou audio záznamů.
4) Navrhněte vlastní řešení pro výpočet podobnostní funkce na základě výběru vhodných deskriptorů.
5) Implementujte aplikaci pro mobilní zařízení na platformě Android pro porovnání záznamu zvukové stopy od uživatele (nahrávka klavírní skladby) s její originální verzí a databází skladeb.
6) Experimentálně otestujte aplikaci z hlediska kvality hledání skladeb a z hlediska rychlosti.
7) Porovnejte vaši aplikaci s existujícími aplikacemi.

Seznam odborné literatury

[1] H-G. Kim, N. Moreau & T. Sikora. “MPEG-7 Audio and Beyond“: Willey, 2005.
[2] B. S. Manjunath, P. Salembier & T. Sikora. “Introduction to MPEG-7”: Willey, 2002.
[3] H-G. Kim, N. Moreau & T. Sikora. “Audio Classification Based on MPEG-7 Spectral Basis Representations“. IEEE Trans. Circuits Syst. Video Technol., vol. 14, 2004.
[4] G. Peeters, S. McAdams, P. Herrera. “Instrument Sound Description in the Context of MPEG-7“. International Computer Music Conference, Berlin, 2000.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 10. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Podobnostní vyhledávání klavírních skladeb

Mai Duc Anh

Vedoucí práce: Ing. Jiří Novák, Ph.D.

16. května 2016

Poděkování

Chci poděkovat všem, kteří mi věnovali svůj čas a pomohli mi při psaní mé bakalářské práce. Zejména svému vedoucímu, panu Ing. Jiřímu Novákovi, Ph.D. za cenné podněty a poskytnutý prostor pro samostatnost. Děkuji Vandě Michalské, své rodině a všem blízkým za bezvýhradnou podporu během mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Duc Anh Mai. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Mai, Duc Anh. *Podobnostní vyhledávání klavírních skladeb*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Práce se zabývá problematikou podobnostního vyhledávání klavírních skladeb a porovnáváním zvukových stop. Popisuje možnosti získání dat ze zvukových nahrávek a způsoby jejich porovnání. Na základě současných metod navrhuje řešení a realizuje aplikaci na mobilní zařízení s platformou Android.

Klíčová slova mobilní aplikace, OS Android, podobnostní vyhledávání, klavírní skladby, porovnávání, zvuková stopa

Abstract

Thesis deals with the issues of similarity searching of piano pieces and comparing the audio tracks. It describes the way to obtain fingerprints from audio recordings and techniques for their comparison. On the basis of current methods thesis proposes solutions and implements application on mobile devices running Android.

Keywords mobile application, OS Android, similarity search, piano compositions, comparison, audio track

Obsah

1 Úvod do problematiky **3**
- 1.1 Obecný postup řešení . . . 6
- 1.2 Základní zvukové charakteristiky . . . 7
- 1.3 Zpracování zvukové stopy . . . 8
- 1.4 Deskriptory . . . 9
- 1.5 Metody porovnávání . . . 15
- 1.6 Metody vyhledávání . . . 18

2 Analýza **21**
- 2.1 Přehled dostupných aplikací . . . 21
- 2.2 Analýza knihoven . . . 25
- 2.3 Požadavky . . . 28
- 2.4 Persony . . . 28

3 Návrh řešení **31**
- 3.1 Výběr deskriptorů a porovnávací metody . . . 31
- 3.2 Platforma . . . 33
- 3.3 Architektura . . . 34
- 3.4 Model tříd . . . 34

4 Realizace **37**
- 4.1 Volba nástrojů . . . 37
- 4.2 Použité knihovny třetích stran . . . 38
- 4.3 Specifika platformy . . . 39
- 4.4 Struktura projektu . . . 44
- 4.5 Implementace . . . 44
- 4.6 Výsledek aplikace . . . 53

5 Testování **55**
- 5.1 Testování relevantnosti . . . 55

5.2 Testování rychlosti 57
5.3 Porovnání s existujícími aplikacemi 60

Závěr **61**

Literatura **63**

A Seznam použitých zkratek **69**

B Obsah přiloženého CD **71**

Seznam obrázků

1.1 Postup vyhodnocování podobnosti 6
1.2 Zpracování zvukové stopy 8
1.3 Standard MPEG-7 10
1.4 Zvuková obálka 12
1.5 Spektrogram hladiny zvuku 12
1.6 Tempogram 14
1.7 Znázornění onsetu 14
1.8 Onsety a jejich síly 14
1.9 L_p metriky 16
1.10 Metoda dynamického borcení časové osy 17
1.11 Rozsahový dotaz a k - nejbližších sousedů 19

2.1 Musipedia 22
2.2 Musica Piano 23
2.3 Bestclassicaltunes 23
2.4 MelodyCatcher 24
2.5 Ludvík 28
2.6 Marie 29
2.7 Eda 29
2.8 Markéta 29

3.1 Znázornění sekvence výšek 32
3.2 Podíl verzí systému Android 33
3.3 Architektura MVP 34
3.4 Model tříd 35
3.5 Třída Database z balíčku Model 35

4.1 Náhledy aktivit 41
4.2 Zdroje (Resources) 42
4.3 Struktura projektu 44
4.4 Abstraktní třída Descriptors s entitami 48

4.5 Výsledek aplikace 53

5.1 Graf vyhodnocení podobnosti pro skladbu Minuet in G 56
5.2 Graf celkového vyhodnocení metod 56
5.3 Graf testování rychlosti vyhledávání 59

Seznam tabulek

1.1 Přehled deskriptorů 12

2.1 Přehled aplikací 24
2.2 Přehled knihoven 27
2.3 Požadavky aplikace 28

4.1 Virtuální zařízení 38
4.2 Reálná zařízení 38

5.1 Nastavení vzorkování 58
5.2 Shrnutí výhod a nevýhod aplikace 60

Předmluva

Jako klavírní hráč a milovník jazzu jsem si velice oblíbil amerického zpěváka a klavíristu Raye Charlese. Do hudby dokázal vdechnout svůj osobitý styl a přinesl do jazzu mnoho nových elementů. Poslouchám jeho hudbu velice často a ani po dlouhé době mě neomrzí. Musím ovšem přiznat, že mívám potřebu poslechnout si něco nového, zároveň však v podobném stylu a se stejnými prvky. V dnešním světě jistě existuje mnoho nadaných klavíristů, kteří by splnili má očekávání a k tomu je jejich tvorba určitě dostupná na některých z dostupných webových kanálů (youtube, iTunes ...). Jediný problém je ten, jak tyto umělce najít. A obzvláště ty méně známé. Třeba se to jednou podaří i díky podobnostnímu vyhledávání, o kterém je tato práce.

Kapitola 1

Úvod do problematiky

Na počátku 60. let minulého století se začíná rozvíjet nové odvětví zvané MIR (Music Information Retrieval). Tato výzkumná činnost se snažila vyvinout nové způsoby, schémata a rozhraní pro vyhledávání v hudbě a zároveň s pomocí internetu zpřístupnit hudební svět široké veřejnosti. Množství hudby je enormní, zejména té digitální. Apple iTunes dnes nabízí přes 43 milionů hudebních děl a do iPodu nahrajeme údajně až 40.000 písní. Přehrání veškerého tohoto obsahu by trvalo zhruba 250 let. Hudba je lehce dostupná každému a získat ji můžeme pouhými několika kliknutími myši, pokud ovšem víme, kam kliknout. A zde narážíme na nejeden problém. Posluchači často nemají dostatek znalostí nebo zkušeností, aby jednoduše našli to, co hledají. Pouhým textovým vyhledáváním často nejsme schopni dostatečně vystihnout podstatu našeho požadavku a někdy to je dokonce nemožné. Internetové vyhledávače tudíž nejsou vždy zrovna tím nejvhodnějším nástrojem. Intuitivnějším způsobem by bylo vyhledávání pomocí vzoru/nahrávky či konkrétní melodie písně. A tímto se také odvětví MIR, potažmo tato práce, zabývá.

Hlavní myšlenka pro vyhledávání v hudbě spočívá v tom, že reprezentujeme zvukovou stopu jako sadu extrahovaných vlastností (otisk). Tyto lze považovat za identifikátory, na kterých můžeme aplikovat vyhledávací mechanizmy jako u vyhledávání textového. Již v roce 1967 se začalo uvažovat o využívání počítačů pro tyto účely, kdy Lincoln nastínil kritéria systému, který by byl schopný indexovat hudební díla [1]. Od té doby se výzkumy snažily najít možnosti, jak efektivně získávat transkripce hudby a pomocí nich vyhledávat. Až v 90. letech došlo k razantnímu oživení odvětví (zejména kvůli vzniku MIDI standardu a množství volně dostupné hudby online) a v roce 1996 McLane uvedl, že následující pokrok ve vyhledávání v hudbě bude směřovat právě k aplikování standardních principů založených na vyhledáváním textovém [2].

V roce 2000 vznikla nezisková organizace *ISMIR*[1], která se zabývá právě problematikou vyhledávání v hudbě. Každoročně pořádá konference (letos v USA), kde se setkávají světoví odborníci a kde sdílejí své nejnovější poznatky. Od roku 2005 je nedílnou součástí konference komunitní framework *MIREX*[2], který si klade za cíl vyhodnocování MIR algoritmů. V poslední době se vyvinulo několik desítek systémů pro vyhledávání v audiu, zajímavý výběr lze najít v přehledu (Typke, 2005 [3]).

Hledání podobností podle vzoru či zařazování skladeb do žánrů však zdaleka nejsou kompletně vyřešené úlohy. Položíme-li si otázku, která hudba je relevantní pro konkrétního člověka zjistíme, že záleží na mnoha faktorech jako je vkus, zkušenost, emocionální stav, aktivita a zároveň jeho kulturní, sociální či hudební pozadí. Kvůli tomu se problém se stává netriviálním a nalezení optimálního řešení je stále velmi složité a aktuální téma. A přestože jsou v současné době metodiky a systémy pro hudební analýzu na vysoké úrovni, stejně zatím nedokáží plně nahradit trénované lidské ucho či lidské vnímání. Každopádně pro běžného hudebního nadšence, který nemá vytrénovaný hudební sluch, může být jakýkoli softwarový nástroj či aplikace dobrým pomocníkem při jeho začátcích.

A to je také jeden z důvodů, proč se v této práci zabýváme tématem podobnostního vyhledávání, resp. porovnávání, klavírních skladeb. V oblasti vyhledávání hudby (zejména té populární) se v současné době těší velké popularitě služby jako Shazam či SoundHound. Stávají se vynikajícími pomocníky v případě, kdy jste si zapamatovali určitou melodii či část písně, ale nemůžete si vzpomenout na její název či interpreta. K tomu vám poslouží právě tyto aplikace, které denně využívají miliony uživatelů. Co se však týče vyhledávání klavírních skladeb, v současnosti existuje jen několik málo služeb, které se tímto zabývají. K tomu jsou mnohdy zastaralé a uživatelsky nepřívětivé.

[1] International Society for Music Information Retrieval, http://www.ismir.net/
[2] Music Information Retrieval Evaluation eXchange

Práce je strukturována do 5 kapitol. Jednotlivé kapitoly postupně uvádějí poznatky, které jsou důležité pro celkové pochopení této práce.

Kapitola Úvod do problematiky vysvětluje základní principy analyzování zvukových stop a popisuje obecný způsob, jak lze postupovat pro řešení našeho úkolu. Teoreticky rozebírá zpracování samotného signálu, extrakci deskriptorů až po metody porovnávání a podobnostní funkce.

Kapitola Analýza podává přehled o existujících aplikacích, navrhuje dostupné knihovny pro zpracování a extrakci dat ze zvukových stop. Uvádí funkční a nefunkční požadavky aplikace a vytvořené persony.

Kapitola Návrh řešení se zaměřuje na konkrétní výběr deskriptorů a porovnávacích metod, vysvětluje volbu platformy, architektury a zobrazuje návrh tříd a uživatelského rozhraní aplikace.

Kapitola Realizace dokumentuje implementaci a nastiňuje podrobnosti týkající se volby nástrojů, využívání knihoven třetích stran, krátce seznamuje se specifikací platformy a technologií a v závěru popisuje implementační detaily aplikace. Samozřejmě včetně ukázek kódu.

Kapitola Testování se zaměřuje na testování aplikace ze 2 hledisek. A to testování kvality vyhledávání a testování rychlosti. Popisuje cíle, metodiku a zobrazuje výsledky. Úzce souvisí s kapitolou *Analýza a návrh*, jelikož tato část ovlivňovala finální návrh implementace.

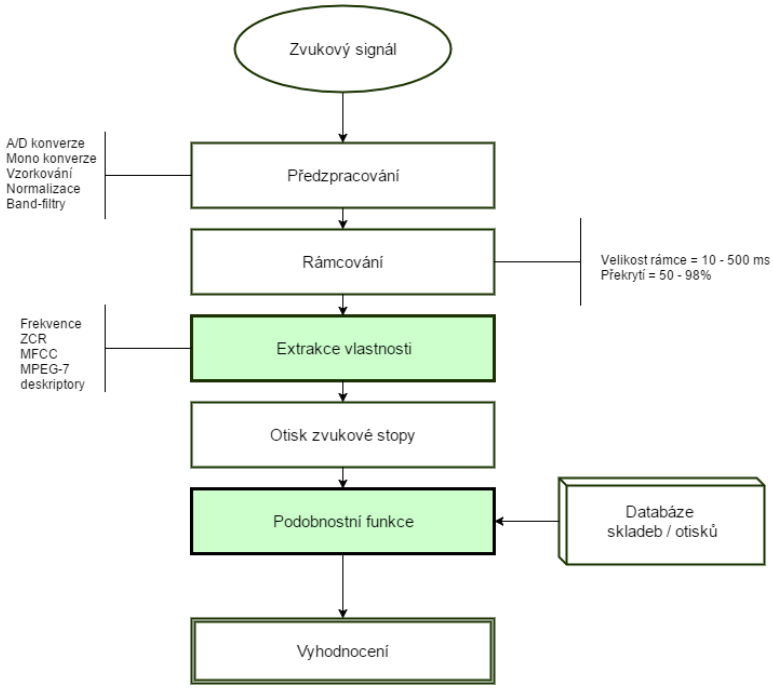
Cílem práce je vytvoření mobilní aplikace na platformě Android, která bude schopna uživateli ohodnotit jeho klavírní hráčské schopnosti. Princip aplikace spočívá v porovnávání záznamu zvukové stopy od uživatele (nahrávka klavírní skladby) s databází skladeb. Aplikace implementuje 2 moduly. Prvním je podobnostní vyhledávání zvukových stop a druhým je porovnávání nahrávky s předem zvolenou skladbou. Pianista nahraje skrze své zařízení (telefon či tablet) svou nahrávku, ze které se vyextrahují potřebná data. V modulu pro podobnostní vyhledávání ji aplikace porovná s databází skladeb a navrhne uživateli podobné včetně procentuálního vyčíslení podobnosti. V druhém modulu uživatel nejprve vybere požadovanou skladbu a právě s ní se nahrávka porovná.

1.1 Obecný postup řešení

Nyní si vysvětlíme zjednodušený postup, jak porovnávání, potažmo vyhledávání, zvukových stop řešit a popíšeme problémy, se kterými se při řešení našeho úkolu setkáme.

Prvním krokem je **extrakce vlastností** (tzv. feature vectors) ze zvukové stopy pomocí vhodných postupů. Těmito vlastnostmi mohou být například tempo, sekvence výšek, celková délka a mnohé další. Druhým krokem je vytvoření algoritmu - **podobnostní funkce** pro porovnání extrahovaných dat a získání míry podobnosti. Ten v nejtriviálnějším případě lineárně projde a porovná otisk vzoru s celou databází a vrátí nám nějakou hodnotu.

Přesný postup se samozřejmě odvíjí od konkrétní úlohy. Hledáme-li například množinu nejpodobnějších písní (náš případ), musíme získat všechny hodnoty podobností mezi dotazovanou skladbou a libovolnou skladbou z databáze a následně je seřadit. V případě zařazování písní do žánrů je nutné otisky napřed seskupit podle určitých kritérií a poté nasadit vhodné způsoby (např. statistické modely) pro porovnání.



Obrázek 1.1: Postup vyhodnocování podobnosti

Celý proces je znázorněn na obr. 1.1. Úplně posledním krokem, který na obrázku není zobrazený, je často finální testování relevantnosti podobnostních funkcí a porovnávacích metod. A to takové, že se výsledky z vyhodnocení porovnají s ideálním řešením, tj. ohodnocením lidským. Počítačové řešení totiž obvykle nedosahuje perfektních výsledků jako je tomu v mnoha jiných oblas-

tech. V následujících sekcích si podrobněji vysvětlíme jednotlivé kroky, o kterých jsme se zde zmínili. Začneme zvukem jako takovým.

1.2 Základní zvukové charakteristiky

Většina metodik a přístupů pro vyhodnocování hudby zakládá na konceptech mezi něž patří analýza primárních zvukových charakteristik. Takřka veškeré hudební nástroje (včetně lidského hlasu) vydávají při hraní periodické vibrace - **tóny** a výsledný zvuk lze proto chápat jako kombinaci určitých frekvencí, které jsou celočíselými násobky fundamentální frekvence zvané *F0*.

Vedle tónů existují i zvuky, které nemají pravidelný frekvenční průběh ani základní frekvenci. Ty vydávají nástroje perkusní a označujeme je jako **hluky**. U nich však můžeme stále sledovat jejich další vlastnosti jako hlasitost a zabarvení.

Základními atributy pro popis zvuku jsou (Klapuri, 2006 [4]):

- **Výška**

  Sluchový vjem, který nám umožňuje intuitivně škálovat tóny na hudební stupnici. Výška je úzce spjata s veličinou **frekvence**, ovšem tyto pojmy nejsou zcela identické. Frekvence je totiž fyzikálně měřitelný atribut, přičemž výška je subjektivní a psychoakutický[3] popis zvuku. Pro zjednodušení můžeme však tyto pojmy zaměnit a výšku definujeme jako frekvenci sinusové křivky odpovídající zvukovému signálu. S rostoucí frekvencí roste také jeho výška a naopak.

- **Hlasitost**

  Charakteristika zvuku související s její silou (energie vibrace). Opět se jedná jako u výšky o subjektivní veličinu, která závisí mimo jiné na individuální citlivosti sluchu. Ekvivalentem pro objektivní hodnocení je **intenzita zvuku**. V počítačovém zpracování vyjádříme hlasitost jako druhou mocninu střední kvadratické hodnoty signálu převedenou do logaritmické stupnice (pro snížení dynamického rozpětí zvuku). Jednotkou hlasitosti jsou decibely.

- **Zabarvení**

  Zvuková charakteristika někdy označovaná jako *témbr*, která umožňuje posluchačům rozeznat 2 různé zdroje zvuků. Například zvuk klavíru a kytary lehce rozeznáme díky svému zabarvení i přestože mohou vydávat tóny o stejné výšce a hlasitosti. Odlišíme zvuky znějící z malého uzavřeného pokoje se zvuky z koncertní haly. Zda-li zdroj pochází z digitální nahrávky nebo z živého vystoupení. Tato vlastnost charakterizuje zvuk

[3] Psychoakustika - věda zabývající se vnímáním zvuků

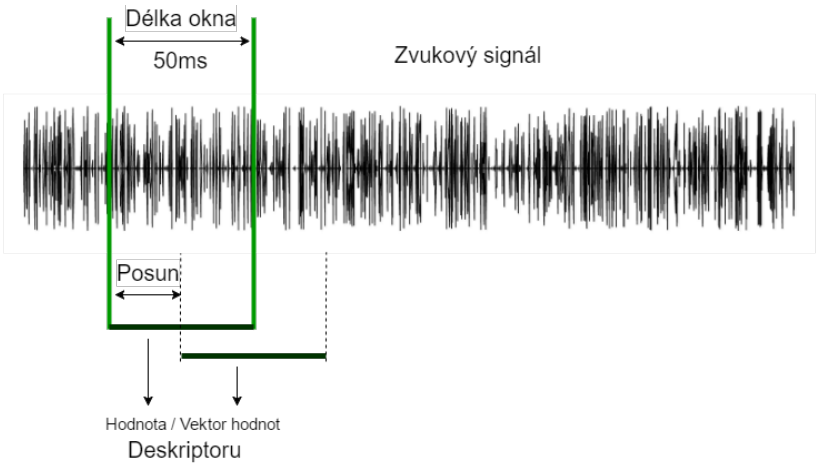
komplexněji a nelze ji tudíž uchopit a vyjádřit jako konkrétní fyzikální veličinu.

Souvisí spíše s rozložením spektrální energie a její distribucí v čase. Zatímco předchozí charakteristiky zvuku vyjádříme skalárními hodnotami (výška - frekvence - Hz, hlasitost - intenzita - W/m^2), zabarvení je vícerozměrný koncept a reprezentujeme ho jako vektor charakteristik.

S těmito vlastnostmi se v průběhu práce ještě mnohokrát setkáme, a proto je nezbytné znát alespoň tento základ. Podrobnější hudební terminologii si dovolíme přeskočit, jelikož ji v této práci nevyužijeme. Budeme směřovat spíše k rovině počítačové, i přestože se jedná o téma z části umělecké či hudební. V další kapitole se proto podíváme na způsob, jak zvukovou stopu zpracovat a následně analyzovat.

1.3 Zpracování zvukové stopy

Zvukovou stopu / nahrávku od uživatele neporovnáváme ve své původní podobě, nýbrž signál nejprve převedeme do podoby (otisku), se kterou lze lépe pracovat. Tento proces nazveme dekódování a probíhá následujícím způsobem:



Obrázek 1.2: Zpracování zvukové stopy

Nahrávka (WAV, MP3, AAC) se dekomprimuje a transformuje do diskrétní časové domény. V této podobě se signál převede do posloupnosti vzorků (sample) s uniformním časovým skokem (délka okna). Počet těchto vzorků za sekundu se nazývá vzorkovací frekvence. V běžné praxi se setkáme se vzorkováním 22050 Hz - 44 100 Hz [5]. V jednotlivých vzorcích lze následně získat určitý popis (deskriptor), resp. jeho hodnotu či vektor hodnot, které nám poslouží pro následné porovnávání. Z každé skladby takto získáme otisky, které

si uložíme a které bude podobnostní funkce porovnávat s otiskem nahrávky od uživatele.

V následující kapitole si ukážeme, jakou sadu deskriptorů můžeme ze zvukových stop získat.

1.4 Deskriptory

V této sekci uvedeme některé příklady zvukových deskriptorů a jejich možnosti využití. Samozřejmě jich existuje celá řada, pro naše účely si vyjmenujeme jen ty nejelementárnější. S deskriptory se setkáme převážně v oblasti analýzy zvuku. Jedná se o kolekci „popisků“, které interpretují zvukovou stopu v mnoha různorodých a zároveň strojově použitelnějších formách. Pokud chceme zvukový signál uchopit a dále s ním manipulovat, je ve většině případech nezbytné deskriptory ze zvukové stopy vyextrahovat.

Specializované nástroje a knihovny uvedené v kapitole 2.2 nám umožní získat velice bohatou množinu různorodých deskriptorů. Rozdělíme je do 5 základních kategorií, a to základní, spektrální, časová doména, melodické a rytmické.

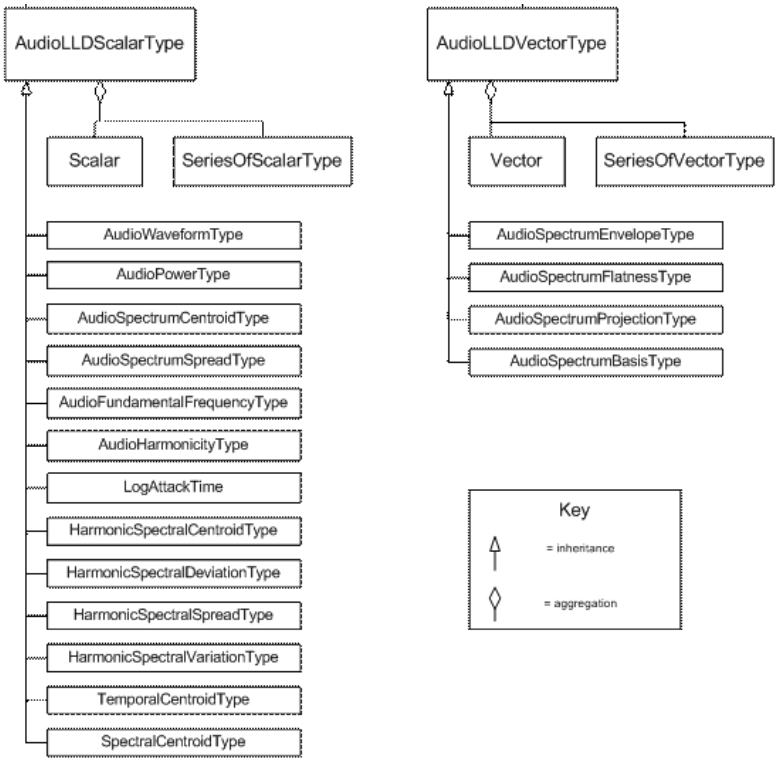
Pro konkrétní účely analýzy a zpracování je podstatné zvážit výběr co nejvhodnějších, zpracování celé sady by bylo velmi neefektivní a drahé. V našem případě je například zbytečné uvažovat deskriptory perkusní, hlasitosti či komplexnější spektrální. Konkrétním výběrem pro naši aplikaci se budeme zabývat později.

1.4.1 MPEG-7

V praxi se pro účely získávání deskriptorů ujal standard MPEG-7 (Multimedia Content Description Interface) [6]. Byl vytvořený komisí MPEG (Moving Picture Experts Group) v roce 1996 a nabízí širokou škálu standardizovaných nástrojů pro popis multimediálního obsahu. Lze ho využít pro získání popisných dat jak ze zvukových souborů, tak i obrázků či videí.

Standard MPEG-7 obsahuje 17 nízkoúrovňových a 5 vysokoúrovňových deskriptorů. Některé z nich si později popíšeme. Celkový přehled na obrázku 1.3.

Nevýhodou tohoto standardu je absence několika velice důležitých deskriptorů, např. výšek, hlasitosti, zero-crossing-rate (průchod nulou) a dalších spektrálních charakteristik. Některé z těchto nedostatků se snaží napravit takzvané MFCC.



Obrázek 1.3: Deskriptory standardu MPEG-7 [7]

1.4.2 Kepstrum a kepstrální příznaky (MFCC)

Kolekce deskriptorů, které se v oblasti MIR využívají možná nejčastěji, jsou **Melovské kepstrální koeficienty (Mel-frequency cepstrum)**, tzv. **MFCC**. Byly uvedeny (Davis a Mermelstein, 1980 [8]) jako efektivní nástroj zejména pro rozpoznávání lidské řeči. Jejich význam spočívá v interpretaci celkové zvukové barevnosti (timbré), které lidské vnímání nejpřirozeněji identifikuje. Proto se v současné době MFCC deskriptory aplikují i v mnoha jiných disciplínách.

Postup pro získání koeficientů [9]

1. Vzorkování signálu do rámců (v řádech ms) a rozdělení do intervalových oken (např. Hanningovo okno)
2. Výpočet FFT[4] pro každý rámec
3. Výpočet spektrálního výkonu a rozdělení do pásem - Využití Melovy stupnice

[4]Fast Fourier Transformation - algoritmus pro spočtení diskrétní Fourierovy transformace a její inverze

4. Logaritmus výkonu v každém pásmu
5. Zpětná Fourierova transformace (pomocí diskrétní kosinové transformace)
6. Výsledkem jsou kepstrální koeficienty (nejčastěji se využívá prvních 13)

MFCC deskriptory nejsou součástí standardu MPEG-7, existuje proto již řada studií založená na porovnávání nízkoúrovňových MPEG-7 deskriptorů a MFCC [10, 11]. Výsledky ovšem nepotvrzují žádné jednotné stanovisko, které z nich jsou lepší. Záleží spíše na charakteru úkolu a v současnosti se využívají obě možnosti.

1.4.3 Přehled deskriptorů

Kategorie	Deskriptory
Základní	Zvuková obálka, Zvukový výkon
Spektrální	MFCC, Spektrální obálka, Spektrální těžiště a rozptyl
Časová doména	Log Attack Time, Zero-crossing rate, Hlasitost, Délka
Melodické	Výška, Klíč
Rytmické	BPM, Onset, Síla onsetu

Tabulka 1.1: Přehled používaných deskriptorů

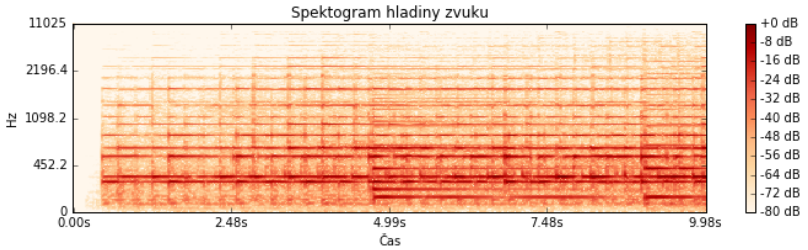
■ Základní (podle standardu MPEG-7)

Základní možností, jak obecně popsat zvukový signál, je získat `AudioWaveform - zvukovou obálku`. Změříme kladnou a zápornou amplitudu v nepřekrývajících se po sobě jdoucích rámcích a dostáváme tak 2 hodnoty pro každý rámec. Výsledkem je série párů (min, max), znázornění na obr. 1.4.

Obrázek 1.4: Zvuková obálka (waveform)

`AudioPower - zvukový výkon` je deskriptor spojený s hlasitostí, který popisuje sílu zvukového signálu v časové doméně - v logaritmické škále (dB). Slouží k rozeznání hlasitých a tichých zvuků.

,



Obrázek 1.5: Spektrogram hladiny zvuku

■ Spektrální

Do této kategorie spadají deskriptory popisující spektrální obsah signálu a berou v potaz logaritmickou škálu lidského vnímání zvuku [7].

`MFCC (Mel-kepstrální koeficienty)` komplexně reprezentují zvukové spektrum, přičemž nezohledňují základní frekvenci a jsou robustní vůči šumu. Jsou indikátorem zabarvení, v praxi to znamená, že získáme vektory charakteristik. Prakticky je lze využít na jakoukoli úlohu od rozpoznávání nástrojů, umělců až po zařazování hudby do žánrů. Způsob jejich extrakce jsme si uvedli v minulé sekci.

`Spectral Envelope (Spektrální obálka)` popisuje logaritmické spektrum signálu. Pomocí něj můžeme například vytvořit jednoduchý spektrogram.

`Spectral Centroid (Spektrální těžiště)` souvisí s obálkou a udává jeho centrum. Můžeme následně vyčíst, zda-li se silové spektrum nachází spíše v nízkých či vysokých frekvencích.

`Spectral Spread (Spektrální rozptyl)` nám ukáže, jestli se audio signál v průběhu času vzdaluje od těžiště a jakým způsobem se rozprostírá ve frekvenčním pásmu.

■ Časová doména

`Log Attack Time (Časový útok)` je čas, který trvá signálu přejít od nehlučnosti do jeho maximální amplitudy. Poukazuje na rozdílnost mezi prudkou a mírnou změnou zvuku. Využívá se pro získávání Onsetů, o nich si povíme v sekci *Rytmické*.

`Zero-crossing rate (Průchod nulou)` udává počet průchodů, kdy signál nabývá nulové hodnoty, tzn. přechodů z kladných do záporných hodnot a naopak. Hodnota ZCR poslouží k rozlišení periodických zvuků (nízká hodnota) a hluků (vysoká). Teoreticky ho lze u monofonních zvuků využít i k primitivní detekci výšek.

Dalšími deskriptory jsou například `Hlasitost` a `Délka trvání`. Obě vrací určitou skalární hodnotu a jejich význam asi není potřeba vysvětlovat.

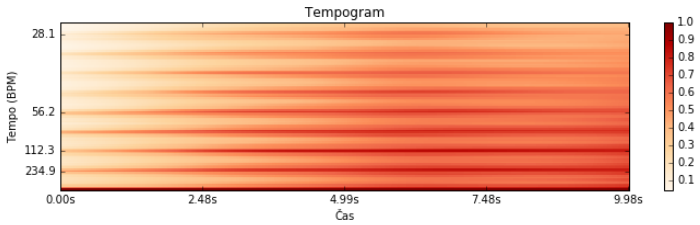
■ Melodické

`Výška`, o tomto atributu jsme se již bavili v úvodu, jedná se o elementární atribut spojený s frekvencí signálu. S rostoucí frekvencí roste také výška. Reprezentujeme ji skalární hodnotou a slouží nám k rozpoznání melodie.

`Klíč` udává informaci o melodické stupnici.

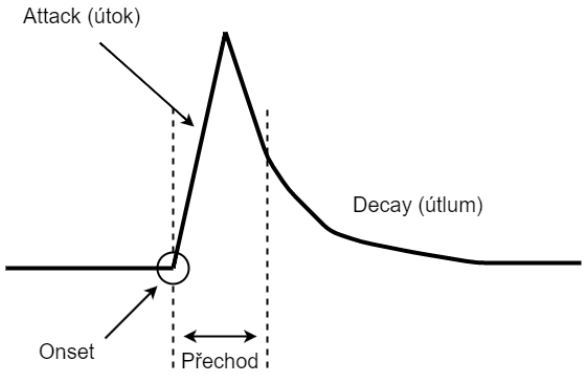
■ Rytmické

`BPM (údery za minutu)` označuje tempo (rychlost) hudby. Úderem rozumíme čtvrťovou notu, exaktně BPM udává počet čtvrťových not za minutu. 60 BPM odpovídá jednomu úderu za sekundu. Konkrétní rozsahy mají i své názvy pocházející z italštiny, např. *Largo* odpovídá 40-60 BPM, *Andante* 76-108 BPM a *Allegro* 120-168 BPM.



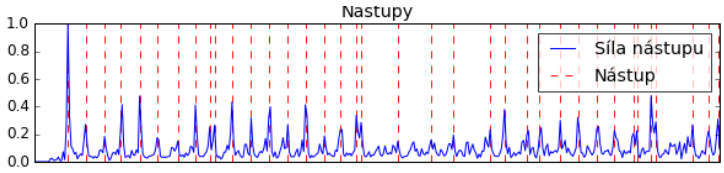
Obrázek 1.6: Tempogram

`Onsety (nástupy)` se využívají zejména pro analýzu hudby obsahující perkusní nástroje. Jedná se o místo nového úderu, resp. místo, kde začíná nový přechod [12]. Znázornění na obr. 1.7.



Obrázek 1.7: Znázornění onsetu

`Síla onsetu` udává amplitudu v místě onsetu, viz. 1.8. Tento deskriptor se často využívá pro rozeznání důrazů a struktury v hudbě.



Obrázek 1.8: Onsety a jejich síly

1.5 Metody porovnávání

Předchozí kapitoly ukázaly, jaké vlastnosti můžeme získat ze zvukových zdrojů. Podobnost je číselná hodnota a určíme ji na základě porovnání jedné, či několika vlastností dohromady. Pro porovnávání hudby mohou být těmito atributy například tempo, výšky/frekvence, délka a klíč. Nyní si zvolíme způsob, kterým budeme vlastnosti porovnávat. Podle Nicola Orio, 2006 [13] rozdělujeme metodiky do tří základních kategorií.

1.5.1 Metodiky

- **Porovnávání na základě termů**

  Tato metoda spočívá v extrahování krátkých „termů/segmentů“ z melodie, které jsou efektivními ukazateli pro popis konkrétní skladby. V případě textu bychom tyto termy mohli brát jako jednotlivá slova. Ovšem na rozdíl od věty, kde slova oddělujeme mezerami, je melodie souvislým proudem informací, který nelze tak jednoduchým způsobem rozčlenit. Jak tedy postupovat?

  Jeden z přístupů je založen na rozdělení na úseky o fixní délce N-not, tzv. N-gramy. Tyto N-gramy se navzájem překrývají a to právě z důvodu neschopnosti určit začátek a konec relevantní části. Celkový počet shodných N-gramů v porovnávaných melodiích je následně měřítkem podobnosti.

  Tato metodika je v oblasti MIR velmi často využívána a lze ji aplikovat i na polyfonní skladby [14]. Experimentální výsledky na kolekci 2300 dokumentů ve formátu Midi ukázaly, že N-gramy jsou v stále tím nejlepším možným způsobem pro porovnávání termů vykazující 98% přesnost. Segmentace ovlivněné předchozí znalostí struktury či jinými hudebními aspekty jsou náchylné k lokálním chybám a průměrná přesnost dosahovala 85%.

- **Porovnávání sekvencí**

  Při porovnávání sekvencí reprezentujeme melodii jako sekvenci not. Porovnání je následně možné provést mnoha různými způsoby. Jedním z nich je například Levenshteinova vzdálenost (1.5.2) využívající se při porovnávání textu, dalším je metoda dynamického borcení časové osy (1.5.2).

  První využití uvedli Ghias et al., 1995 [15], kde melodii reprezentovali jako sekvenci pouhých tří symbolů - a to zvýšení, snížení a shodnost předcházející noty. Nazýváme je také jako *Parsons Code* [16].

  Porovnávání sekvencí se často využívá i v oblasti bioinformatiky, kde se setkáme s přístupem založeným na vyhledávání konkrétní sekvence v celé kolekci (pattern discovery). Tento přístup aplikovali Bainbridge et

al., 1999 [17], kde je vzorem zjednodušená a krátká kontura not/výšek vyhledávaná uvnitř skladeb.

- **Geometrické porovnávání**

  Přístup využijeme zejména v oblasti porovnávání polyfonních melodií. Dokument reprezentujeme jako kolekci bodů či úseček v rovině, přičemž tyto body mohou mít navíc svou váhu. Horizontální osa značí obvykle časový průběh a vertikální označuje výšku/frekvenci. Konkrétní bod tedy nese kompletní informaci o výšce, času nástupu a jeho váha je spočtena z jeho délky trvání.

  Porovnání se spočítá například pomocí metriky Earth Movers Distance určující cenu převodu jednoho histogramu na druhý [18].

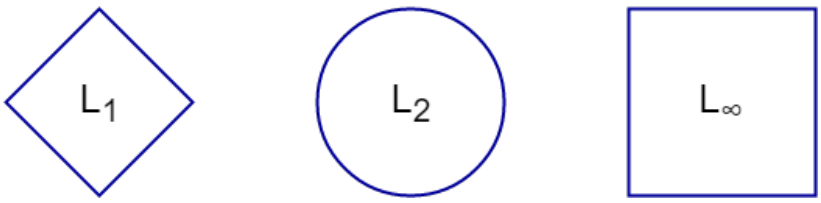
1.5.2 Vzdálenostní funkce

Minkowského vzdálenosti

Zvané také jako L_p metriky zastřešují celou rodinu metrických funkcí. Jsou definovány na n-dimensionálních vektorech reálných čísel:

$$L_p[(x_1, ..., x_n), (y_1, ..., y_n)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{1.1}$$

L_1 metrika se nazývá **Mannhattanská vzdálenost**, L_2 značí obecně známou **Euclidovskou vzdálenost** a L_∞ odpovídá **Maximální vzdálenosti**. L_p metriky jsou vhodné například pro porovnávání vektorů charakteristik a často se využívají pro porovnávání obrázků. V této práci se s druhou zmíněnou metrikou ještě setkáme.



Obrázek 1.9: L_p metriky

Levenshteinova vzdálenost

Metrika, zvaná také jako editační vzdálenost, udává minimální počet znakových operací potřebných k převodu jednoho textového řetězce (v našem případě např. řetězce not) na druhý. Operacemi jsou nahrazení, vložení nebo vypuštění libovolného znaku v řetězci, přičemž pro splnění pravidla o symetrii musí mít každá z nich shodnou cenu. Například vzdálenost mezi slovy

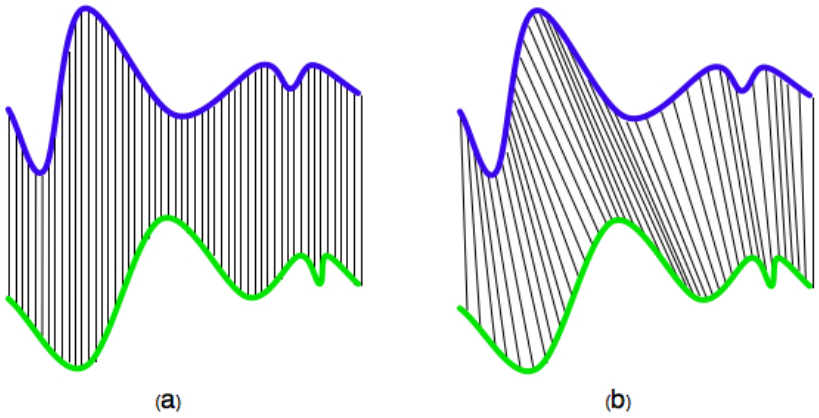
„PROGTEST“ a „PROTESTY“ je 2 (vypuštění G, a přidání Y).

Dynamické borcení časové osy (DTW)

Metoda *DTW (Dynamic Time Warping)* umožňuje rozpoznat shodné sekvence, i pokud jsou odlišně roztaženy v časové ose [19]. Důležitou vlastností je schopnost vyrovnat se i s nelineárním roztažením. Porovnáváme-li dvě časové série pomocí euklidovské metriky, výsledek je roven součtu vzdáleností mezi jednotlivými n-tými body jedné série a n-tými body druhé. Hlavní nevýhodou euklidovské metriky je ta, že je velmi neintuitivní. Pokud máme 2 identické časové série, ovšem jedna je jen posunutá v čase, metrika je vyhodnotí jako velmi odlišné. DTW tento problém odstraňuje a získáváme tak intuitivnější výsledky i s ohledem na globální, tak lokální posuny v čase.

Původně bylo DTW využíváno zejména pro rozpoznávání mluvené řeči. Dnes se tento způsob využívá i v oblasti data-miningu a MIR, jehož zastoupení nacházíme v úlohách, kde je nutné vyrovnat se s časovými deformacemi a různorodostí spojenou s časově závislými daty.

Algoritmus postupně prochází porovnávané sekvence a dynamicky „bortí“ jejich časové osy tak, aby ve výsledku měly tyto sekvence lépe zarovnaná minima a maxima (jednoduše tvar). Příklad vidíme na obr. 1.10.



Obrázek 1.10: a) Euclidovská vzdálenost b) Metoda DTW

Výpočet vzdálenosti dvou sekvencí určíme následovně. Jsou dány sekvence $\vec{x} = (x_1, \ldots, x_n)$ a $\vec{y} = (y_1, \ldots, y_m)$. Vytvoříme matici A typu m/n, kde prvky matice $(a_{ij})_{i=1,\ldots,m,j=1,\ldots,n}$ odpovídají vzdálenostem $\delta(x_i, y_j)$ mezi prvky x_i a y_j daných sekvencí. Tyto vzdálenosti můžeme vypočítat pomocí libovolné metriky, lze použít např. Manhattanskou vzdálenost (viz. 1.5.2) :

$$\delta(x_i, y_j) = |x_i - y_j|$$

Nyní hledáme „cestu“ maticí A, která určuje mapování prvků jedné posloupnosti na prvky posloupnosti druhé. Tímto průchodem matice získáme

určitou posloupnost hodnot - vektor $\vec{W} = (w_1, \ldots, w_K)$, jehož složkami jsou vzdálenosti na sebe namapovaných prvků

$$w_k = a_{ij}, \quad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}.$$

Průchod maticí A hledáme následujícím způsobem:

- $w_1 = a_{11}$ a $w_K = a_{mn}$. Průchod začíná v jednom rohu matice, a končí v protějším. Začátky a konce porovnávaných sekvencí jsou mapovány na sebe.
- Cesta je spojitá a „nevrací“ se v žádné ze sekvencí, což znamená, že omezíme výběr možných kroků v průchodu. Pro každý prvek $w_k = a_{ij}$ a následující $w_{k+1} = a_{i'j'}$ musí platit jedna z podmínek:
  - $i' = i + 1 \ \wedge \ j' = j$
  - $i' = i \quad \wedge \ j' = j + 1$
  - $i' = i + 1 \ \wedge \ j' = j + 1$

1.6 Metody vyhledávání

Pro podobnostní vyhledávání se využívají *podobnostní dotazy*. Nejčastěji se setkáme se třemi druhy, a to s dotazy na rozsah, nejbližšího souseda a k - nejbližších sousedů. V této práci budeme implementovat k - nejbližších sousedů a to sekvenčně na celé databázi (označujme ji jako $\mathcal{D}$ (universum)).

1.6.1 Rozsahový dotaz

Pravděpodobně nejpoužívanější typ dotazu (range query), který slouží k nalezení množiny objektů, které jsou od dotazovaného vzdálené o daný práh. Rozsahový dotaz $R(o, r)$ je určen dotazovaným objektem o a poloměrem(vzdáleností) r. Formálně:

$$R(o, r) = \{x \in \mathcal{D} : d(o, x) \leq r\} \tag{1.2}$$

Příkladem může být dotaz: Nalezni všechny skladby, které mají tempo nižší či vyšší maximálně o 10 úderů za minutu než má nahrávka.

1.6.2 Nejbližší soused

Z anglického překladu nearest neighbor. Vrací prvek, který je nejblíže dotazovanému (jeho nejbližším sousedem).

$$NN(o) = \{x \in \mathcal{D} : \forall y \in \mathcal{D}, d(o, x) \leq d(o, y)\} \tag{1.3}$$

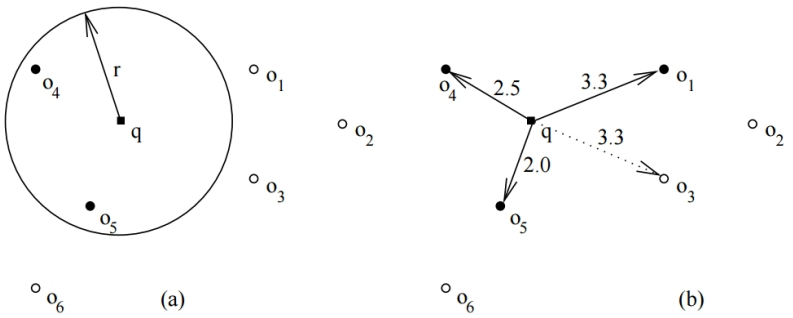
Příklad: Nalezni skladbu, která má tempo nejpodobnější nahrávce.

1.6.3 K - nejbližších sousedů

Poslední typ je pouze speciálním případem předchozího. Umožňuje nám získat k nejbližších sousedů k dotazovanému prvku, tj. množinu $A \in \mathcal{D}$ takovou, že $|A| = k$

$$k - NN(o) = \forall x \in A, y \in \mathcal{D} - A : d(o, x) \leq d(o, y) \tag{1.4}$$

Příklad: Nalezni 5 nejpodobnějších skladeb k tvé nahrávce (náš případ).



Obrázek 1.11: a) Rozsahový dotaz $R(q, r)$ b) k-nejbližších sousedů $3 - NN(q)$. Vyhovující prvky jsou vyznačené tučně. [20]

Kapitola 2

Analýza

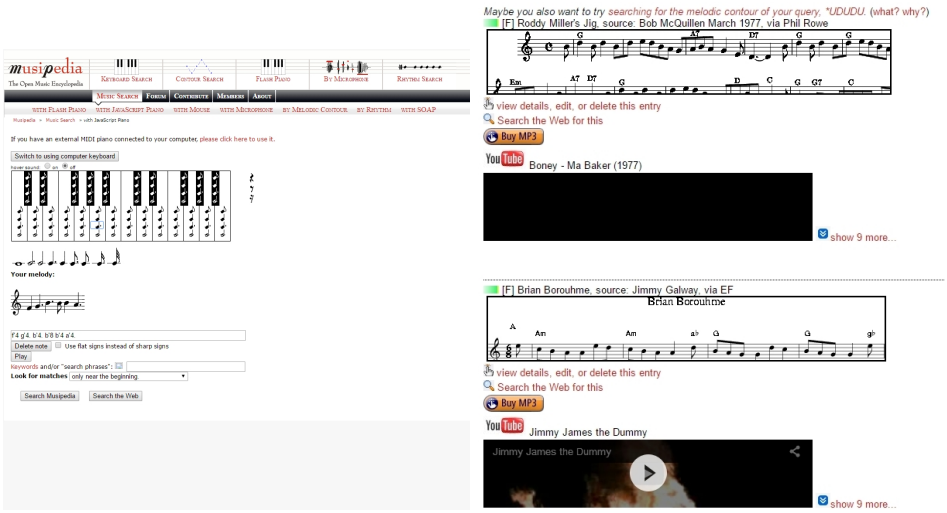
Nyní se podíváme na samotnou analýzu a návrh. Nejprve prozkoumáme existující aplikace pro podobnostní vyhledávání klavírních (obecně zvukových) skladeb, dále si představíme knihovny, které budeme moci využít k získání audio otisků (deskriptorů) a vysvětlíme jejich význam. V závěru se zaměříme na konkrétní řešení z hlediska výběru knihoven, deskriptorů a podobnostních funkcí.

2.1 Přehled dostupných aplikací

V oblasti vyhledávání audia (obecně) se můžeme setkat s mnoha aplikacemi napříč všemi platformami. Nejpopulárnějšími jsou zejména Shazam a Soundhound, které poslouží v případě, kdy vám v hlavě zní melodie písně, nicméně si nemůžete vzpomenout na její název či interpreta. Zazpíváte ji do mikrofonu a během několika sekund aplikace poskytne kompletní informace o titulu. Vývojář Shazamu Wang, 2003 [21] zveřejnil metodu, jakým se zvukové stopy porovnávají. Algoritmus v databázi vyhledává otisk vaší nahrávky ve formě spektogramu. Obdobných aplikací je velké množství, dalšími příklady jsou TrackID, ACRCloud, musiXmatch nebo Sound Search od Googlu.

Co se však týče **podobnostního vyhledávání klavírních skladeb**, situace již není tak příznivá. Na trhu je dostupných pouze několik aplikací, které mají převážně zastaralé webové rozhraní a v horším případě ani nefungují. Aplikace jsem vyzkoušel, pojďme se tedy podívat na některé blíže.

2.1.1 Musipedia



Obrázek 2.1: Musipedia - vyhledávání a výsledky

Popis: Webová aplikace[5] nabízí širokou škálu možností, jak identifikovat skladbu. Můžete vyťukávat sekvenci tónů na virtuálních klávesách nebo přidávat noty přímo na notovou osnovu (viz. obr. 2.1). Dále je zde možnost nahrání stopy přes mikrofon, ovšem tato funkcionalita mi v době psaní práce nefungovala.

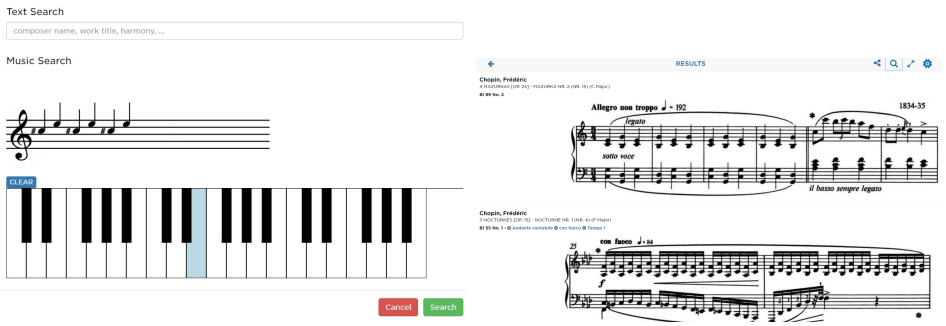
Vyhledávání: Spočívá v hledání odpovídajících *Parsonových kódů*[6], o kterých jsme se zmínili v sekci 1.5.1. Aplikace uvádí stěžejním aspektem výšku a rytmus.

Výstup: Seznam odpovídajících titulů včetně grafického znázornění not a možností přehrání písně ze serveru youtube. Dokonce jsou ke skladbám přidány odkazy na jejich zakoupení ze serveru Amazon.

2.1.2 Musica Piano

Popis: Musica Piano[7] slouží primárně pro hledání not, ovšem můžeme ho využít i jako vyhledávač skladeb. Je implementována na platformě iOS a zároveň jako webová aplikace. Aplikace jsou sice zdarma, ovšem pro pohodlné používání bez vyskakujících oken je potřebné měsíční předplatné ve výši 1 - 9 dolarů.

[5] dostupná na www.musipedia.org
[6] Jednoduchá notace reprezentující melodii pouze pomocí 3 symbolů
[7] app.musicapiano.com

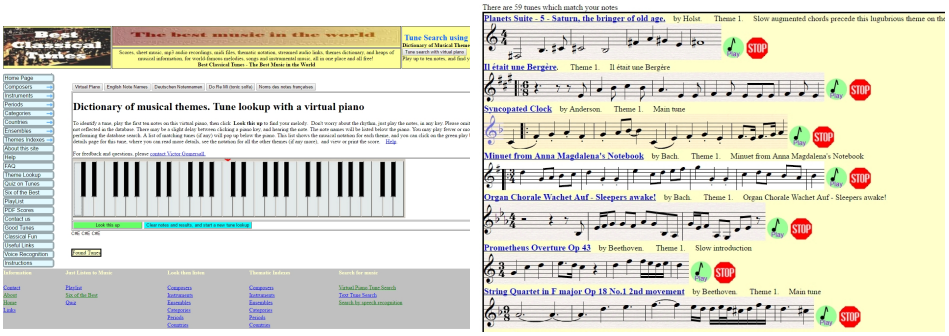


Obrázek 2.2: Musica Piano - vyhledávání a výsledky

Vyhledávání: Při vyhledávání, stejně jako u aplikace Musipedia, zadáváme sekvenci tónů na virtuální klávesy. Hledání můžeme navíc zkombinovat spolu s textem (známe-li např. část názvu, případně skladatele apod.).

Výstup: Opět seznam skladeb s notami (primární účel aplikace), které si můžeme transformovat do různých temp, pokud to skladba umožňuje.

2.1.3 Bestclassicaltunes



Obrázek 2.3: Bestclassicaltunes - vyhledávání a výsledky

Popis: Platforma nabízí různé možnosti vyžití týkající se klasické hudby. Od informací o skladatelích, přes stahování MIDI souborů skladeb, až po samotné vyhledávání. Webové stránky jsou však velmi zastaralé a vyhledávací engine funguje poměrně nerobustně, jelikož po zadání „delší“ sekvence not (4 a více) nenachází žádné výsledky.

Vyhledávání: V uživatelském rozhraní opět vyťukáváte tóny (nezáleží na rytmu). Můžete si je zpětně přehrát či upravovat.

Výstup: Sada odpovídajících skladeb, s možností jejich přehrání a zobrazení notové osnovy. Nebylo zjevné, zda-li jsou výsledky nějakým způsobem seřazené.

2.1.4 MelodyCatcher

Obrázek 2.4: MelodyCatcher - vyhledávání a výsledky

Popis: Vyhledávací engine MelodyCatcher[8] je implementovaný jako Java applet. Kvůli tomu jsem měl potíže se spuštěním v prohlížeči Google Chrome, ovšem na Firefoxu fungovala aplikace bez problému. Aplikace umožňuje vyhledávání přímo přes webové rozhraní a zároveň i přes externí klávesy připojené k počítači.

Vyhledávání: Princip je naprosto totožný z předchozími. Dokumentace uvádí, že pro aplikaci je postačujících prvních 5-7 tónů pro identifikování melodie.

Výstup: Seřazený seznam odpovídajících zvukových stop podle relevantnosti. Aplikace čerpá z různorodých databází a mezi výsledky se velmi často objevují náhodné nehudební nahrávky.

Aplikace	**Platforma**	**Cena**	**Funkcionalita**				
			1	**2**	**3**	**4**	**5**
Musipedia	Web	Zdarma	+	−	−	+	+
Musica Piano	Web + iOS	Předplatné	+	−	+	+	+
Bestclassicaltunes	Web	Zdarma	+	−	−	+	−
MelodyCatcher	Web (Java applet)	Zdarma	+	−	−	−	−

Tabulka 2.1: Přehled aplikací

Funkcionalita

1. Vyhledávání pomocí virtuálních kláves
2. Vyhledávání pomocí mikrofonu
3. Ukládání nahrávek
4. Přehrání skladeb
5. Uživatelská přívětivost

[8] www.melodycather.com

2.2 Analýza knihoven

V následující kapitole uvedu existující knihovny či aplikační rozhraní, které lze využít k extrakci a zpracování dat (deskriptorů) ze zvukových stop. Tabulka 2.2 v závěru ukazuje přehled s jednotlivými funkcionalitami.

ESSENTIA

Open-source C++ knihovna pro zvukovou analýzu a získávání dat. Obsahuje rozsáhlou kolekci používaných algoritmů a mimo jiné i zpracování vstup-výstupních operací, digitální zpracování signálu, statistické charakteristiky získaných dat a množství zvukových deskriptorů. Vyvinuta skupinou MTG na univerzitě Pompeu Fabra v Barceloně [22].

Musly

Musly je vysoce efektivní knihovna v C/C++, která v současné době implementuje 2 porovnávací algoritmy. Lze ji využít buď jako knihovní součást, nebo v prostředí příkazového řádku. Autorem je Dominik Schnitzer [5].

MARSYAS

Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals) je oblíbený framework (C++) pro zpracování zvukových stop vyvíjen Georgem Tzanetakis [23].

LibROSA

LibROSA je balíček v jazyce Python. Poskytuje rozhraní pro vytvoření popisných dat a zvukových otisků. [24]

MPEG-7 Audio Encoder

Tato javovská knihovna poskytuje jednoduché rozhraní pro popis audio obsahu pomocí deskriptorů standardu MPEG-7. Jedná se o proprietární software, který jako vstup přijímá zvukovou stopu, a výstupem je XML dokument obsahující její deskripci.

jAudio + jMIR

Softwarový balíček (Java) pro extrakci dat z audio souborů. Lze ho využít v mnoha oblastech vyhledávání v audiu, v praxi zejména využíván spolu s metodami strojového učení (jMIR).

jMIR je open-source software navržený v jazyce Java pro použití v oblasti MIR. Extrahování dat, algoritmy pro strojové učení a vytěžování metadat jsou také součástí. Nevýhodou je nížší propracovanost dokumentace. [25].

TarsosDSP

Framework poskytující nástroje pro real-time[9] zpracování a analýzu audia. Implementován v jazyce Java, je lehce rozšířitelný a nemá žádné závislosti na jiných knihovnách. Zároveň je kompatibilní i pro vývoj na platformě Android [26].

Echo Nest API

Aplikační serverové rozhraní Echo Nest API umožňuje volání interních metod, které vracejí soubory JSON či XML. Přistupovat lze pomocí HTTP požadavků. Tuto knihovnu používá v komerčním světě mnoho společností, jednou z nich je například Spotify [27].

2.2.1 Výběr knihovny

Při výběru knihoven pro extrakci dat a zpracování nahrávek jsme zohledňovali zejména přehlednost dokumentace, funkcionalitu rozhraní a přívětivost z hlediska procesu naší aplikace. Nabízeli se dva scénáře, a to zpracování přímo na mobilním zařízení (tenký klient) nebo na serveru (tlustý klient). Pro každý z nich byl výběr odlišný.

Podíváme-li se na tabulku 2.2, pro implementaci **tenkého klienta** odpovídaly knihovny napsané v jazyce Python, Ruby, či C++ (ESSENTIA, MARSYAS A LibROSA). Díky přehledné dokumentaci a nízkým závislostem na třetích knihovnách jsme se pokusili využít knihovnu LibROSA. Spolu s interaktivním prostředím IPython Notebook[10] jsem měli možnost i graficky znázorňovat vlastnosti zvukových signálů, spektrogramy a další atributy.

Ve finální verzi aplikace jsme volbu tenkého klienta zavrhli. Knihovnu jsme tudíž využili jen pro získání obrazových podkladů pro znázornění deskriptorů. Rozhodli jsme se tak z několika důvodů. Tím hlavním byl požadavek na provoz aplikace bez nutnosti připojení k internetu, dalším byla komplikovanost z hlediska budoucího nasazení (v ČR je pouze málo webhostingů s dostupným interpretem pro jazyk Python) a v neposlední řadě bylo také spekulativní, zda-li nám serverové řešení přinese rychlostní výhody (odesílání velkých dat, špatné odezvy atd.). Rozhodli jsme se proto zvolit **tlustého klienta**.

Znamenalo to implementaci řídící logiky, matematických výpočtů a ukládání dat přímo v telefonu, z Java knihoven jsme měli na výběr z MPEG-7 Audio Encoder, jAudio a TarsosDSP. Z důvodu kompatibility pro vývoj na platformě Android, nulovým závislostem na jiných knihovnách (vč. javax.sound.xxx) a propracované dokumentaci, jsme nakonec zvolili knihovnu TarsosDSP 2.2.

[9] Zpracování v reálném čase (např. okamžitá odezva při nahrávání zvuku)
[10] ipython.org/notebook.html

Tabulka 2.2: Přehled knihoven

Nástroj	Prostředí	Základní	Spektrální	Časová	Melodické	Rytmické	Podobnostní funkce
ESSENTIA	C++/Python	+	+	+	+	+	+
Musly	C++	–	–	–	–	–	+
MPEG-7 AE	Java	+	+	–	–	–	–
jAudio/jMIR	Java	+	+	–	–	+	–
TarsosDSP	Java, Android	+	+	+	+	+	–
MARSYAS	C++/Ruby/Python/Java	+	+	+	–	+	–
LibROSA	C++/Python	+	+	+	–	+	–
Echo Nest API	REST API	–	–	–	–	–	+

2.3 Požadavky

Požadavky byly stanovené na základě existujících aplikací, které jsme uvedli v sekci 2.1. Snaží se pokrýt jejich funkcionalitu a zároveň poskytnout něco navíc. Požadavky jsou rozdělené na funkční a nefunkční, přehled znázorněn v tabulce 2.3.

Funkční	**Nefunkční**
Nahrání zvukové stopy Přehrání zvukové stopy Uložení nahrávky Smazání nahrávky Zobrazení databáze skladatelů Zobrazení databáze skladeb Přehrání skladby Vyhledávání nahrávky v databázi skladeb Porovnávání nahrávky se zvolenou skladbou	Platforma Android (SDK 17 a více) Funkčnost bez internetového připojení Interní databáze Česká lokalizace Dodržování Android Guidelines Využití Android Material Design

Tabulka 2.3: Požadavky aplikace

2.4 Persony

Persona je fiktivní profil typického zákazníka včetně údajů jako je jméno, věk, fotografie, povolání, záliby a další. Jedná se o cílovou skupinu uživatelů, která je rozšířena a zaměřena na konkrétní osoby. Význam person spočívá v tom, aby byl designer schopen navrhnout aplikaci směrem k potřebám koncových zákazníků. Obor zabývající se uživatelskou přívětivostí se nazývá UX[11] design. V praxi totiž nestačí pouze usoudit, že jsou cílovou skupinou mladí lidé ve věku 20-25 let. Pro pochopení chování každého jednotlivého uživatele je nutné jít více do hloubky, a pro tyto účely se vytvoří persony. Designer se pak dokáže lépe vcítit do jednotlivých postav, odhadnout jejich potřeby a dívat se na problém z různých úhlů pohledu.

Jméno: Ludvík
Věk: 60 let
Povolání: Pianista a hudební skladatel
Profil: Skladatel vážné hudby a klavírní virtuos. Hudební kompozici vystudoval v Miláně a po studiích komponoval taneční skladby, později se vrátil

Obrázek 2.5: Ludvík

[11] Uživatelská zkušenost

k pianu. Jeho hudba se zakládá na minimalismu a působí i v oblasti filmové produkce.

Obrázek 2.6: Marie

Jméno: Marie
Věk: 42 let
Povolání: Učitelka na gymnáziu
Profil: Učitelka na gymnáziu. Vyučuje český jazyk a hudební výchovu. Má na starosti školní pěvecký sbor, který při vystoupeních doprovází hrou na klavír. Každý rok organizuje soutěž pro střední školy s názvem Karlovarský skřivánek.

Jméno: Eda
Věk: 30 let
Povolání: Kytarista
Profil: Kytarista v jazzové kapele. Každý den mají s kapelou zkoušky a snaží se skládat nové písně. Eda se věnuje hudbě již od svých 5 let a s kapelou pravidelně vystupují v místních podnicích.

Obrázek 2.7: Eda

Obrázek 2.8: Markéta

Jméno: Markéta
Věk: 23 let
Povolání: Studentka
Profil: Studentka Akademie múzických umění. Věnuje se oboru Komorní hra – klavírní trio a většinu svého času tráví ve zkušebně. Moderní technologie využívá v každodenním životě, zajímá ji zejména estetická stránka a uživatelská přívětivost.

Kapitola 3

Návrh řešení

3.1 Výběr deskriptorů a porovnávací metody

Nyní se pustíme do návrhu našeho řešení. Je před námi několik problémů. Nejprve zvážíme výběr vhodných deskriptorů a následně zvolíme porovnávací metody pro výpočet podobnosti.

Existují 2 základní možnosti, jak vhodné deskriptory zvolit - automatická a manuální. První z nich je nechat zvolenou knihovnou vygenerovat všechny dostupné deskriptory a nasadit metodu strojového učení. Na trénovací množině by nám byla schopna určit relevantní vlastnosti. Druhou možností je inspirovat se z uskutečněných prací a výzkumů a zvolit ručně již osvědčené deskriptory.

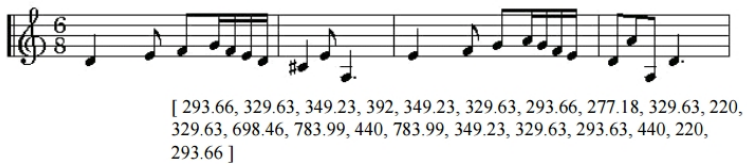
My se vydáme cestou kompromisu a pokusíme se spojit obě dvě možnosti dohromady. Budeme proto vycházet z metodik a přístupů z existujících výzkumů strojového učení a provedeme manuální verifikaci výsledků (viz. testování relevantnosti 5.1) a ten nejlepší způsob –> deskriptor/y nakonec zvolíme pro naši implementaci.

3.1.1 Sekvence výšek

Dle Selfridge-Field, 1998 [28] hraje melodie hlavní roli. *„It is melody that enables us to distinguish one work from another. It is melody that human beings are innately able to reproduce by singing, humming, and whistling. It is melody that makes music memorable: we are likely to recall a tune long after we have forgotten its text.“*. Spolu s rytmem, harmonií a zabarvením vytvářejí hlavní dimenze pro popis hudby. Otázka zní, co ta melodie vlastně je a jak ji reprezentovat. Podle Emilia Gómeze[12] můžeme definovat melodii několika způsoby. Nejčastěji ji definujeme jako sekvenci výšek, případně jako sadu atributů (klíč, kontura a rozsah výšek, melodická „intenzita/hustota“) [29]. Kontura výšek

[12]Člen Music Technology Group Barcelonské univerzity Pompeu Fabra specializující se na výzkum v oblasti MIR a spolu tvůrce knihovny ESSENTIA.

byla využita a aplikována v aplikacích pro vyhledávání podobnosti, klasifikace melodií či query by humming[13] systémech [30,31]. Například Lindsay, 1996 [32] ve své práci získával hodnoty frekvencí za pomoci detektoru výšek založeném na CQT (constant-Q transform) [33]. Naší první volbou proto bude **sekvence (kontura) výšek**, resp. sekvence hodnot frekvencí. Porovnávací funkcí bude algoritmus DTW (Dynamic Time Warping) (viz. 1.5.2).



Obrázek 3.1: Znázornění sekvence výšek [29]

2) MFCC

Pojďme se podívat na další možnost. Jedna z prvních metod pro vyhledávání podobnosti na základě vzoru spočívala v porovnávání Melovských kepstrálních koeficientů (MFCC). Navrhli ji Logan a Salomon, 2001 [34] a jeho metoda každých 26 ms získávala prvních 13 koeficientů, následně tyto shlukovala do k-clusterů. Podobnost se spočítala pomocí EMD [18]. Až do současnosti si MFCC získalo své uplatnění a mnoho prací na nich také zakládá. Aucouturier a Pachet, [35] zase pro výpočet podobnosti využili MFCC spolu se statistickým modelu GMM (Gaussian Mixture Model). Dalším příkladem je úspěšný algoritmus vytvořený Mandel a Ellis, 2005 [36], který se zakládal na metodě Logana a spojoval 2 základní algoritmy: nejvýkonnější algoritmy MIREXu 2006 od Eliase Pampalka a MIREXu 2009/2010 od Pohle-Schnitzer [37].
Naší druhou možností bude tedy získávání **MFCC**. Jako v práci Logana budeme extrahovat prvních 13. Porovnávací funkcí bude opět algoritmus DTW.

3) Onsety

Stejně důležitou složkou, která je spolu s melodií a zabarvením velmi důležitým aspektem vnímání podobnosti, je rytmus. Několik aplikací se zakládá právě na tomto aspektu, příkladem je výzkum založený na rytmických deskriptorech a „vzorech kolísání“ (fluctuation pattern) [38]. Výpočet podobnosti využíval Euklidovské metriky. Pohle et. al, 2009 [37] vyvinuli metodu zvanou *onset patterns*, které byly dle MIREXU[14] výkonnostně nejefektivnějším nástrojem pro porovnávání hudby. Upřednostňovali samotné onsety nad rytmickými vzory a reprezentovali periodicitu na logaritmickém rozsahu. Na metodě spolupracoval Dominik Schnitzer, tvůrce knihovny Musly. Naší poslední volbou bude

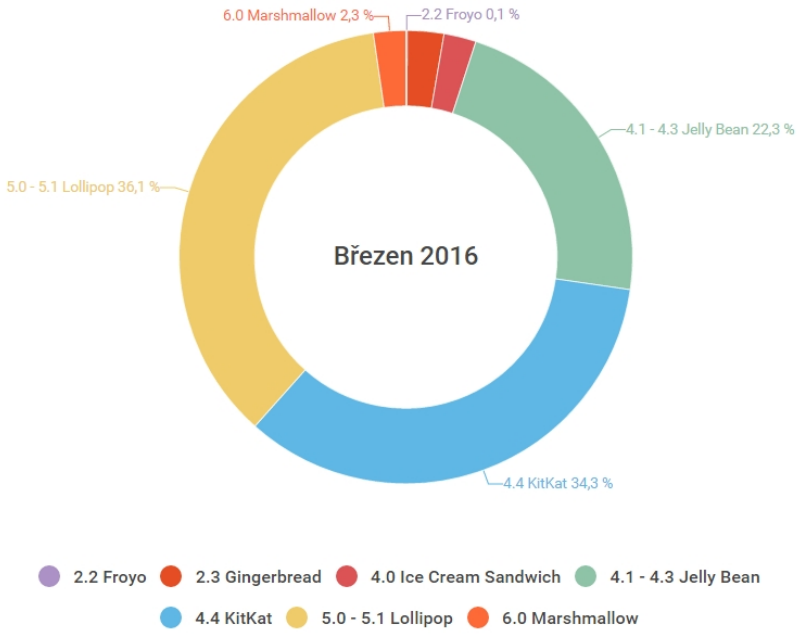
[13] dotaz broukáním/hučením
[14] Komunita pro ohodnocování algoritmů a MIR systémů

porovnávání **onsetů**, čistě na základě rytmu. Porovnávání provedeme pomocí euklidovské vzdálenosti (viz. 1.5.2).

3.2 Platforma

Při volbě platformy jsem vycházel hned z několika aspektů. Tím prvním byla samotná použitelnost aplikace pro koncové uživatele a vůbec zjednodušení celého procesu. Zohledňoval jsme existující aplikace, které byly vesměs implementovány jako webové rozhraní a jen jedna na platformě iOS. Posledním důvodem byl fakt, že jsem si chtěl vyzkoušet programování na moderních platformách, se kterými jsem ještě neměl možnost pracovat. Zvolil jsem tudíž platformu Android.

Dalším rozhodnutím byl výběr verze. Na grafu 3.2 je znázorněno tržní rozložení verzí Androidu v březnu 2016 [39]. Nejrozšířenější verzí je Lollipop (5.0 - 5.1), za ním je v těsném závěsu Kitkat (4.4) a na třetím místě je s 22% zastoupením Jelly Bean (4.1 - 4.3). Starší verze Androidu (Froyo, Gingerbread a Ice Cream Sandwich) zaujímají dohromady necelých 8% trhu. Vzhledem k této statistice jsem se rozhodl implementovat aplikaci podporující verzi Jelly Bean a vyšší (což znamená více jak 92% zastoupení). Konkrétně se jedná o API 17 - 23, které již poskytuje i propracovanější rozhraní a pro naše účely nám v mnoha případech ulehčí práci.



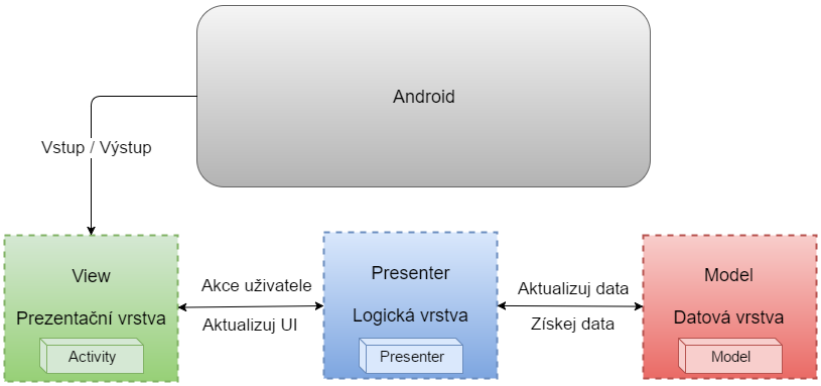
Obrázek 3.2: Podíl verzí systému Android

3.3 Architektura

Na rozdíl od vývoje na platformě iOS, kde je softwarovou architekturou MVC[15], vývoj na Androidu nemá žádné vzorové specifikace. Vrstvy a implementaci volíme na základě našeho uvážení. V naší aplikaci se nicméně architekturou MVC necháme inspirovat a zvolíme její duševně spřízněnou MVP[16] [40], kterou mírně modifikujeme.

Tato architektura rozděluje aplikaci do tří vrstev - **datový model, uživatelské rozhraní a řídící logiku**. Model - datová vrstva spravující data aplikace, View - prezentační vrstva řešící vykreslování na obrazovce, Presenter - řídící vrstva zprostředkovávající komunikaci mezi View a Modelem. Modifikace každé z nich má pouze minimální vliv na ostatní, tudíž je toto rozdělení vhodné pro další vývoj a udržitelnost kódu.

V naší aplikaci balíčky a třídy přirovnáme k jednotlivým vrstvám následujícím způsobem (obr. 3.3). Veškeré třídy z balíčku *Activity* včetně layout definic v jazyce XML zařadíme do vrstvy `View`. O řídící logiku [`Presenter`] se starají třídy z balíčku *Presenter* (RecordPresenter, PlayPresenter, ThreadPresenter ...) a do `Modelu` patří entity z balíčku *Model* (Composer, Song, Recording ...).



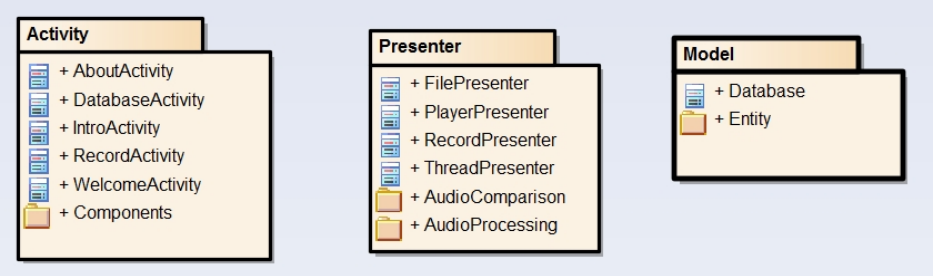
Obrázek 3.3: Architektura MVP

3.4 Model tříd

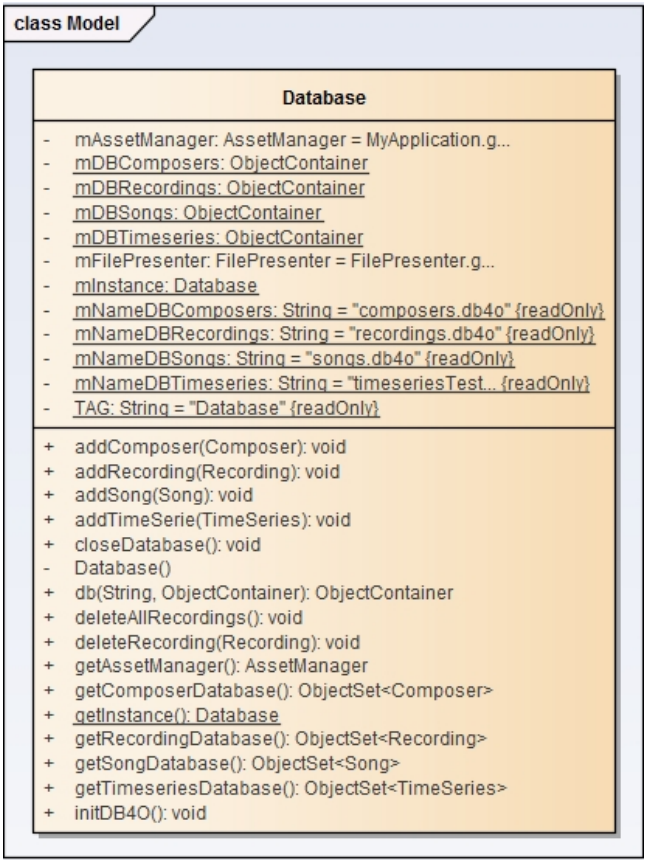
Návrhový model tříd bude vycházet ze zvolené architektury. Aplikace je proto rozdělena do tří základních balíčků – *Activity (View), Model a Presenter*. Kompletní model tříd je součástí dokumentace, kvůli jeho velikosti ho zde neuvádíme. Pro ukázku jsou níže zobrazené určité části. Obrázek 3.4 znázorňuje skladbu tříd v jednotlivých balíčcích. Detailní popis třídy `Database` včetně

[15] Model-View-Controller
[16] Model-View-Presenter

jeho atributů a metod je na obr. 3.5. Tyto podklady byly vygenerované pomocí nástroje Enterprise Architect.



Obrázek 3.4: Model tříd



Obrázek 3.5: Třída Database z balíčku Model

Kapitola 4

Realizace

V kapitole uvedeme konkrétní detaily týkající se implementační části práce. Nejprve se podíváme na volbu nástrojů, následně si popíšeme některá specifika platformy týkající se implementace a v závěru nahlédneme do samotného kódu.

4.1 Volba nástrojů

Ukážeme si nástroje, které jsme využili při realizaci našeho projektu. A to vývojové prostředí a výběr emulátoru.

4.1.1 Vývojové prostředí

Pro implementaci aplikace jsme zvolili vývojové prostředí od české společnosti JetBrains, a to **Android Studio 1.4.1**. Je prakticky totožné s oblíbeným prostředím IntelliJ IDEA pro vývoj Java aplikací a je doporučováno Googlem pro vývoj nativních Android aplikací. Navíc je Android Studio zcela zdarma.

Existují i další IDE[17] jako Eclipse a NetBeans. Ty jsou ovšem podle recenzí neodladěné a často vykazují chyby.

4.1.2 Emulátor

Android Studio nabízí interní emulátor pro vývoj a testování aplikace. Vzhledem k jeho pomalé odezvě raději zvolíme externí řešení od společnosti **Genymotion**, který lze velmi jednoduše integrovat přímo do IDE ve formě pluginu. Spolu s emulátorem budeme aplikaci ladit přes USB rozhraní přímo na mobilním telefonu.

Pro testování se budeme snažit pokrýt co nejširší škálu možných zařízení, tabulka 4.1 znázorňuje zastoupení virtuálních a 4.2 reálných.

[17] Integrated Development Enviroment, česky vývojové prostředí

Tabulka 4.1: Virtuální zařízení

Model	Verze Androidu	API	Rozlišení	RAM (MB)
Google Nexus 5	6.0.0	23	1080x1920	2048
Samsung Galaxy S6	5.1.0	22	1440x2560	3072
HTC Evo	4.2.2	17	720x1280	1024
Custom tablet	4.2.2	17	2560x1600	2048

Tabulka 4.2: Reálná zařízení

Model	Verze Androidu	API	Rozlišení	RAM (MB)
Doogee X5	5.1.0	22	720x1280	1024
Google Nexus 7	6.0.0	23	1920x1200	2048

4.2 Použité knihovny třetích stran

Pro nahrávání audio stopy, získávání deskriptorů a matematické výpočty jsme využili dostupných knihoven. Nyní si je v krátkosti uvedeme.

4.2.1 RehearsalAudioRecorder

Nahrávání zvuku zajišťuje třída `RehearsalAudioRecorder`. Nejedná se o knihovnu jako takovou, spíše o fasádu[18] zastřešující třídy Android frameworku (AudioRecorder, NoiseSuppresor, MediaRecorder, ...). Její výhodou je jednoduché rozhraní a možnost nekomprimovaného nahrávání do různých formátů (např. wav). Samotný `MediaRecorder` toto neumožňuje a používání třídy `AudioRecorder` je značně nízkoúrovňové.

4.2.2 TarsosDSP

Knihovnu jsme již představili v sekci 2.2 a použili jsme ji na zpracování nahrávky a následnou extrakci deskriptorů. Výběr této knihovny byl z důvodu jednoduché integrace do naší aplikace. Knihovna je open-source, pod licencí *GNU General Public License*, což je příkladem copyleftové licence[19].

4.2.3 FastDTW

Knihovna vyvinutá Stan Salvador a Philip Chan, 2007 [41] na technologickém institutu v Melbourne. Využili jsme ji na matematické výpočty, konkrétně pro výpočet vzdálenostní funkce dynamického borcení časové osy 1.5.2. Knihovna

[18] Návrhový vzor Facade (fasáda) se používá k vytvoření jednotného rozhraní pro logickou skupinu tříd, které se tak sdruží do subsystému

[19] Vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí

je distribuována pod licencí *MIT*, což znamená volné užití s podmínkou zachování informace o původním autorství a zřeknutím se odpovědnosti za případné škody.

4.2.4 Db4o

Poslední využitou knihovnou byla db4o od společnosti actian. Jedná se o objektovou NoSQL databázi, která je implementovaná v jazycích Java a .NET. Díky ní jsme se nemuseli starat o mapování našich entit do relačních tabulek. Záznamy jsou totiž ukládány a získávány přímo v podobě objektů, tím jsme ušetřili mnoho času i samotného zdrojového kódu. Je licencována jako *GNU General Public License*.

4.3 Specifika platformy

Níže uvedeme důležitá specifika vývoje aplikace pro Android, která jsou nutná pro pochopení architektury. Nebudeme se však zabývat celým frameworkem [42] a uvedeme jen ty aspekty, které jsou podstatné pro naši aplikaci.

4.3.1 AndroidManifest

Každá aplikace na platformě Android musí deklarovat v kořenovém adresáři soubor „AndroidManifest.xml“. Jedná se o dokument obsahující základní metadata o komponentách aplikace, které je nutné předat operačnímu systému před začátkem samotného spuštění.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="cz.mai.pianosimilarity"
    android:versionCode = "1"
    android:versionName = "0.2">
       <uses-sdk
        android:minSdkVersion = "17"
        android:targetSdkVersion = "23" />
```

Hlavička souboru definuje atributy pro **název balíčku** (unikátní pro veškeré aplikace), **verzování aplikace** (versionCode a versionName) a volitelně vnořený tag **uses-sdk** (alternativou je deklarace v gradle.build). Atribut definuje **minimální a cílové SDK** (Software development kit) určující verzi sady nástrojů (API) pro vývoj a korektní běh programu na podporovaných zařízeních.

V mém případě je aplikace navržena v API 23 a je zpětně kompatibilní až do verze 17. Spustí ji tedy zařízení s Androidem 4.2 (označení Jelly Bean MR1) a vyšší.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
```

Element **application** nastavuje obecné parametry jako název (label) aplikace, ikonku, základní vzhled (schéma) a další. Notaci @[..] vysvětlujeme v sekci 4.3.4.

```
    <activity
        android:name="cz.mai.pianosimilarity.Activity.WelcomeActivity"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity
        android:name=".Activity.RecordActivity"
        android:label="@string/record_activity_label"
        android:theme="@style/AppTheme.NoActionBar" >
    </activity>
```

V Manifestu je nutné definovat také **aktivity** (podrobněji 4.3.2), se kterými bude aplikace pracovat. Ve stručnosti se jedná o komponenty, které reprezentují jednotlivé obrazovky a do kterých se vykresluje uživatelské rozhraní.

V rámci souboru je důležité povšimnout si elementu intent-filter s akcí MAIN, který nám zaručí načtení konkrétní aktivity (zde WelcomeActivity) po spuštění aplikace.

```
<uses-permission
   android:name="android.permission.RECORD_AUDIO" />
```

Posledním důležitým elementem je **povolení (uses-permission)**. Před samotnou instalací aplikace (např. z Google Play) se uživateli zobrazí okno pro schválení seznamu činností (resp. práv), které program při běhu vykonává[20]. Tyto povolení se zapisují do souboru AndroidManifest a v našem případě potřebujeme schválení pro nahrávání zvuků (mikrofon).

4.3.2 Aktivity

Aktivita je třída, která je ekvivalentní s obrazovkou v aplikaci a se kterou uživatel interaguje. Každé aktivitě přísluší okno, do kterého vykresluje své

[20]Platí pro zařízení s Android 5.1 a nižší. Ve verzi 6.0+ jsou povolení vyžadovány až při samotném běhu aplikace.

uživatelské rozhraní [43]. Aktivity mají také svůj životní cyklus a stavy, ve kterých se může aplikace vyskytovat, implementují konkrétní metody.

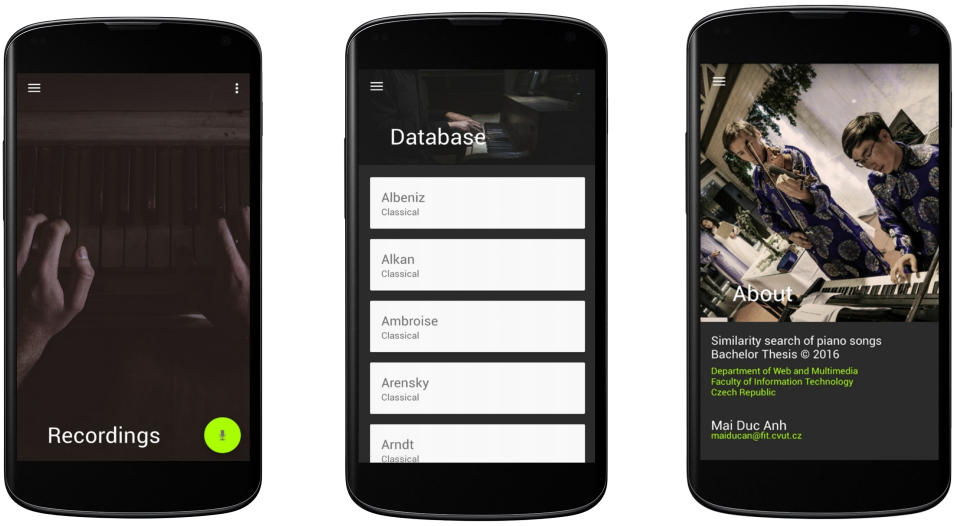
Aplikace se většinou skládá z několika aktivit, mezi kterými se uživatel pohybuje. V případě naší implementace se bavíme například o:

`WelcomeActivity` s úvodní animací po spuštění aplikace

`RecordActivity` pro nahrání zvukové stopy a zobrazení seznamu nahrávek

`DatabaseActivity` se seznamem skladatelů a možností vyhledávání skladeb

`AboutActivity` poskytuje informace o programu a autorovi



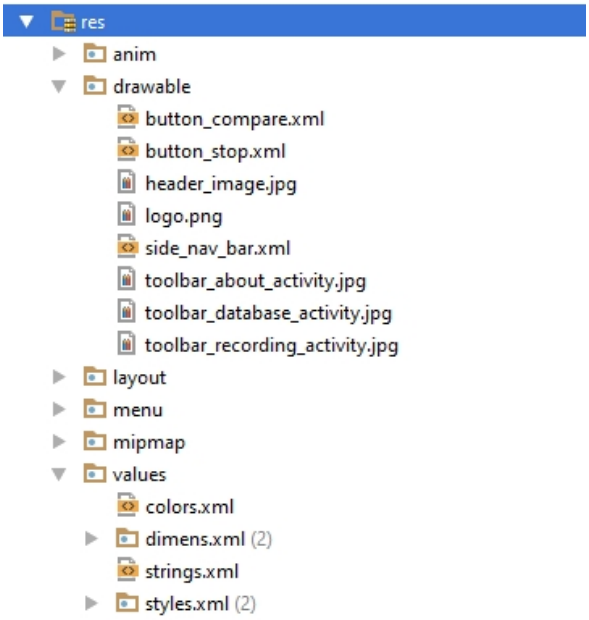
Obrázek 4.1: Náhledy aktivit

4.3.3 Fragmenty

Fragmenty jsou komponenty v aktivitě, které se snaží obalit nějaký celek (např. seznam skladatelů, nahrávek). Do jedné aktivity lze vložit více fragmentů a můžeme je využívat napříč všemi aktivitami. Chápeme je jako recyklovatelné „pod-aktivity“ a jejich životní cyklus je ovlivněn jejich hostitelskou [44]. V naší aplikaci máme fragmenty pro seznam skladatelů `ComposerFragment` a seznam skladeb s možností jejich přehrávání `SongsFragment`, které se nacházejí v aktivitě `DatabaseActivity`.

4.3.4 Zdroje (Resources)

Layouty[21], animace, menu, barvy a další vizuální komponenty jsou oddělené od samotné logiky aplikace [45]. Jejich umístění a definice se nachází ve *zdrojích* se ve složce res. Nejpoužívanějšími komponentami jsou **drawable, layout a values**.



Obrázek 4.2: Zdroje aplikace

Drawable – zde jsou umístěné obrázky a stylování elementů (např. button compare.xml pro nastavení vzhledu tlačítka pro porovnávání nahrávky), můžeme si všimnout formátu xml, který je primárně využíván pro tyto účely.

Layout – ve složce se nachází soubory s definicemi UI[22] jednotlivých aktivit. Opět ve formátu xml.

Values – posledním důležitým prvkem jsou values, kde deklarujeme globální proměnné platné pro celou aplikaci. Tzn. texty, barvy, stylování a další. Použití je pomocí notací, např. @color/barva nebo @string/mujtext (proměnné barva a mujtext si definujeme v souborech ve složce values).

[21] Rozvržení uživatelského rozhraní
[22] User Interface - uživatelské rozhraní

4.3.5 Assety

Jako u zdrojů se jedná o místo, kam můžeme uložit svá data. Assety využijeme tehdy, kdy potřebujeme přidat soubory přímo nesouvisející s vizuální složkou aplikace. Assety jsou zkompilovány do .apk v nezměněné podobě a v našem případě je zde uložena databáze skladatelů, jejich písní ve formátu MIDI a knihovny FFMPEG.

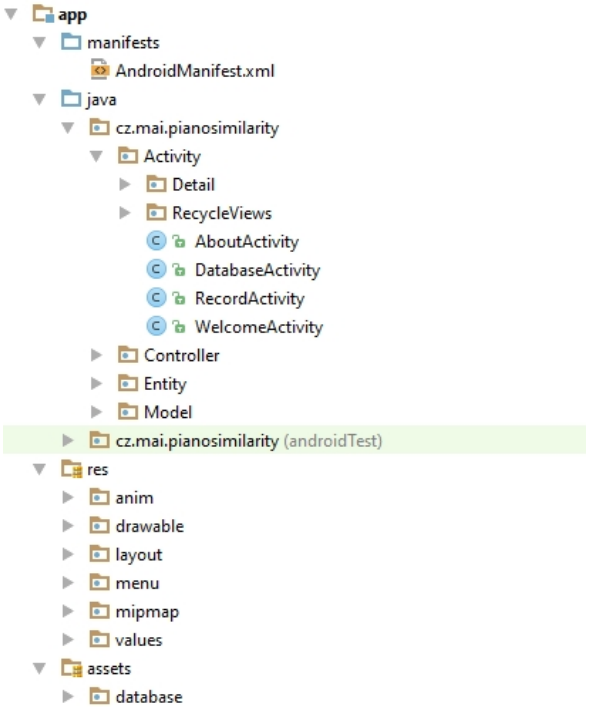
4.3.6 Vlákna

Po spuštění aplikace se vytvoří primární vlákno `(UI thread)`, které zajišťuje vykreslování uživatelského rozhraní a zpracovávání událostí. Pouze toto vlákno má možnost vytvářet či modifikovat komponenty na obrazovce a pokud bychom v něm vykonávali časově náročnější úlohu, např. načítání hudby z databáze nebo stahování dat z internetu, mohlo by dojít k jeho zablokování, případně pádu celé aplikace [46]. V tomto případě je vhodné spouštět akce na pozadí a to zavoláním nových vláken. V Androidu jsou připravené třídy, které nám výrazně ulehčí práci.

- **Thread** je klasické vlákno, o které se musíme starat sami. Od spuštění, přes synchronizaci s hlavním vláknem až po ukončování v případě nečekaných událostí v aktivitě (otočení displeje, přechod na jinou aktivitu apod.).
- **AsyncTask** je třída, která nám poskytuje veškerou správu vláken a jejich synchronizaci s vláknem hlavním. Zároveň nám předává indikátory průběhu akcí. Pro použití stačí pouze implementovat metody *doInBackground() a onPostExecute()* a o zbytek se nemusíme starat. Má však omezení v podobě počtu volaných akcí apod [47].
- **Handler** je také součástí Android frameworku pro správu vláken. Oproti AsyncTask nám však poskytuje větší svobodu a volnou ruku při jejich manipulaci. Nemá žádné omezení a umožňuje komunikaci pomocí zpráv mezi jednotlivými vlákny.
- **Service** je komponenta, která umožňuje v pozadí vykonávat dlouhotrvající úlohy i bez nutnosti zapnuté aplikace. Není vázána na životní cyklus aktivit.

V naší aplikaci jsme implementovali první tři možnosti. Detaily týkající se obsluhy vláken v sekci 4.5.1.

4.4 Struktura projektu



Obrázek 4.3: Zjednodušená struktura projektu

Kořenová složka **app** se větví na tři části - build, libs a src. Build slouží pro ukládání předkompilovaných částí souborů, využívá ho zejména vývojové prostředí a uživatel se tímto nemusí zabývat. Libs je určen pro přidávání externích knihoven, což v našem případě taktéž nevyužijeme. Nejdůležitější složkou je tedy **src**, kde je umístěn samotný kód.

Obrázek 4.3 znázorňuje její zjednodušený obsah. Složka se dále větví na *java*, *res* a *manifest*.

- java – umístění implementace (třídy, logika aplikace, aktivity 4.3.2, ...)
- res – umístění pro layouty, animace, obrázky (podrobněji v sekci4.3.4)
- manifest – obsahuje soubor AndroidManifest.xml (více 4.3.1)

4.5 Implementace

Podíváme se na implementační detaily aplikace Piano Similarity. Projdeme si ty nejdůležitější části celého procesu, od nahrávky stopy uživatele až po zobrazení výsledků.

4.5.1 Obsluha procesů a vláken

Začneme komponentou, která obsluhuje průběh prováděných akcí v celé aplikaci a má na starosti správu vláken a jejich vzájemnou komunikaci. Jedná se o třídu `ThreadPresenter`, která dědí objekt `Handler` (viz. 4.3.6). Implementuje metodu *handleMesssage(Message)* reagující na přijaté zprávy z různých částí aplikace a pracujících vláken. Využíváme ji pro synchronizaci vícevláknových úloh, tzn. pro odesílání oznámení o jejich dokončení, resp. vyvolání dalších operací. Jednotlivé zprávy jsou zapouzdřené do instance `Message` a nastavíme ji příznak (proměnná what), na jehož základě správce rozhodne o provedení další operace. Objekt `Message` odešleme jako parametr pomocí volání metody *sendMessage(msg)*, která je součástí rodičovské třídy `Handler`.

Ukázka odeslání zprávy

```
// Send message to handler to record audio //
Message msg = Message.obtain();
msg.what = RECORD_AUDIO;
mThreadHandler.sendMessage(msg);
```

Metoda *handleMessage(Message)* spravující procesy

```
public void                          handleMessage(Message msg) {

        switch (msg.what){

                // Record audio
                case RECORD_AUDIO:          runRecording();
                                                break;
                // Stop recording
                case STOP_RECORDING:        runStopRecording();
                                                break;
                // Extract descriptors from audio
                case EXTRACT_DESCRIPTORS:   runExtraction();
                                                break;
                // Compare with database
                case COMPARE_DATABASE:      runComparisonDatabase();
                                                break;
                // Compare with selected song
                case COMPARE_SONG:          runComparisonSong();
                                                break;
                // Show results
                case SHOW_RESULTS:          showResults();
                                                break;

        }
}
```

4.5.2 Nahrávání audio stopy

Pro nahrání zvukové stopy budeme pracovat s následujícími třídami. Interakci s uživatelem poskytuje aktivita `RecordActivity`, o řídící logiku se stará `ThreadPresenter` a `RecordPresenter` a samotný proces nahrávání zajišťuje třída `RehearsalAudioRecorder`.

V aktivitě se nachází metoda *setListenters()*, která obsahuje handlery[23]. V okamžiku spuštění nahrávání se nejprve ověří oprávnění pro použití interního mikrofonu (o povoleních jsme se zmínili v sekci 4.3) a je-li zařízením/uživatelem uděleno, odešle se zpráva `Message` do `ThreadPresenteru`, který zajistí další akce. Ten vyvolá metodu *record()* nacházející se v presenteru `RecordPresenter`. Zároveň na displeji nastaví vizuální komponenty (zatemnění pozadí, tlačítko pro ukončení nahrávání, dialog ve spodní liště) pro informování uživatele o prováděné akci.

Metoda v presenteru spustí nové vlákno, které nastaví třídní proměnné (počet nahrávek, čas nahrávky, jméno nahrávky atd.) a vytvoří instance tříd `Recording` (tato se bude ukládat do databáze) a `RehearsalAudioRecorder`, která má na starosti nahrávání zvuku. V parametrech konstruktoru nastavíme, zda-li má být nahrávka komprimovaná, jaké je vstupní zařízení, nastavení vzorkovací frekvence, mono/ stereo kanálu a kódování. Na instanci následně zavoláme metody *prepare()* a *start()*, které nahrávání spustí.

```
public void record()  {

mThread = new Thread(new Runnable() {

        public void run() {

                mCountRecordings++;
                mTime = String.valueOf(
                           System.currentTimeMillis() / 1000);

                mRecordingName = mDirectoryPath + mTime + "_" +
                                 mCountRecordings + ".wav";

                mAudioRecorder = new RehearsalAudioRecorder(
                        RehearsalAudioRecorder.RECORDING_UNCOMPRESSED,
                        MediaRecorder.AudioSource.MIC,
                        44100,
                        AudioFormat.CHANNEL_IN_STEREO,
                        AudioFormat.ENCODING_PCM_16BIT);

                mAudioRecorder.setOutputFile(mRecordingName);
                mAudioRecorder.prepare();
                mAudioRecorder.start();

        }
});
```

[23] obsluha událostí

```
mThread.start();

}
```

Jakmile uživatel nahrávání ukončí, opět se odešle zpráva o ukončení akce, následně presenter uloží instanci nahrávky (entita `Recording`) do databáze - pomocí třídy `Database` v balíčku Model (architektura MVP, viz. 3.3).

4.5.3 Práce s databází

Ukládání, získávání, editace a mazání dat má na starosti třída `Database` z balíčku Model. Zajišťuje správu entit `Recording, Composer, Song`, obsahuje CRUD metody pro objekty z databáze Db4o a poskytuje referenci k assetům. Třída je vytvořena na základě návrhového vzoru *Singleton*[24], při spuštění aplikace se proto vytvoří pouze jediná instance. Níže je ukázka třídních metod pro vytvoření/otevření databází, získání instance na assety a přístup ke kolekci nahrávek.

```
/**
* Method creates or open the database from the path
* and returns the ObjectContainer instance
*/
public ObjectContainer  db(String filePath, ObjectContainer oc) {

        try {
                if (oc == null || oc.ext().isClosed()) {
                        oc = Db4oEmbedded.openFile(filePath);
                        Log.d(TAG, "Opened␣new␣database␣connection");
                }

                return oc;

        } catch (Exception ie) {
                Log.e(TAG, "Unable␣to␣open␣database");
                return null;
        }
}

/* Method provide an instance for accessing assets */
public AssetManager             getAssetManager() {
        return MyApplication.getAppContext().getAssets();
}

/* Method returns the collection of all recordings */
public ObjectSet<Recording>     getRecordingDatabase() {

        ObjectSet<Recording> result;
        result = mDBRecordings.query(Recording.class);
```

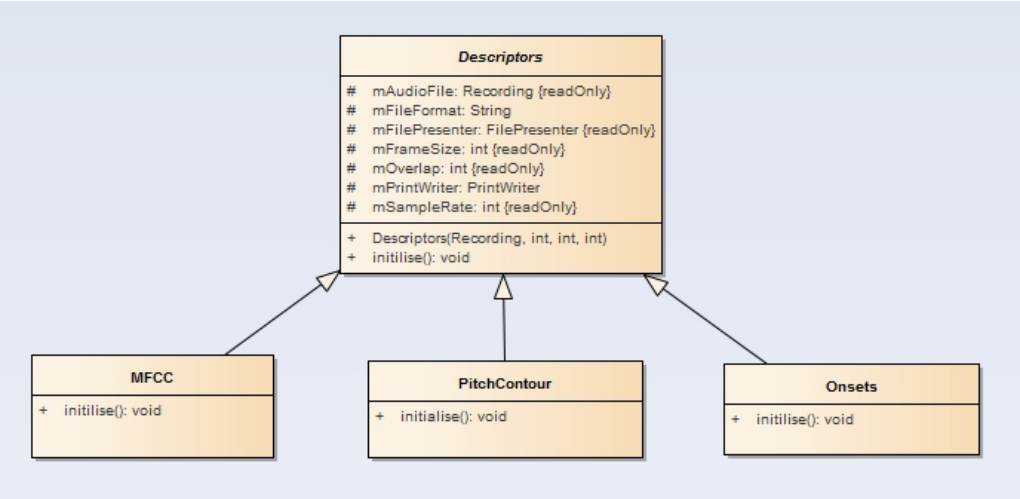
[24] Česky jedináček nebo unikát

```
        return result;

}
```

4.5.4 Extrakce deskriptorů

Extrahování deskriptorů z nahrávek zajišťují následující třídy. Veškerou řídící logiku pro získávání deskriptorů zajišťuje třída `Extraction` z balíčku Presenter. Samotné deskriptory jsou zapouzdřené do stejnojmenných tříd(`PitchContour`, `MFCC` , `Onsets`) a dědí od abstraktní třídy `Descriptors` v balíčku Model. Ta obsahuje atributy – nahrávka (Recording), vzorkovací frekvence (int), velikost okna (int), překrytí (int), soubor pro uložení deskriptorů (PrintWriter)). A poslední komponentou, kterou při extrakci využijeme, je samotná knihovna TarsosDSP 2.2 pro extrahování dat.



Obrázek 4.4: Abstraktní třída Descriptors s entitami

Vytvoříme instanci třídy `Extraction`, do jehož parametru předáme naši nahrávku, resp. entitu `Recording`. Na instanci zavoláme metodu pro získání deskriptorů *extractDescriptors(int)* a v parametru předáme hodnotu konkrétního typu (1 - PitchContour, 2 - MFCC, 3 - onsety). Metoda na základě volby určí další kroky extrakce. Například pro konturu výšek se následně vytvoří nový objekt `PitchContour`, což je již samotná entita deskriptoru. Předáme námi zvolené parametry vzorkování, velikosti okna a překrytí. Úplně v posledním kroku zavoláme na instanci metodu *initialise()*, která provede zpracování nahrávky a vyextrahuje požadovaná data. Nyní si ukážeme implementační detail závěrečného procesu v této metodě a využití knihovny TarsosDSP pro extrakci kontury výšek.

1. Inicializace FFMPEG knihoven, které TarsosDSP využívá. V parametru předáváme kontext aplikace.

```
new AndroidFFMPEGLocator(MyApplication.getAppContext());
```

2. Vytvoření instance `AudioDispatcher`, která převezme soubor s nahrávkou a definuje vstupní parametry (vzorkování, okno, překrytí)

```
AudioDispatcher adp = AudioDispatcherFactory.fromPipe(
        mFilePresenter.getRecordingFolderPath() +
        mAudioFile.getTitle() +
        ".wav",
        mSampleRate,
        mFrameSize,
        mOverlap);
```

3. Na instanci zavoláme metodu *addAudioProcessor()*, který přijímá parametry podle typu deskriptoru, který chceme získat. Pro konturu výšek přijímá následující parametry - instanci třídy `PitchProcessor`, u které si zvolíme algoritmus pro detekci výšek (zde FFT_YIN [48]), vzorkování, okno a instanci třídy `PitchDetectionHandler`, který ze zvukové stopy vyextrahuje hodnoty frekvencí (float). Ty nakonec zapíšeme do souboru.

```
adp.addAudioProcessor(new PitchProcessor(
        PitchProcessor.PitchEstimationAlgorithm.FFT_YIN,
        44100,
        mFrameSize,
        new PitchDetectionHandler() {

                public void handlePitch(
                                PitchDetectionResult pitchDetectionResult,
                                AudioEvent audioEvent) {

                        if (pitchDetectionResult.isPitched()) {

                        float[] pitch = new float[1];
                        pitch[0] = (float) pitchDetectionResult.getPitch();

                        // Print data to the file
                        PitchContour.this.mPrintWriter.println(pitch[0]);

                }
        }
}));
```

4. Cestu k tomuto souboru získáme zavoláním metody *getFilePath()* ve vytvořené instanci `PitchContour`. Hodnoty jsou zapsané každá na novém řádku.

4.5.5 Porovnávání otisků

Po extrahování otisku z nahrávky následuje porovnání s databází. Celý proces rozdělíme do 3 kroků.

1. **Ověření a příprava**

   V prvním kroku ověříme, jestli uživatel nahrál validní záznam, resp. zda-li je soubor s deskriptory neprázdný. V případě neúspěchu informujeme uživatele o opakování nahrávání, jinak vytvoříme instanci třídy `Comparison` a do jeho konstruktoru předáme počet vláken, které chceme využít pro následné výpočty. Dále vytvoříme instanci `TimeSeries` (reprezentace hodnot deskriptorů), se kterou funkce pro porovnávání pracuje. Nakonec vyvoláme metodu *compareWithDatabase(TimeSeries)* na vytvořené instanci `Comparison`, která provede samotné porovnání.

```
// Check, if user recorded anything
if (mExtraction.getExtractionSize() == 0) {
        RecordActivity.setNothingRecordedState();
        return;
}

// Otherwise continue, inform user on Activity
RecordActivity.setComparisonState();

// Create new Comparison instance
// And set number of threads for computing
int numOfThreads = 4;
mComparison = new Comparison(numOfThreads);

// Create timeseries from extraction file
final TimeSeries newTimeserie = new TimeSeries(
                mExtraction.getDescriptorsFilePath(),
                false,
                false,
                ',');

// Run comparison with database
mComparison.compareWithDatabase(newTimeserie);
```

2. **Porovnávání s databází**

   Následuje inicializace pole vláken `Thread[] mThreads`, jehož velikost jsme určili při vytváření instance (v ukázce proměnná `threadCount`), vyhrazení paměti pro ukládání mezivýsledků jednotlivých vláken `TreeMap<Double, Integer>[] mSemiResults` a celkových výsledků `Collections.synchronizedMap(new`

`TreeMap<Double,Integer>()) mResults`, která shromažďuje jednotlivé mezivýsledky a je *thread-safe*[25].

```
mThreadCount = threadCount;
mThreads = new Thread[mThreadCount];
mSemiResults = new TreeMap[mThreadCount];
mResults = Collections.synchronizedMap(new TreeMap<Double,Integer>());
```

Porovnávání probíhá paralelně na celé databázi (počet skladeb = m) a každé vlákno zpracovává právě (m / threadCount) porovnání. Pro výpočty využíváme knihovny `FastDTW` a její metodu *getWarpDistBetween()*, které předáme 2 instance `TimeSeries` - nahrávky a skladby, a vzdálenostní funkci (v našem případě Euklidovská). Jednotlivá vlákna zapisují své výsledky do datové struktury `mSemiResults`, kde je prvním atributem (klíčem) vypočtená vzdálenost mezi nahrávkou a skladbou, a druhým (hodnotou) je identifikátor skladby. Tato kolekce nám rovnou ukládá výsledky seřazené podle klíče (resp. vzdálenosti) od nejnižšího (nejpodobnější skladby), a nemusíme se tak starat o finální řazení. Výpočetní složitost přidávání je log(n).

```
TimeSeries eachTimeSerie = mTimeSeriesList.get(compareNum);
distance = FastDTW.getWarpDistBetween(
                mRecordingTS,
                eachTimeSerie,
                mDistanceFunction);
sumDistance += distance;

mSemiResults[threadNum].put(distance, eachTimeSerie.getID());
```

Každé vlákno je nezávislé na ostatních a nedochází k žádnému přepisování výsledků. V okamžiku, kdy vlákna dokončí svou práci, přiřadí své výsledky do finální struktury `mResults`. Uchováváme také sumu všech vzdáleností `mDistanceSum`, kterou později využijeme pro výpočet podobnostní míry.

```
synchronized (mResults) {
        mDistanceSum += sumDistance;
        mResults.putAll(mSemiResults[threadNum]);
}
```

3. **Filtrace výsledků a výpočet hodnot podobností**

   Po porovnávání získáváme celou kolekci výsledků ve formě dvojic (vzdálenost, ID písně). Nyní vyfiltrujeme k - nejpodobnějších a u nich ještě převedeme hodnotu vzdálenosti na míru podobnosti.

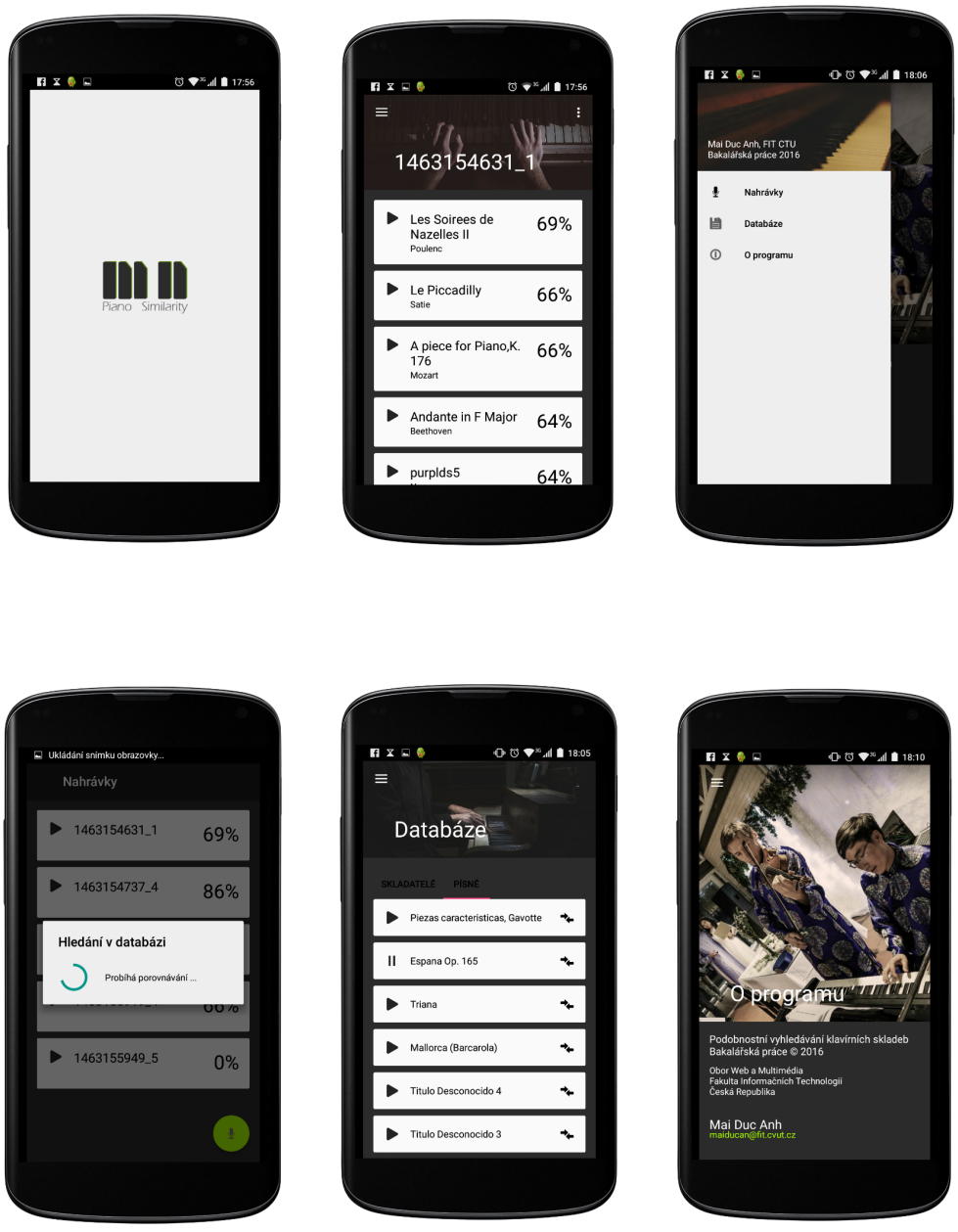
[25] vláknově bezpečná - sdílení datových struktur takovým způsobem, které zaručuje jejímu bezpečnému využívání pomocí většího počtu vláken současně

Situaci máme ulehčenou tím, že jsme výsledky řadili již při vkládání do datové struktury, k - nejpodobnějších skladeb tudíž odpovídá prvním k dvojicím v kolekci `mResults`. Nyní zbývá spočítat podobnostní míry. Budeme je reprezentovat celočíselnými hodnotami (0 - 100) a získáme je následovně.

Nejprve spočteme průměrnou hodnotu vypočtených vzdáleností. Jednotlivé podobnosti pak odpovídají podílu jejich hodnot vzdáleností a tohoto průměru, převedené do podobnostního skóre (procenta). Rozhodli jsme se pro výpočet využít zprůměrované hodnoty a nikoli maximální, jelikož rozpětí výsledků bylo vždy velice široké. Podobnostní míry by v tom případě nabývaly velmi vysokých (neodpovídajících) hodnot.

```
double meanDistance = mDistanceSum / mDatabaseSize;
double similarity = (1 - value / meanDistance) * 100;
```

4.6 Výsledek aplikace



Obrázek 4.5: Výsledek aplikace

Kapitola 5

Testování

5.1 Testování relevantnosti

5.1.1 Cíl

Jsou implementovány 3 různé metody, pomocí kterých se porovnává vstupní melodie s databází skladeb. Cílem tohoto testování bylo zjištění, který z těchto 3 způsobů je z lidského vnímání nejlepší.

5.1.2 Metodika

Postup testování byl následující. Nejdříve jsme vybrali 5 referenčních skladeb a každou z nich jsme v modulu pro podobnostní vyhledávání otestovali.

Respondent následně obdržel 5 souborů. Každý z nich obsahoval jednu referenční skladbu a 2 nejpodobnější z každé metody. Celkem tedy 6 podobných zvukových stop. Tento počet byl zvolený proto, abychom získali relativně dostatečné množství dat a zároveň se doba celého testování vešla do 10 minut. Úryvky měly délku 10 sekund. Respondent nebyl předem obeznámen s porovnávacími metodami, hodnocení tím tudíž nebylo nijak ovlivněno. Úkolem respondentů bylo ohodnotit každou z těchto 6 skladeb známkou 1-10 (10 nejvyšší) a to ze tří hledisek. Z hlediska melodie, rytmiky a celkového zabarvení/nálady. Hodnocení respondent zapisoval do tabulek.

5.1.3 Výsledky

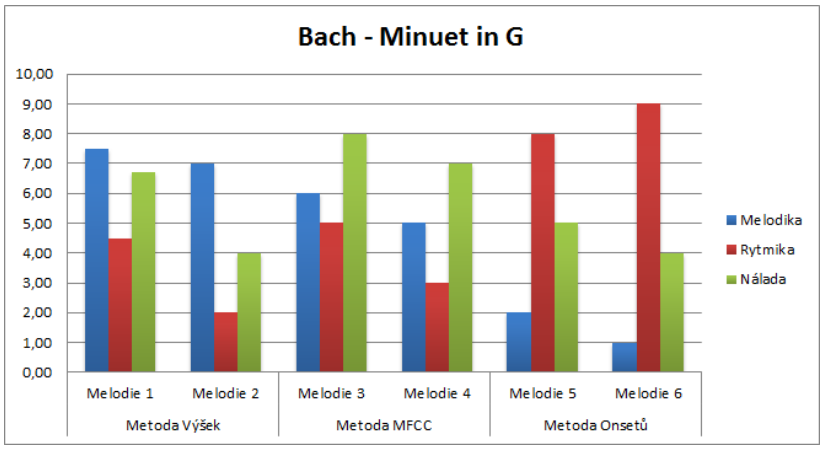
Průzkumu se zúčastnilo celkem 45 respondentů. Z toho se 32 dotazovaných aktivněji zajímá či zajímalo o oblast hudby (např. ovládá určitý hudební nástroj). Zbylých 13 jsou hudební laikové.

Získali jsme od respondentů odpovědi, které nám nyní pomohou určit, která z navržených porovnávacích metod je z lidského vnímání nejlepší. Můžeme si všimnout, že kritéria pro ohodnocení podobnosti (melodika, rytmika,

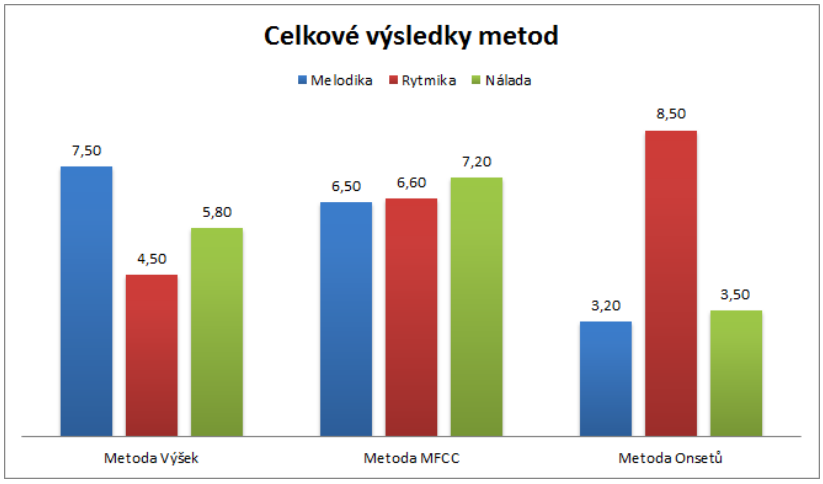
zabarvení/nálada) jsou vybrána ne zcela náhodně. Jednotlivé metody pro porovnávání s nimi totiž úzce souvisejí.

Mohli jsme očekávat, že první metoda, kde získáváme sekvenci výšek/frekvencí, bude vykazovat lepší výsledky v melodice. Metoda porovnávání onsetů bude zase velmi přesná na rytmus, ovšem melodicky nemusejí být skladby vůbec podobné. Metoda porovnání zprůměrovaných MFCC oba předchozí a mnohé další atributy zohledňuje. Odhadovali jsme, že bude vykazovat nejlepší výsledky.

Grafy níže znázorňují zprůměrované výsledky. Graf 5.1 pro skladbu Minuet in G (Johann Sebastian Bach) a graf 5.2 udává celkové výsledky napříč všemi soubory. Můžeme vyčíst, že naše odhady o účinnosti metod byly celkem pravdivé.



Obrázek 5.1: Graf vyhodnocení podobnosti pro skladbu Minuet in G



Obrázek 5.2: Graf celkového vyhodnocení metod

5.1.4 Volba řešení

Výsledky nás nepřekvapily, bohužel ani dostatečně nepřesvědčily. Metodu Onsetů můžeme rovnou vyloučit pro nízkou relevanci z hlediska melodiky, která je tou nejstěžejnější. Metoda výšek i MFCC vykazují poměrně shodné výsledky, ovšem ani jedna výrazněji nesplnila očekávání. Nabízí se řešení spojit všechny metody do jedné a následně aplikovat porovnání. To je ovšem výpočetně velice náročné a výpočet by trval řádově minuty. Aplikace by byla v praxi nepoužitelná. Ve finální implementaci jsme však nakonec zvolili porovnávání pomocí sekvence výšek a to z důvodu výpočetní rychlosti porovnání.

5.2 Testování rychlosti

5.2.1 Cíl

Testování rychlosti je velmi důležitým měřítkem pro ohodnocení kvality implementace. Aplikace implementuje metody, které fungují na principu porovnávání extrahovaných deskriptorů v určitém časovém úseku. Tato sekvence může nabývat variabilní velikosti, neboť můžeme porovnávat různě dlouhé melodické úseky (např. prvních 10, 20, či 30 sekund) a k tomu ještě melodii jakkoli jemně vzorkovat.

Například při vzorkovací frekvenci 44100Hz (dnes nejčastěji užívaná) a velikosti okna 1024 (v milisekundách) získáváme vzorek/hodnotu každých 1024/44100 sekund 23ms. Jedna sekunda tudíž odpovídá 43 skalárům. Pro 10 sekundovou melodii získáváme posloupnost 430 hodnot, které nyní musíme porovnat s každou takovou sérií v databázi (v našem případě o velikosti 1600).

Volba délky porovnávací melodie, jemnost vzorkování a velikost okna je samozřejmě čistě na našem zvážení. Tyto parametry budou proto zkoumáním pro testování rychlosti. Pokusíme se odpovědět na otázku, jaké nastavení vstupních parametrů bude pro naše řešení optimální, zdali nám vyšší vzorkování přinese relevantnější výsledky nebo se jen zvýší výpočetní nároky a nezískáme žádnou výhodu.

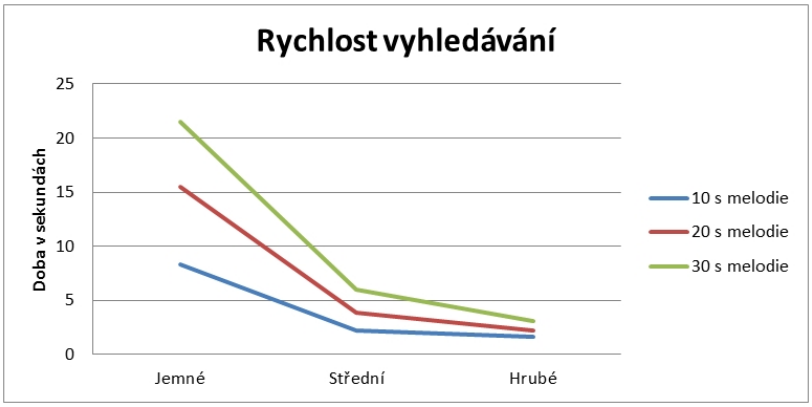
5.2.2 Metodika

Kombinace vstupních parametrů jsme zvolili následujícím způsobem. Vytvořili jsme 3 sady (jemné, střední a hrubé vzorkování) a celkem 9 různých kombinací, přehled v tabulce 5.1. Testování bylo provedeno na stolním počítači (AMD Phenom II X4, 3.6Ghz, 8GB RAM) bez využití paralelních výpočtů, na jednom jádře.

Vzorkování	**Vzorkovací frekvence**	**Velikost okna (ms)**	**Překrytí (ms)**	**Perioda (v ms)**	**Délka melodie (sekundy)**	**Celkový počet vzorků/hodnot**
Jemné	44100	512	256	11,61	10	1723
					20	3445
					30	5168
Střední	44100	1024	0	23,22	10	431
					20	861
					30	1292
Hrubé	22050	1024	0	46,44	10	215
					20	431
					30	646

Tabulka 5.1: Nastavení vzorkování

5.2.3 Výsledky



Obrázek 5.3: Graf testování rychlosti vyhledávání

Výsledky byly spočteny na základě zprůměrovaných časů pro 5 měření (odchylky byly velmi malé). Zajímavé bylo, že při každém prvním pokusu, resp. po změně vstupních parametrů, byl čas vyšší o 50-100% než při dalších pokusech. Důvodem je zřejmě cachování na procesoru.

Podíváme-li se na výsledky, jemné vzorkování a délku melodie 20s, výpočet trval bezmála 16 sekund. Což sice není tak hrozné, otázkou je, zdali nám takovéto vzorkování přinese relevantnější výsledky. Z testování vyplynulo naopak zhoršení kvality vyhledávání. Perioda vzorkování 11,61 ms s polovičním překrytím se ukázala být jako velmi nízká a algoritmus pro detekci výšek často naměřil nesprávnou hodnotu. Do úvahy tedy přichází vzorkování střední či hrubé, které má zároveň i slušné rychlostní předpoklady.

V oblasti MIR (Music Information Retrieval) se setkáme nejčastěji se vzorkováním hrubým. Segmentace na úseky o délce zhruba 50 ms s polovičním překrytím je v praxi využíváno zejména kvůli detekci nízkých frekvencí. Na druhou stranu je zde nebezpečí ztráty informace (Shannonův teorém).

Pro naše účely tedy zvolíme vzorkování střední (vzorkovací frekvence 44100, okno 1024 ms) a délku porovnávání vždy prvních 10 sekund. Tyto vstupní parametry byly i s ohledem na testování relevantnosti nejoptimálnější. Rychlost porovnávání se pohybuje okolo 2 sekund (na stolním počítači), což je přijatelné.

5.3 Porovnání s existujícími aplikacemi

Na závěr shrneme výhody a nevýhody naší aplikace vůči existujícím aplikacím, které jsme popsali v sekci 2.1. Budeme zohledňovat tyto aspekty: funkcionalita, kvalita vyhledávání, rychlost a uživatelská přívětivost.

Výhody	**Shoda**	**Nevýhody**
Vyhledávání pomocí mikrofonu Porovnávání se zvolenou skladbou Ukládání nahrávek Bez nutnosti připojení k internetu Platforma Moderní design Uživatelská přivětivost	Kvalita vyhledávání Přehrávání skladeb	Rychlost vyhledávání Absece vyhledávání pomocí kláves Absence zobrazování not

Tabulka 5.2: Shrnutí výhod a nevýhod aplikace

Závěr

V práci jsme uvedli základní principy analýzy hudby a podobnostního vyhledávání. Vysvětlili jsme obecný způsob, jakým lze postupovat při porovnávání zvukových stop, od základních zvukových charakteristik, přes extrahování deskriptorů (včetně jejich popisu) až po metodiky a podobnostní funkce, na jejichž principech staví současné MIR systémy.

Dále jsme prozkoumali dostupné aplikace. Zjistili jsme, že v oblasti vyhledávání audia (veškerého) existuje veliké množství služeb. Co se však týče podobnostního vyhledávání klavírních skladeb, nalezli jsme pouze 4 webové aplikace, k tomu většinou zastaralé a s minimální uživatelskou přívětivostí. Provedli jsme také rešerši knihoven a aplikačních rozhraní pro extrahování deskriptorů z nahrávek. Měli jsme na výběr z 8 knihoven, z toho 4 implementované v jazyce C++/Python, 3 v jazyce Java a 1 jako serverový klient s aplikačním rozhraním REST. S ohledem na architekturu a možné nevýhody pro serverové řešení, jsme se nakonec rozhodli implementovat tlustého klienta a vybrali jsme jednu z Java knihoven.

V návrhu řešení jsme na základě existujících prací a výzkumů založených na strojovém učení zvolili 3 druhy deskriptorů, které jsme v závěru podrobili manuálnímu otestování. Jednalo se o deskriptory výšek, MFCC a onsety. Porovnávací funkcí byl algoritmus dynamického borcení časové osy (DTW). Následně jsme se pustili do implementace, která se odvíjela od softwarového návrhu a zvolené architektury. Využili jsme knihoven třetích stran pro usnadnění aplikačních procesů a implementaci matematických výpočtů.

V závěru jsme provedli otestování. Z výsledků testování kvality, resp. deskriptorů, jsme se rozhodli pro finální implementaci zvolit deskriptor výšek. Co se týče testování rychlosti, zkoumali jsme výběr vstupních parametrů pro vzorkování a z výsledků jsme usoudili jako nejoptimálnější konfiguraci střední jemnost samplování.

Na základě finálního porovnání s existujícími aplikacemi se nabízí několik možností vylepšení. Hlavním nedostatkem je především rychlost. I přestože je aplikace implementována jako vícevláknová a využívá maximální výpočetní

kapacitu, narážíme na strop mobilních procesorů, které jsou pro tyto úlohy nedostatečně výkonné. Úplně nejoptimálnějším řešením by bylo předzpracování nahrávky na mobilním zařízení (extrakce deskriptorů) a následné porovnání otisků na serveru. Dalším vylepšením by byla optimalizace algoritmu pro vyhledávání pomocí indexace. Museli bychom u každé skladby nalézt pasáž (např. refrén či opakující se část), která by ji reprezentovala a touto pasáží bychom mohli skladbu identifikovat. Nabízí se také další možná vylepšení z hlediska funkcionality (např. zobrazení not u skladeb) či grafické znázornění podobných částí nahrávky a skladby.

Literatura

[1] Lincoln, H. B.: Some criteria and techniques for developing computerized thematic indices. In Heckmann, editor, Electronische Datenverarbeitung in der Musikwissenschaft, Regensburg, 1967.

[2] McLane, A.: Music as information. In M.E. Williams, editor, Arist, volume 31, chapter 6, pages 225–262. American Society for Information Science, 1996.

[3] Typke, R.; Wiering, F.; Veltkamp, R. C.: A Survey of Music Information Retrieval Systems. In *In ISMIR*, 2005, s. 153–160.

[4] Klapuri, A.: *Signal Processing Methods for Music Transcription*. New York: Springer, 2006, ISBN 978-0-387-30667-4.

[5] Schnitzer: *Indexing Content-Based Music Similarity Models for Fast Retrieval in Massive Databases*. Dissertation, 2012.

[6] B. Manjunath, P. S.; Sikora, T.: *Introduction to MPEG-7: Multimedia content description interface*. Wiley, 2001, ISBN 0-471-48678-7.

[7] Kim, H.-G.; Moreau, N.; Sikora, T.: *MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval*. John Wiley & Sons, 2005, ISBN 047009334X.

[8] Davis, S.; Mermelstein, P.: *Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences*. IEEE Trans. on Acoustic, Speech and Signal Processing, 1980.

[9] Sahidullah, M.; Goutam, S.: *Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition*. Speech Communication 54 (4): 543–565, 2012.

[10] Hyoung-Gook Kim, T. S.: Comparison of MPEG-7 Audio spectrum projection features and MFCC applied to speaker recognition, sound classification and audio segmentation. 2004, content Based Retrieval. Dostupné z: http://elvera.nue.tu-berlin.de/files/0781Kim2004.pdf

[11] Xiong, Z.; Radhakrishnan, R.; Divakaran, A.; aj.: Comparing MFCC and MPEG-7 audio features for feature extraction, maximum likelihood HMM and entropic prior HMM for sports audio classification. In *ICME*, IEEE Computer Society, 2003, ISBN 0-7803-7965-9, s. 397–400.

[12] Bello, J. P.; Daudet, L.; Abdallah, S.; aj.: A tutorial on onset detection in music signals. In *IEEE Transactions in Speech and Audio Processing*, 2004.

[13] Orio, N.: Music Retrieval: A Tutorial and Review. *Foundations and Trends® in Information Retrieval*, ročník 1, č. 1, 2006: s. 1–90, ISSN 1554-0669, doi:10.1561/1500000002. Dostupné z: http://dx.doi.org/10.1561/1500000002

[14] Doraisamy, S.; Ruger, S.: A polyphonic music retrieval system using Ngrams. In *Proceedings of the International Conference on Music Information Retrieval*, 2004, s. 204–209.

[15] Ghias, A.; Logan, J.; Chamberlin, D.; aj.: Query by Humming: Musical Information Retrieval in an Audio Database. In *Proceedings of the Third ACM International Conference on Multimedia*, MULTIMEDIA '95, New York, NY, USA: ACM, 1995, ISBN 0-89791-751-0, s. 231–236, doi:10.1145/217279.215273. Dostupné z: http://doi.acm.org/10.1145/217279.215273

[16] Parsons, D.: *The Directory of Tunes and Musical Themes*. S. Brown, 1975.

[17] Bainbridge, D.; C.G. Nevill-Manning, I. W.; L.A. Smith, R.: Towards a digital library of popular music. In *Proceedings of the ACM Conference on Digital Libraries*, 1999, s. 161–169.

[18] BYDŽOVSKÁ, H.: Podobnost digitálních obrázků pomocí Earth Mover's Distance [online]. 2007 [cit. 2016-05-09]. Dostupné z: DostupnézWWW<http://is.muni.cz/th/139544/fi_b/>

[19] Müller, M.: *Information Retrieval for Music and Motion*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007, ISBN 3540740473.

[20] Zezula, P.; Amato, G.; Dohnal, V.; aj.: *Similarity Search: The Metric Space Approach*. Springer Publishing Company, Incorporated, první vydání, 2010, ISBN 1441939725, 9781441939722.

[21] Wang, A.: An Industrial Strength Audio Search Algorithm. In *ISMIR 2003, 4th International Conference on Music Information Retrieval*, 2003. Dostupné z: http://www.ee.columbia.edu/~{}dpwe/papers/Wang03-shazam.pdf

[22] Bogdanov, P.; Wack, N.; Gómez, E.; aj.: *ESSENTIA: an Audio Analysis Library for Music Information Retrieval.* International Society for Music Information Retrieval Conference (ISMIR'13)., 493-498.

[23] Tzanetakis, G.; Cook, P.: MARSYAS: A Framework for Audio Analysis. *Org. Sound*, ročník 4, č. 3, Prosinec 1999: s. 169–175, ISSN 1355-7718, doi: 10.1017/S1355771800003071. Dostupné z: http://dx.doi.org/10.1017/S1355771800003071

[24] McFee, B.; Raffel, C.; Liang, D.; aj.: *librosa: Audio and Music Signal Analysis in Python.* Proc. of the 14th PYTHON in science conf. (SCIPY 2015).

[25] Mcennis, D.; Mckay, C.; Fujinaga, I.; aj.: jAudio: An Feature Extraction Library. In *Proceedings of the 6th International Conference on Music Information Retrieval*, London, UK, September 11-15 2005, http://ismir2005.ismir.net/proceedings/2103.pdf.

[26] Six, J.; Cornelis, O.; Leman, M.: TarsosDSP, a real-time audio processing framework in Java. In *Proceedings of the 53rd AES Conference*, editace K. Brandenburg; M. Sandler, 2014, s. AES53–0022–1–AES53–0022–7. Dostupné z: http://0110.be/posts/TarsosDSP_Paper_and_Presentation_at_AES_53rd_International_conference_on_Semantic_Audio

[27] The Echo Nest: Echo Nest API Overview. 2011. Dostupné z: http://developer.echonest.com/docs/v4/index.html

[28] Selfridge-Field: Conceptual and representational issues in melodic comparison. *Melodic Similarity – Concepts, Procedures, and Applications*, 1998.

[29] Gómez, E.; Klapuri, A.; Meudic, B.: Melody Description and Extraction in the Context of Music Content Processing. *Journal of New Music Research*, ročník 32, 2003, content Based Retrieval. Dostupné z: http://mtg.upf.edu/files/publications/jnmr32-egomez.pdf

[30] Wei, Y. K.; Chai, W.; Garcia, R.; aj.: Analysis of a Contour-based Representation for Melody. In *In Proceedings of International Symposium on Music Information Retrieval*, 2000.

[31] De Roure, D. C.; Blackburn, S. G.: Content-based Navigation of Music Using Melodic Pitch Contours. *Multimedia Syst.*, ročník 8, č. 3, Říjen

2000: s. 190–200, ISSN 0942-4962, doi:10.1007/s005300000044. Dostupné z: http://dx.doi.org/10.1007/s005300000044

[32] Lindsay, A. T.: *Using contour as a mid-level representation of melody*. Diplomová práce, Massachusetts Institute of Technology, Program in Media Arts: Sciences, 1996.

[33] Brown, J. C.: Musical fundamental frequency tracking using a pattern recognition method. *The Journal of the Acoustical Society of America*, ročník 92, č. 3, 1992: s. 1394–1402, doi:http://dx.doi.org/10.1121/1.403933. Dostupné z: http://scitation.aip.org/content/asa/journal/jasa/92/3/10.1121/1.403933

[34] Logan, B.; Salomon, A.: A music similarity function based on signal analysis. In *Multimedia and Expo, 2001. ICME 2001. IEEE International Conference on*, 2001, s. 745–748. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1237829

[35] Aucouturier, J.-J.; Pachet, F.: Music Similarity Measures: What's the use? In *Proceedings of the Third International Conference on Music Information Retrieval*, Paris, France, October 13-17 2002, s. 157–163, http://ismir2002.ismir.net/proceedings/02-FP05-2.pdf.

[36] Mandel, M. I.; Ellis, D. P. W.: Song-Level Features and Support Vector Machines for Music Classification. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, editace J. D. Reiss; G. A. Wiggins, Září 2005, s. 594–599.

[37] Pohle, T.; Schnitzer, D.; Schedl, M.; aj.: On Rhythm and General Music Similarity. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, Kobe, Japan, October 26-30 2009, s. 525–530, http://ismir2009.ismir.net/proceedings/OS6-1.pdf.

[38] Pampalk, E.; Rauber, A.; Merkl, D.: Content-based Organization and Visualization of Music Archives. In *Proceedings of the Tenth ACM International Conference on Multimedia*, MULTIMEDIA '02, New York, NY, USA: ACM, 2002, ISBN 1-58113-620-X, s. 570–579, doi:10.1145/641007.641121. Dostupné z: http://doi.acm.org/10.1145/641007.641121

[39] Pavlíček, M.: Android v únoru: Lollipop na špici a probouzející se Marshmallow. 2016. Dostupné z: http://mobilenet.cz/clanky/android-v-unoru-lollipop-na-spici-a-probouzejici-se-marshmallow-29731

[40] Potel, M.: MVP: Model-View-Presenter: The Taligent Programming Model for C++ and Java. 1996. Dostupné z: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

[41] Salvador, S.; Chan, P.: Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, ročník 11, č. 5, Říjen 2007: s. 561–580, ISSN 1088-467X. Dostupné z: http://dl.acm.org/citation.cfm?id=1367985.1367993

[42] Google, A. D.: Application Fundamentals. 2016. Dostupné z: http://developer.android.com/guide/components/fundamentals.html

[43] Google, A. D.: Activities. 2016. Dostupné z: http://developer.android.com/guide/components/activities.html

[44] Google, A. D.: Fragments. 2016. Dostupné z: http://developer.android.com/guide/components/fragments.html

[45] Google, A. D.: Providing Resources. 2016. Dostupné z: http://developer.android.com/guide/topics/resources/providing-resources.html

[46] Google, A. D.: Processes and Threads. 2016. Dostupné z: http://developer.android.com/guide/components/processes-and-threads.html

[47] Google, A. D.: AsyncTask. 2016. Dostupné z: http://developer.android.com/reference/android/os/AsyncTask.html

[48] de Cheveigné, A.; Kawahara, H.: YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, ročník 111, č. 4, 2002: s. 1917–1930. Dostupné z: http://recherche.ircam.fr/equipes/pcm/cheveign/pss/2002_JASA_YIN.pdf

Příloha A

Seznam použitých zkratek

API Application interface

BPM Beats per minute

DTW Dynamic Time Warping

EMD Earth Movers Distance

GUI Graphical user interface

HTML Hypertex markup language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Enviroment

ISMIR International Society for Music Information Retrieval

JSON JavaScript object notation

MFCC Mel-frequency cepstrum

MIDI Musical Instrument Digital Interface

MIR Music Information Retrieval

MIREX Music Information Retrieval Evaluation eXchange

MPEG Moving picture experts group

MVC Model-View-Controller

MVP Model-View-Presenter

REST Representational State Transfer

SDK Software Development Kit

UI User Interface

UX User Experience

XML Extensible markup language

ZCR Zero-crossing-rate

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `PianoSimilarity.apk` ..... instalační balíček mobilní aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `uml` ..... UML diagramy
  - `documentation` ..... vygenerovaná dokumentace
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF