České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Tisančín**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Aproximace zvuku resyntézou**

Pokyny pro vypracování:

Prozkoumejte oblast resyntézy zvuku, tj. syntézy zvuku s cílem napodobit předložený cílový zvuk. Předpokládejte využití virtuálních (softwarových) syntezátorů ve standardu VST. Navrhněte a realizujte vlastní systém automatické resyntézy zvuku založený na prohledávání (optimalizaci) parametrů syntezátoru např. pomocí genetického algoritmu. Tento systém implementujte na zvolené počítačové platformě a následně posuďte úspěšnost řešení sadou porovnání alespoň 10 různých vstupních a výstupních zvuků. Ze zřejmých důvodů se předpokládá jen omezená věrnost této resyntézy. Návrh, implementaci, i rozsah práce konzultujte s vedoucím práce.

Seznam odborné literatury:

Ladefoged, P (1996) Elements of Acoustic Phonetics. University of Chicago Press.
Snoman R. (2008) Dance Music Manual: Tools, Toys, and Techniques Paperback. Focal Press.

Vedoucí: Ing. Adam Sporka, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

[redacted]
prof. Ing. Jiří Žára, CSc.
vedoucí katedry

L.S.

[redacted]
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 24. 3. 2015

České vysoké učení technické Praha
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

# Aproximace zvuku resyntézou

**Květen 2015**

Student: Bc. Tomáš Tisančín
Vedoucí práce: Ing. Adam Sporka, Ph.D.

## Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona 4. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Přezleticích 10. května 2015 .....

Podpis autora práce

# Poděkování

Rád bych tímto poděkoval Ing. Adamovi Sporkovi, Ph.D. za to, že mi vždy vyšel vstříc, za jeho odborné rady a pomoc a za nadšení při vytváření této diplomové práce. Rád bych také poděkoval rodině, přátelům a kolegům, kteří mě po celou dobu studia podporovali.

# Abstrakt

Resyntéza zvuku je obtížným a časově náročným úkolem, který vyžaduje hluboké znalosti daného syntetizéru i syntézy obecně. Syntetizérů je nepřeberné množství a každý je svým způsobem odlišný jak svými parametry, tak zvukem, který produkuje. V této práci využíváme genetického algoritmu k napodobení předloženého cílového zvuku pomocí automatické optimalizace parametrů daného syntetizéru.



# Klíčová slova

Resyntéza, Zvuk, GA, MFCC, Porovnávání zvuků, VST, DTW



# Abstract

Sound resynthesis is complex and time-consuming task, which requires deep knowledge of used synthesizer and synthesis in general. There is countless number of synthesizers and each one is different in both its parameters and sound it produces. In this thesis we use Genetic Algorithm to match target sound using automated optimalization of parameters of given synthesizer.



# Key words

Resynthesis, Sound, GA, MFCC, Sound matching, VST, DTW

# Obsah

# Seznam obrázků

# Seznam tabulek

# Kapitola 1

# Úvod

První syntetizéry si v oblasti tvorby zvuků a hudby našly své místo již v roce 1967, kdy byl do světa vypuštěn syntetizér Moog, který byl vytvořen tak, aby zpopularizoval celé odvětví. Postupem času začaly vznikat nové a nové techniky syntézy, které rozšířili škálu syntetizovaných zvuků. Úplně jiný rozměr dostala tato expanze v roce 1996, kdy syntetizéry dostaly svou softwarovou podobu ve formě *Virtual Synthesizer Technology* (VST).

Vytvořit či alespoň napodobit konkrétní, již existující zvuk, je pro zvukaře časově náročný problém, jelikož je nucen ručně nastavovat parametry syntetizéru. Dokonalá znalost konkrétního syntetizéru, ale i znalost syntetizérů obecně je v takovém případě nutností. Počet parametrů některých syntetizérů se navíc pohybuje i řádu stovek. Typické jsou však syntetizéry s 20-40 parametry. Význam hodnot parametrů navíc může být často nelineární a matematicky komplexní, takže parametry jsou na svých hodnotách navzájem závislé. Uživatel může chtít resyntézou dosáhnout přesné reprodukce zvuku, mnohem častěji mu však jde o prozkoumání možností syntetizéru a reprodukci zvuku podobného.

## 1.1 Obsah a cíle práce

Počátek práce je teoretickým úvodem do oblasti zvuku a jeho syntézy. Jsou zde objasněny některé pojmy, na které později práce navazuje. Zároveň je zde také rozebráno několik předchozích prací na podobné téma a taktéž jsou zde popsány existující metody a možnosti. Druhá část práce se zaměřuje na konkrétní řešení problému resyntézy zvuku pomocí automatického nastavení parametrů konkrétního syntetizéru. K tomu je využit vlastní genetický algoritmus (GA), který optimalizuje hodnoty všech parametrů zároveň.

Výsledek práce (resp. program) by měl sloužit jak amatérům, tak i profesionálům v oblasti syntézy zvuku či hudby. Program může být dobrým návodem na pochopení procesů syntézy. V uživatelském rozhraní je možno sledovat změny parametrů syntetizéru vůči změnám ve zvuku a odvodit si význam jednotlivých parametrů. Program může posloužit jako pomůcka při snaze o syntézu zvuku, který se uživateli líbí, a pouze neví, jaký syntetizér by byl pro syntézu vhodný, či jak nastavit správně parametry syntetizéru. Zkušeným zvukařům dále nabízí generování škály zvuků, které jsou některému konkrétnímu zvuku podobné.

Program je částečně omezen nutností zadávat vstupy ve formě syntetizéru a MIDI (resp. not). Taktéž se předpokládá pouze určitá omezenost a robustnost celého systému, jelikož v současnosti neexistuje automatická metoda, která by dokázala spolehlivě porovnat dva zvuky. Úkolem této práce je tedy navrhnout, dobré, nikoliv dokonalé, řešení tohoto problému.

Navržené řešení je nakonec otestováno na několika vzorcích. Práce zároveň obsahuje popis a výsledky experimentů s nastavováním parametrů využitých metod.

# Kapitola 2

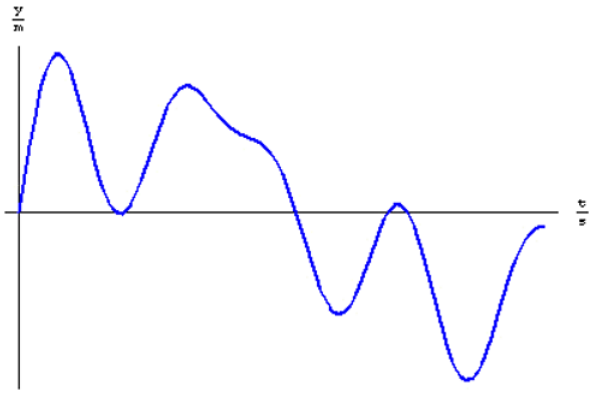
# Pozadí

V této kapitole rozebereme některé nezbytné znalosti a pojmy, které se týkají syntézy zvuku. Také se podíváme na tři existující práce zabývající se podobným tématem. V dalších částech práce se budeme na fakta zde uvedená odkazovat.

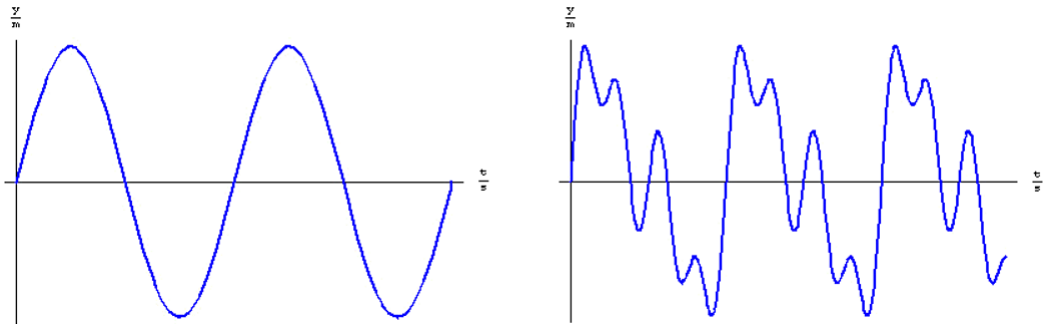
## 2.1 Zvuk

Z fyzikálního pohledu je zvuk uspořádaný kmitavý pohyb molekul. Tento pohyb je přenášen působením vzájemných sil a nazývá se zvuková vlna. Frekvence tohoto vlnění určuje frekvenci zvuku, která definuje výšku tónu. Frekvence, které je schopen člověk vnímat se značně individuálně liší, nicméně můžeme říci, že leží v intervalu 16 Hz až 20 000 Hz [1].

Zvuky lze obecně rozdělit do dvou základních skupin: tóny a hluky. Tóny mají periodický průběh okamžité výchylky závislý na čase, kdežto hluky ne (viz 2.1 a 2.2). Dále lze tóny dělit na jednoduché a složené. Jednoduché tóny mají harmonický průběh, kde je časový průběh sinusoida. U složených tónu toto neplatí. [2]



Obrázek 2.1: Neperiodický průběh zvuku (hluk) [2]



Obrázek 2.2: Periodický průběh zvuku (tóny) [2]

### 2.1.1 Tóny

Matematický zápis jednoduchího tónu o frekvenci $f$ je [2]:

$$y = y_m sin\omega t, \omega = 2\pi f \tag{2.1}$$

Složený tón je obdobný, jen obsahuje více členů (předpokládá se $f_1 < f_2 < \dots f_n$ a $y_{m_1} > y_{m_2} > \dots y_{m_n}$) [2]:

$$y = y_{m_1} sin2\pi f_1 t + y_{m_2} sin2\pi f_2 t + \cdots + y_{m_n} sin2\pi f_n t \tag{2.2}$$

Frekvence $f_1$ určuje výšku složeného tónu a nazývá se fundamentální frekvencí. Ostatní se nazývají vyšší harmonické (spolu s fundamentální pouze harmonické). Lidské sluchové ústrojí nevnímá jednotlivé tóny, ale pouze tón složený jako jeden celý tón, který má určitou barvu, kterou mu dodávají vyšší harmonické frekvence. Pomocí Fourierovy transformace lze získat spektrum frekvencí složeného tónu.

## 2.2 MIDI

MIDI je protokol, pomocí něhož je možné přenášet informace mezi elektronickými nástroji a dalšími hudebními digitálními nástroji. Těmito informacemi jsou myšleny události, jež mohou dít na MIDI klávesách. Především jde o stisky not a různá nastavení.

**MIDI událost** MIDI událost je dvojice hodnot. Jednou z hodnot je tzv. MIDI Message. Druhou hodnotou je *tick*. *Tick* určuje, kdy má být MIDI message zpracován. Jde o nejkratší časový úsek, který lze v MIDI reprezentovat. Aby se s těmito hodnotami dalo jednoduše pracovat, je nutné je převést na milisekundy. Převod je přímo závislý na *beats per minute* BPM (tempo) a *pulses per quater* PPQ. PPQ je hodnota rozlišení MIDI a udává, kolik "pulzů"je v jedné čtvrťové notě. Výpočet z tick na milisekundy se řídí následujícím výpočtem:

$$ms = 60/BPM * (tick/PPQ) * 1000 \tag{2.3}$$

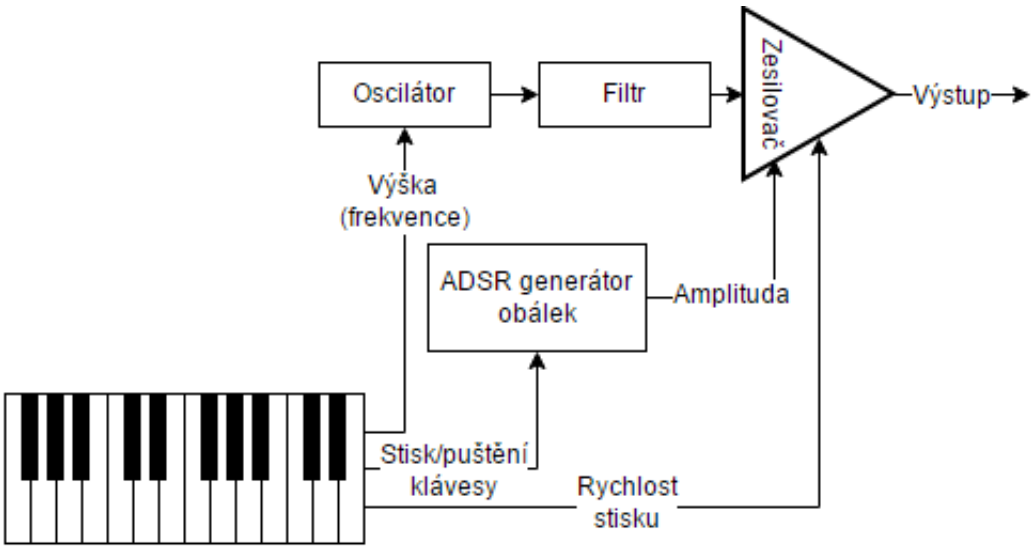
**MIDI zpráva** MIDI zpráva nese hlavní informaci události a existuje hned několik typů. Mezi základní patří: *Note Off, Note On* a *Control Change*. *Note Off* a *Note On* přepínají stavy jednotlivých not. V tuto chvíli je pro nás však důležitější *Control*

*Change*, ze kterého je možné vyčíst hodnotu BPM. Hodnotu PPQ je možno přečíst z hlavičky souboru. [3]

## 2.3 Zvukový syntetizér

Zvukový syntetizér je zařízení, které vytváří zvuk přesným určením fundamentálních vlastností, jako jsou výška, barva a hlasitost. Syntetizéry jsou schopné emulovat, tedy syntetizovat, širokou škálu zvuků. Syntetizérů existuje několik typů (viz sekce 2.3.2). Tyto typy se liší tím, jak vytváří zvuky, resp. složením jeho vnitřních součástí a jejich propojením.

Mezi základní součásti, které může syntetizér obsahovat, patří: oscilátor, filtr, generátor obálky, modulátor, zesilovač, atd. Vstupem syntetizéru je jeho konfigurace a posloupnost MIDI tónů s určitou délkou a rychlostí stisku. Výstupem je zvukový signál o jednom nebo několika kanálech.




Obrázek 2.3: Příklad syntetizéru - diagram [4]

### 2.3.1 Součásti syntetizéru

Jak již bylo řečeno, syntetizér se skládá z několika propojených částí. Ty základní zde stručně popíšeme, jelikož znalost jejich funkce je nutná pro práci se syntetizéry.

**Oscilátor** Jde o naprosto základní součást syntetizéru, někdy nazývaný jako generátor tónů. Pomocí něho syntetizér tvoří zvuky. Výstupem je signál o nějakém

konkrétním tvaru, který se opakuje. Frekvence opakování pak jako jeden z faktorů definuje výšku syntetizovaného zvuku. Mezi základní tvary signálu patří: sawtooth, triangle, square, sine (viz B.1). Zmíněné tvary určují mimo jiné barvu syntetizovaného zvuku. Mnoho syntetizérů také pracuje s náhodnými tvary, které se projevují jako šum.

**Filtr** Filtr umožňuje v signálu zesílit nebo zeslabit, popř. úplně odstranit, některé frekvence a dovoluje tak zaoblit původní tvar signálu. Například pokud oscilátor generuje sawtooth, pak má zvuk mnoho podtónů [1]. Množství podtónů a jejich intenzita je dána ostrostí hran signálu. Pokud tedy na tento tvar signálu aplikujeme správný filtr, můžeme tak průběh zaoblit, čímž je možné získat sinusoidu.

**Zesilovač** Jak již název napovídá, zesilovač slouží k zvětšení či změnšení amplitudy signálu a tedy i k zesílení či zeslabení hlasitosti syntetizovaného zvuku.

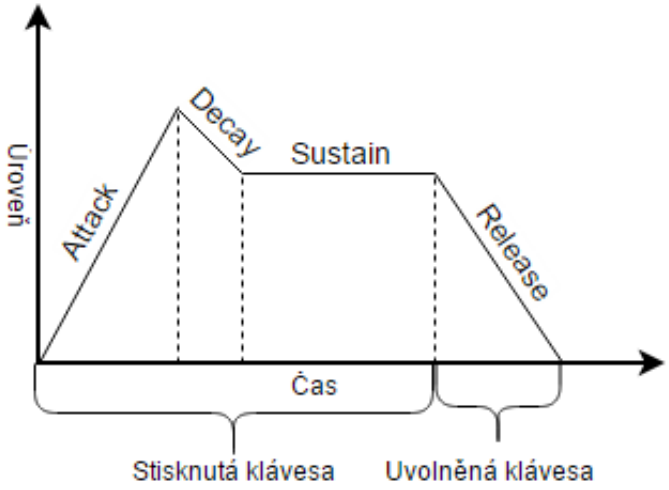
**Obálka** Aby zněl výsledný zvuk přirozeně je třeba upravit hlasitost (popř. frekvenci) vzhledem k časovému průběhu. Samozřejmě toto není povinnou součástí syntetizéru, ale pokud chceme, aby měl výsledný zvuk nějaký postupný nástup, průběh a útlum, je třeba využít právě obálky. Takové zvuky jsou pak více reálné a nezní staticky. Typicky má vlastní obálku oscilátor, filtr i zesilovač (popř. další součásti), všechny však mohou být závislé na stejném nastavení. Obálka oscilátoru typicky mění frekvenci syntetizovaného zvuku, obálka zesilovače mění hlasitost a obálka filtru může měnit barvu, vše samozřejmě v průbehu času.

Základní části obálky jsou známy jako **ADSR**:

- **A**ttack - nástup od 0 do maxima (po stisknutí klávesy)
- **D**ecay - první lehké odeznění (po uplynutí krátké doby od stisknutí klávesy)
- **S**ustain - držení úrovně (při držení klávesy)
- **R**elease - últum do 0 (začíná po puštění klávesy)

**Nízkofrekvenční oscilátor** Podobně jako oscilátor, generuje i nízkofrekvenční oscilátor signál konkrétního tvaru, který se stále opakuje. Rozdílem je frekvence, ve které se daný tvar opakuje. Zatímco klasický oscilátor pracuje ve vysokých frekvencích, aby vytvořil základní zvuk, nízkofrekvenční slouží k modulaci zvuku a proto pracuje v nízkých frekvencích (jednotky, maximálně desítky Hz).

[1] Podtón je jakákoli frekvence ve zvuku, která je vyšší než jeho fundamentální frekvence. Vyšší harmonické tóny jsou tedy součástí podtónů, podtóny však mohou být i neharmonické.




Obrázek 2.4: Základní fáze obálky

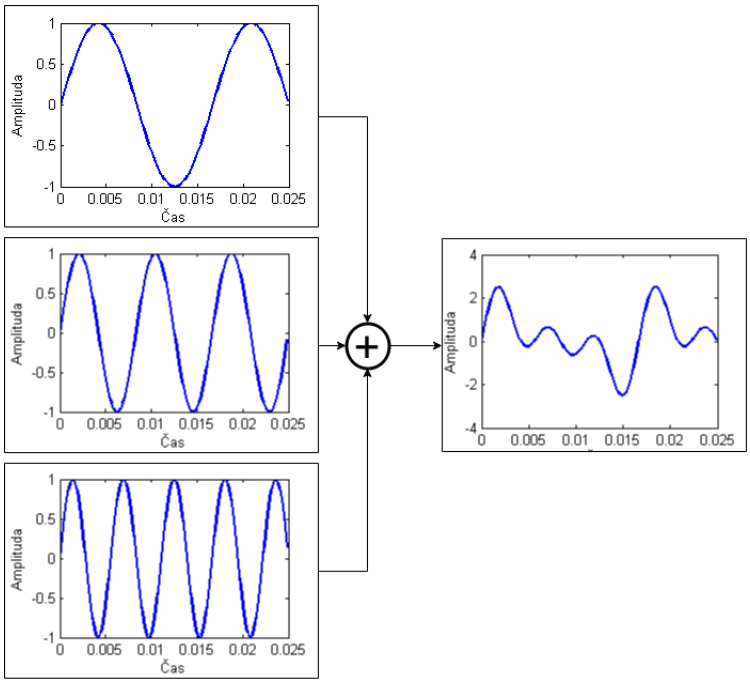
Lze ho připojit k ostatním součástem syntetizéru, podobně jako obálku. Pokud je nízkofrekvenční oscilátor připojen k oscilátoru, obvykle mění výšku tónu v čase. Výsledkem může být například typický tón sirény, který postupně frekvenčně kolísá. U filtru dokáže měnit filtrovanou frekvenci a u zesilovače výslednou hlasitost.

### 2.3.2 Metody syntézy zvuku

Existuje mnoho technik syntézy zvuku, které dodávají různým syntetizérům jejich jedinečnost. Vybrané metody, které syntetizéry využívají jsou:

**Aditivní syntéza** Základní druh syntézy, kdy dochází ke skládání několika vlnových komponent dohromady. Specifickou aditivní syntézou je tzv. Fourierova syntéza, která vytváří zvuk pomocí skládání sinusových vln. Pokud skládáme dostatečně malé části, můžeme tak teoreticky vytvořit jakýkoli zvuk, reálně to však přiliš není.

**Subtraktivní syntéza** Přesný opak aditivní syntézy. Tato metoda se využívá u většiny analogových i digitálních syntetizérů, ale vyskytuje se i u jiných typů. Odečítání signálu je docíleno pomocí filtrace. Na rozdíl od aditivní syntézy je subtraktivní mnohem jednodušší, vzhledem k tomu, že je reprezentována filtry. A navíc díky vysoké zvukové účinnosti se tato metoda stala jednou z nejoblíbenějších. Fyzikální princip této syntézy se shoduje s přirozeným vznikem zvuků v hudebních nástrojích nebo při tvorbě zvuku lidského hlasu.



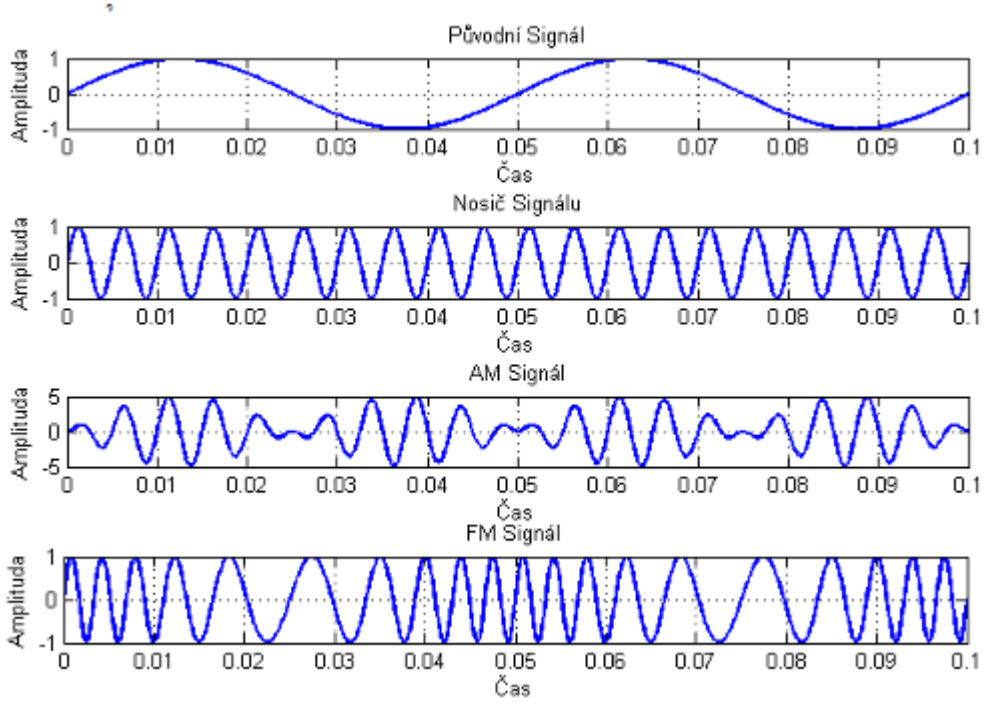

Obrázek 2.5: Jak funguje aditivní syntéza

**Modulační syntéza** Tato metoda funguje na základě nelineárnosti modulačního procesu, kdy vznikají nové harmonické složky úměrné součtům a rozdílům frekvencí již přítomných . Do této kategorie spadá frekvenční (FM), amplitudová (AM) a kruhová modulace. Zatímco frekvenční modulační syntéza se používá autonomně, amplitudová a kruhová se používají spíše jako doplňkové efekty k jiným typům syntéz.

Frekvenční modulační syntéza mění přímo frekvenci signálu na základě modulační vlny, kdežto amplitudová mění amplitudu signálu (viz 2.6). Kruhová modulační syntéza vytváří zvuk spojením součtu a rozdílu nosné a modulační vlny. Modulární syntetizéry se často využívají při snaze o reprodukci reálných zvuků.

**Tvarová syntéza** Na rozdíl od předchozích metod využívá tvarová syntéza operace v časové oblasti, které mění tvar signálu. Základní metodou tvarování je průchod signálu obvodem s nelineární charakteristikou. Díky nelinearitě je možné dosáhnout nových harmonických složek. Dále je možné přímo zadávat časový průběh například pomocí grafického rozhraní a jednoduše tak ovlivnit tvar průběhu. Nevýhodou této metody je prakticky nulová kontrola výsledného vyznění. Dále můžeme zmínit fázové zkreslení, které přehází jednotlivé vzorky signálu v diskrétní podobě a vznikne tak úplně nový tvar vlny s jiným spektrem.

**Slučovací syntéza** Segmentační, granulační a formantová syntéza. Tyto metody mají společné to, že syntetizují zvuky za pomoci skládání malých zvukových jednotek,




Obrázek 2.6: Jak funguje FM a AM syntéza zvuku

které jsou dlouho pouze několik málo milisekund, do větších celků. Tyto malé jednotky se navzájem liší délkou, původem a složitostí. Často tak lze vytvořit zajímavé zvukové efekty. Segmentační metoda využívá skládání různých segmentů zadaných křivek, zatímco granulační syntéza používá pro generování zvuků krátkých úseků jiného zvuku, tzv. granulek. Tyto granulky se přehrávají za sebou velmi rychle, takže je lidský mozek vnímá jako spojitý zvuk. Poslední zmíněnou syntézou je formantová. Ta se využívá výhradně při zpracování řečových signálů. Signál se rozdělí na malé kousky a následně je tak možné syntetizovat jakoukoli větu.

**Fyzikální modelování** U této metody jde o nový způsob syntézy, se kterou přišla firma YAMAHA v roce 1993. Jak už název napovídá, jde o modelování fyzikálních vlastností hudebních nástrojů. Jde tedy o maximální množství údajů, které nástroj popisují (např. materiál, tvar a jiné specifické údaje pro různé nástroje). Tyto parametry jsou následně použity v matematických rovnicích, které se podílí na výsledném signálu.

**Wavetable syntéza** Tento druh syntézy vytváří zvuky pomocí krátkých cyklických vzorků. Průběh těchto vzorků je různý, od jednoduchých vln a křivek, až po komplexní. Tyto vzorky jsou uloženy v tabulce syntetizéru a mohou být kombinovány, editovány a obohaceny tak, aby vytvořily zvuk.

## 2.4 Virual Studio Technology

*Virtual Studio Technology* (VST) je softwarové rozhranní pro integraci zvukových syntetizérů a efektových pluginů. Systém byl vytvořen firmou Steinberg v roce 1996. VST se často používá jako náhrada reálného analogového syntetizéru. Virtuální syntetizéry však nedokážou plně nahradit ty klasické analogové, namísto toho ale nabízí další zajímavé možnosti syntézy, kterých by jinak nebylo možné dosáhnout. VST nefunguje samostnaně, aby bylo možné ho využívat, je třeba použít VST host. Ten s VST pracuje jako s černou skříňkou, které má nějaké vstupy a nějaké výstupy, procedury uvnitř této skříňky mu nejsou známé.

VST se dělí na dva typy, efekty a instrumenty. Efekty upravují již existující zvuk a instrumenty vytvářejí zvuk úplně nový a to pomocí MIDI zpráv. V této práci je předpokladem, že použitý syntetizér je instrumentem. Pokud by se mělo jednat o efekt, nebyla by resyntéza ze zřejmých důvodů vůbec možná.

## 2.5 Existující práce

Většina dosavadních prací s tématem resyntézy zvuku se zaměřují na resyntézu pomocí vlastní implementace FM syntetizérů. Ke zjištění parametrů syntetizérů využívají převážně **genetické algoritmy** (GA), jelikož právě ty se ukázaly jako nejvhodnějším nástrojem pro řešení problému optimalizace parametrů.

### 2.5.1 2011, Johnson

Tento projekt [5] čerpá z prací Yonga [6] a Laie [7] na které je často odkazováno. Pro otimalizaci je využito GA a selekce pomocí **turnajové metody**. Johnson využívá vlastní FM syntetizér, který by měl být teoreticky schopný syntézy jakéhokoli zvuku (jak syntetizovaných, tak reálných), toho ale prakticky nelze dosáhnout. Pro porovnávání zvuku využívá **spektrálních norem** a **centroidů**.

Jak však Johnson v závěru uvádí, nepodařilo se mu tímto způsobem reprodukovat žádný zvuk korektně. Nejlépe reprodukovatelné byly zvuky, které byly jeho syntetizérem vytvořené. Dále také uvádí, že **některé zvuky jsou lépe reprodukovatelné než ostatní**. Ač nebylo dosaženo velkého úspěchu, Johnson podotýká, že výsledek je možné použit pro zkoumání zvuků, resp. reprodukci nepřesných, ale podobných zvuků. I přes menší neúspěch je jeho projekt dobrým odrazovým můstkem při vytváření dalších prací.

### 2.5.2 1996, Tan

Tento paper [8] se zaměřuje na resyntézu pomocí dvojité frekvenční modulace (DFM), která je odvozena od klasické FM. Jde tedy opět o resyntézu na vlastním syntetizéru. K optimalizaci parametrů využívá **genetický algoritmus** propojený se **simulovaným žíháním** (SA - Simulated Annealing) a komnbinuje výhody obou těchto algoritmů. Zatímco GA se hodí pro prohledávání velkého prostoru a je méně časově náročný, SA se soustředí na prohledávání menších prostorů. SA je tedy schopno najít lepší řešení než GA, ale zpracování trvá déle, jelikož se algoritmus hůře dostává z lokálních minim.

Výsledek této práce je otestován na několika pokusech o syntézu zvuků různých instrumentů jako je hoboj, francouzský roh a saxofon. Porovnávání je zde však provedeno pouze podle **harmonických frekvencí**, které se jeví jako úspěšné. Vyzdvihována je tu především kombinace obou algoritmů, která má být efektivnější, než při použití algoritmů samostatných.

### 2.5.3 2008, McDermott

Tato práce [9] je mému zadání asi nejbližší, jelikož je založena na syntéze na daném syntetizéru (XSynth) o 32 parametrech, přičemž je zde předpoklad, že cílové zvuky byly syntetizovány právě tímto syntetizérem. Využíván je zde taktéž **genetický algoritmus** a porovnávání zvuků je provedeno pomocí **perceptuálních metrik**, ale také pomocí porovnání surového signálu, diskrétní furierovy transformace a porovnání parametrů syntetizéru. McDermott dále experimentuje s nastavením GA.

# Kapitola 3

# Genetický algoritmus

Nejprve by bylo vhodné odůvodnit, proč byl pro aproximaci parametrů syntetizérů zvolen právě genetický algoritmus. Výběru předcházela rešerše, která se zaměřila na prozkoumání existujících řešení v této oblasti. Prakticky všechny práce, které mají za cíl resyntézu zvuku pomocí aproximace parametrů, využívají genetické algoritmy.

Příkladem může být A. Johnson [5], S. Yong [6] nebo V. Välimäki [10]. Tyto práce mají s genetickými algoritmy velmi dobré výsledky a právě proto jsem se rozhodl jít v jejich stopách. Avšak i tak jsem každé rozhodnutí, které jsem provedl, pečlivě zvážil, odůvodnil. Dále jsem se rozhodl pokusit genetický algoritmus podpořit. Tato podpora měla spočívat ve znalosti vlivu jednotlivých genů. Jak se však nakonec ukázalo, zjistit a využít tyto znalosti není vůbec jednoduchým úkolem.

## 3.1 Jak funguje genetický algoritmus

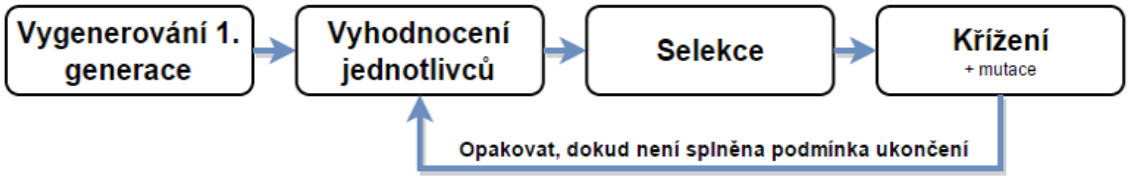
Genetický algoritmus je odvozený od principu evoluce. Genetické algoritmy se snaží najít vhodné řešení nějakého problému. Na počátku je typicky daná populace jedinců, kde každý jedinec představuje jedno řešení problému. Každý jedince má své geny, které určují, jak problém řeší. Tyto geny jsou přímo závislé na správnosti řešení (dále kvalita[1]).

V populaci je následně vyvoláno křížení, kdy dochází ke smíchání genů mezi dvojicemi (někdy i trojicemi) jedinců (rodiče). Tomuto křížení předchází selekce jedinců vhodných ke křížení. Tito jedinci jsou obvykle náhodým výběrem z jedinců, jejichž řešení je nejlepší. Z těchto vyselektovaných jedinců se následně vytváří nová generace a staří jedinci zaniknou. Někdy, pokud je to žádoucí, se může stát, že do další generace

[1]Kvalita je vlastně vzdálenost od dokonalého řešení, resp. od cíle. Čím menší je číslo označující kvalitu, tím více je jedinec kvalitní.

se dostane i jedinec z generace předchozí, tomuto jevu se říká elitismus. V nové generaci by se měly v průměru nacházet tací jedinci, jejichž řešení je lepší, než řešení jedinců z generace předešlé.

Při křížení může také dojít k mutaci genu. Mutací se rozumí událost, jež nastane s malou pravděpodobností a má za následek změnu hodnoty genu, často nezávisle na hodnotách genů rodičů, kteří se na křížení podílí. Celý proces se opakuje, dokud není nalezeno kvalitní řešení, které by odpovídalo předem definovaným požadavkům. Algoritmus je také možné zastavit po určitém počtu iterací, či po určité době vykonávání.




Obrázek 3.1: Diagram genetického algoritmu

## 3.2 Selekce jedinců z populace

Selekčních metod je pro genetické algoritmy hned několik. Jednoduchou metodou je tzv. *Vážená ruleta*, kdy je jedincům přidělen na pomyslné ruletě takový počet políček, který odpovídá kvalitě jedince (čím lepší kvalita, řím více políček). Následně je proveden náhodný výběr políček. Tato metoda je však náchylná k příliš velkému nadržování silným jedincům, kteří tak ovládnou políčka rulety. U genetického algoritmu by se nemělo stávat, že vybráni budou pouze ti nejkvalitnější jedinci, jelikož tak může snadno dojít k situaci, kdy algoritmus bude konvergovat k jednomu ne úplně správnému řešení. I méně kvalitní jedinec může obsahovat takovou hodnotu genu, která se u kvalitnějších jedinců nevyskytuje a tím by tato hodnota genu zanikla. Tomuto jevu se dá však částečně předejít, viz 3.3.1.

Lepší, než se spoléhat na záchranný mechanismus mutace, je vyhnout se tomuto jevu již při selekci. Vhodnější metoda, která nahrává i slabším jedincům, se nazývá *Stochastický univerzální výběr*. Tato metoda je velice podobná *Vážené ruletě*, ale liší se výběrem jedinců. Dále je zde také nutnost usprořádání jedinců za sebou sestupně dle kvality. Na počátku tohoto výběru je vybráno náhodné číslo od 0 do 1/N, kde N je počet jedinců, kteří mají být vybráni. Toto číslo určuje počáteční pozici ukazatele, který vybírá jednotlivce. Tento ukazatel se následně posouvá k horším jedincům přičtením konstanty 1/N. [11] [12]

Tím se dostáváme k sofistikovanějším metodám selekce, jako je metoda *Turnajů*. Pokud bych se měl řídit předchozími pracemi, tak podle Johnsona [5] a Laie [7] se pro podobné účely hodí právě tento typ selekce. Ta funguje tak, že je náhodně prove-

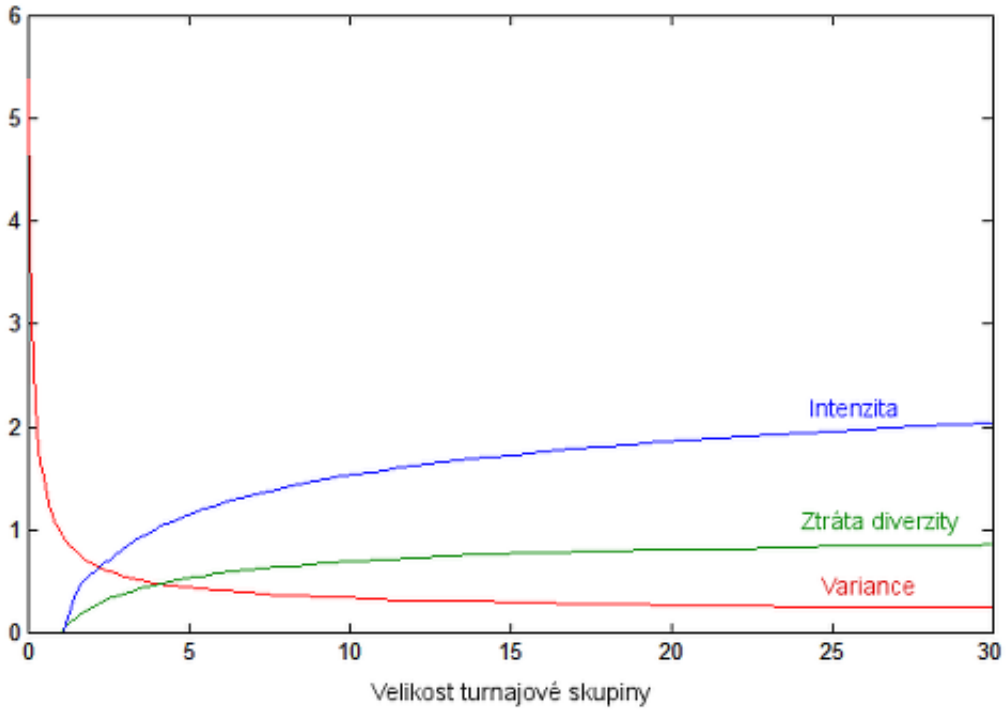
den výběr předem daného počtu jedinců a z nich je vybrán ten nejkvalitnější. Vše se opakuje, dokud není vybrán daný počet jedinců.

Tato selekční metoda má dva vstupní parametry. Jedním je velikost turnajové skupiny, druhým je počet jedinců, který se má vyselektovat. Velikost turnajové skupiny ovlivňuje přímo míru selekční intenzity ($I$), ztráta diverzity ($D$) a variance ($V$) dle následujících vzorců, kde $T$ je velikost skupiny. Následující vzorce jsou převzdaté z [12]:

$$I(T) \approx \sqrt{2 \cdot (ln(T) - ln(\sqrt{4.14 \cdot ln(T)})} \tag{3.1}$$

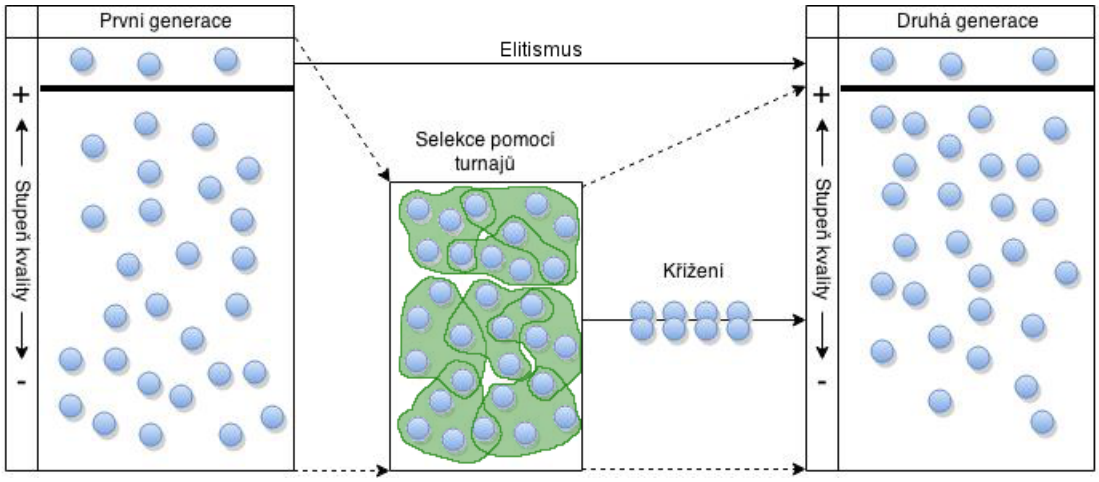
$$D(T) = T^{\frac{-1}{T-1}} - T^{\frac{-T}{T-1}} \tag{3.2}$$

$$V(T) \approx \frac{0.918}{ln(1.186 + 1.328 \cdot T} \tag{3.3}$$




Obrázek 3.2: Vliv velikosti turnajové skupiny na vlastnosti selekce

**Elitismus** Již jsem se zmínil o pojmu elitismus. Elitismus slouží pro zrychlení konvergence a pro zachování opravdu kvalitních jedinců pro další generace. V případě, že by například byl již v první generaci vytvořen jedinec, který je dostatečně kvalitní, bez elitismu by tento jedinec nepostoupil do další generace a jeho geny by byly křížením znehodnoceny. Celý proces selekce popisuje diagram 3.3.

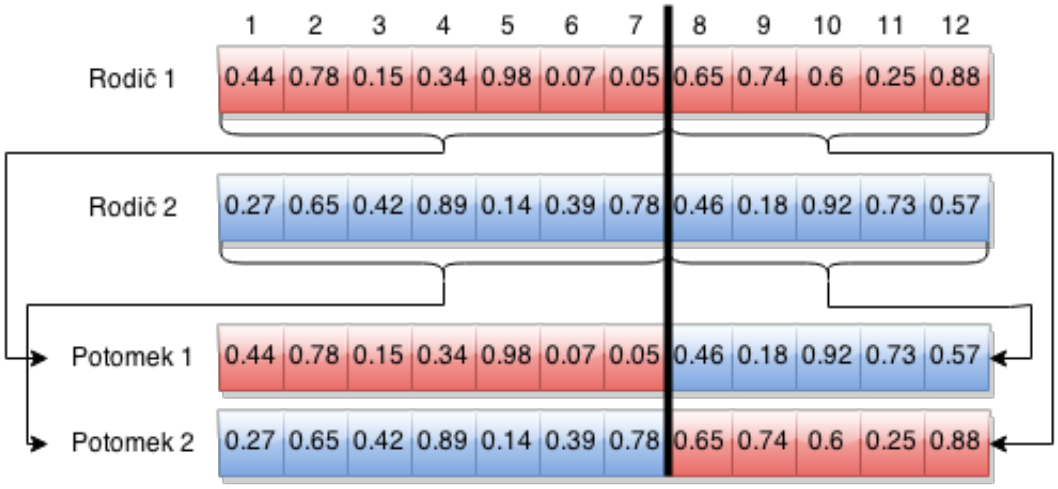



Obrázek 3.3: Diagram selekce turnajové metody

## 3.3 Křížení jedinců a vznik nové generace

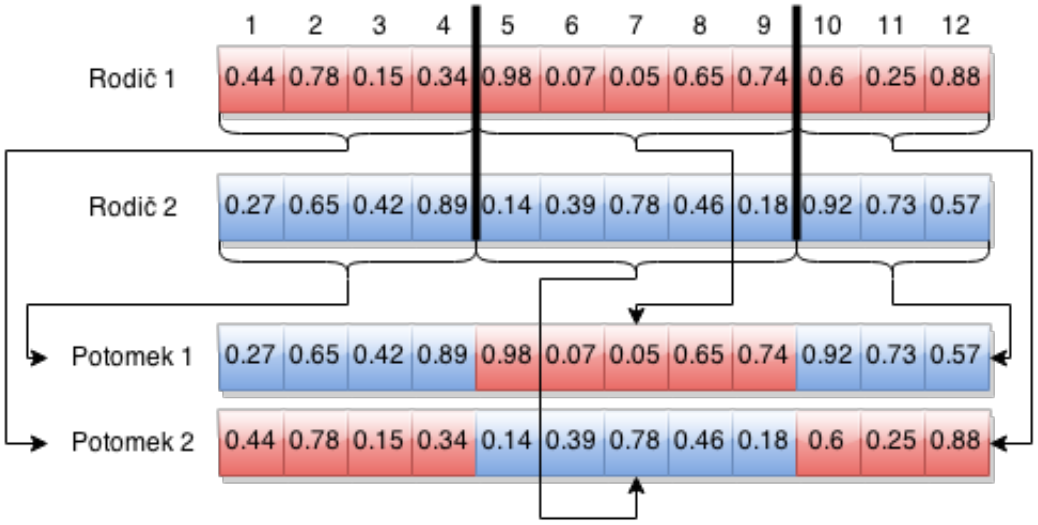
Způsobů křížení, podobně jako selekce, je několik. Liší se v komplexnosti, rychlosti a efektivnosti. Většina křížení pracuje s genem jako s řetězcem informací. V našem případě můžeme takto gen reprezentovat velice snadno a to pouhým poskládáním hodnot parametrů za sebe.

Nejjednodušším způsobem je *jednobodové křížení* (*one-point crossover*), kde dochází k náhodnému zvolení jednoho bodu v genu obou rodičů a tím dojde k rozdělení těchto genů. První část genu prvního rodiče se spojí s druhou částí genu druhého rodiče a druhá část prvního rodiče se spojí s první částí druhého rodiče. Takto mohou tedy vzniknout až dva potomci, ale je možné uvažovat pouze jednoho.




Obrázek 3.4: Jednobodové křížení

Rozšířením *jednobodového křížení* je *dvoubodové křížení* (*two-point crossover*). Místo jednoho bodu jsou náhodně zvoleny body dva. Jeden potomek tak získá počátek a konec genu jednoho rodiče a prostředek druhého rodiče a naopak. Tento typ křížení je použit v práci Johnsona [5].



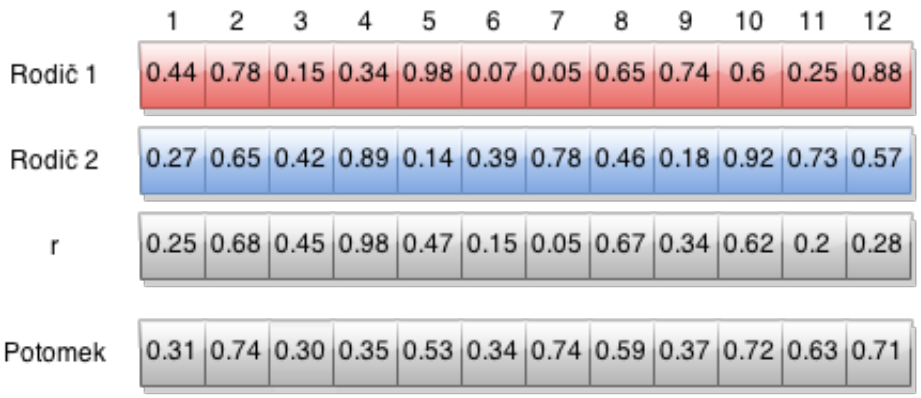

Obrázek 3.5: Dvoubodové křížení

*Vícebodové*, resp. *uniformní křížení* (*uniform crossover*) je dalším způsobem, který funguje na podobném principu, jako předchozí dvě metody. Jako vstupní parametr má pravděpodobnostní hodnotu $p$. Pro každou část genu (resp. pro každý gen) je náhodně vybrané číslo $k, 0 \leq k \leq 1$. Pokud platí $k \leq p$, tak si první potomek bere gen od prvního rodiče a druhý od druhého. V opačném případě je to naopak. Typicky se volí $p = 0.5$, aby oba potomci zdědili průměrně polovinu genů, ale však možno zvolit i hodnotu vyšší [13], aby došlo k upřednostnění jednotlivých rodičů u potomků.


Obrázek 3.6: Uniformní křížení

Zatím jsme se omezili pouze na takové metody křížení, které slučují určité části genů rodičů, ale nemění hodnoty genu jako takových. Jedna z metod, která mění hodnoty genů v závislosti na hodnotách rodičů, se nazývá *ploché křížení* (*flat crossover*). Při tomto typu křížení dochází ke generování náhodné hodnoty $0 \leq r_i \leq 1$ pro každý jednotlivý gen $i$. Tato hodnota pak určuje poměr zastoupení genu jednotlivých rodičů. Uvažujme tedy rodiče $R1 = (x_{1,1}, \ldots, x_{1,n})$ a $R2 = (x_{2,1}, \ldots, x_{2,n})$ a náhodný vektor $r = (r_1, \ldots, r_n)$. Potomek $P = (x_1, \ldots, x_2$ vznikne jako lineární kombinace [13]

$$x_i = r_i x_{1,i} + (1 - r_i) x_{2,i}, i = 1, \ldots, n. \tag{3.4}$$

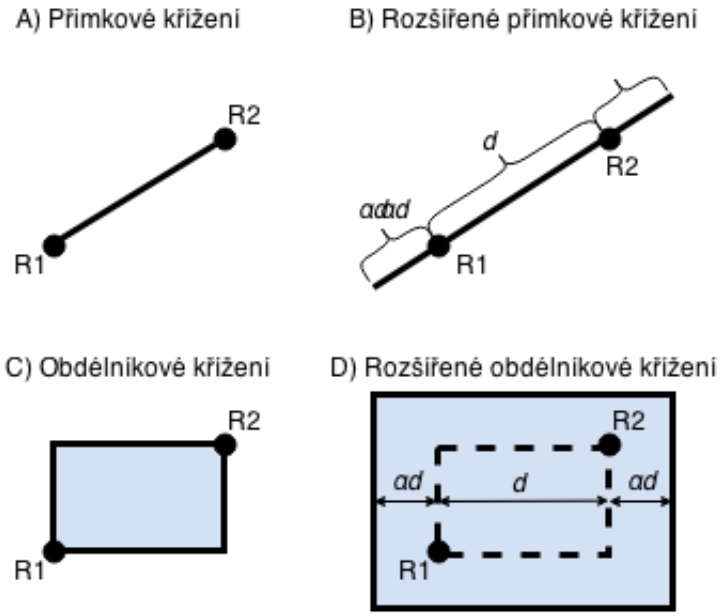



Obrázek 3.7: Ploché křížení.

Předchozí metoda se také nazývá *obdélníkové křížení* (*box crossover*) [14]. Podobně funguje také *přímkové křížení* (*line crossover*), s tím rozdílem, že se generuje pouze jedno číslo, které slouží jako koeficient poměru zastoupení. Jména jsou odvozena od tvaru grafické reprezentace (viz 3.8).

Při zkoušení zmíněných metod jsem se však setkal s problémem. Pokud byl některý ze spojitých parametrů u syntézy cílového zvuku nastaven na okrajovou hodnotu (0 nebo 1), algoritmus jen zřídka produkoval řešení, která by měla tyto okrajové hodnoty. Řešení tohoto problému nabízí metody rozšířeného přímkového a obdélníkového křížení. Rozšíření spočívá v povolení překročení hranice dané geny rodičů. U klasického křížení může gen potomka nabývat pouze hodnot v intervalu mezi hodnotami danými geny rodičů, u rozšířeného křížení je tento interval procentuálně rozšířen.

Johnson ve své práci [5] narazil na podobý problém. Algoritmus jen zřídka nastavil hodnotu genu na 0. Jelikož i malá hodnota některého z genu mohla způsobit velkou změnu ve výsledném zvuku, rozhodl se ke každému genu vytvořit další gen, tzv. regulační. Ten mohl nabývat pouze hodnoty 0 nebo 1 a byl přímo navázán na funkčnost genu, který reguloval. Hodnota 0 vyrušila hodnotu regulovaného genu a gen se tedy choval, jako by měl hodnotu 0. Když mel regulační gen hodnotu 1, regulovaný gen se choval normálně. Algoritmus velmi rychle poznal, zda má být gen "vypnutý"nebo "zapnutý", takže problém s příliš častým a zbytečným "vypnutím"genu nenastával.

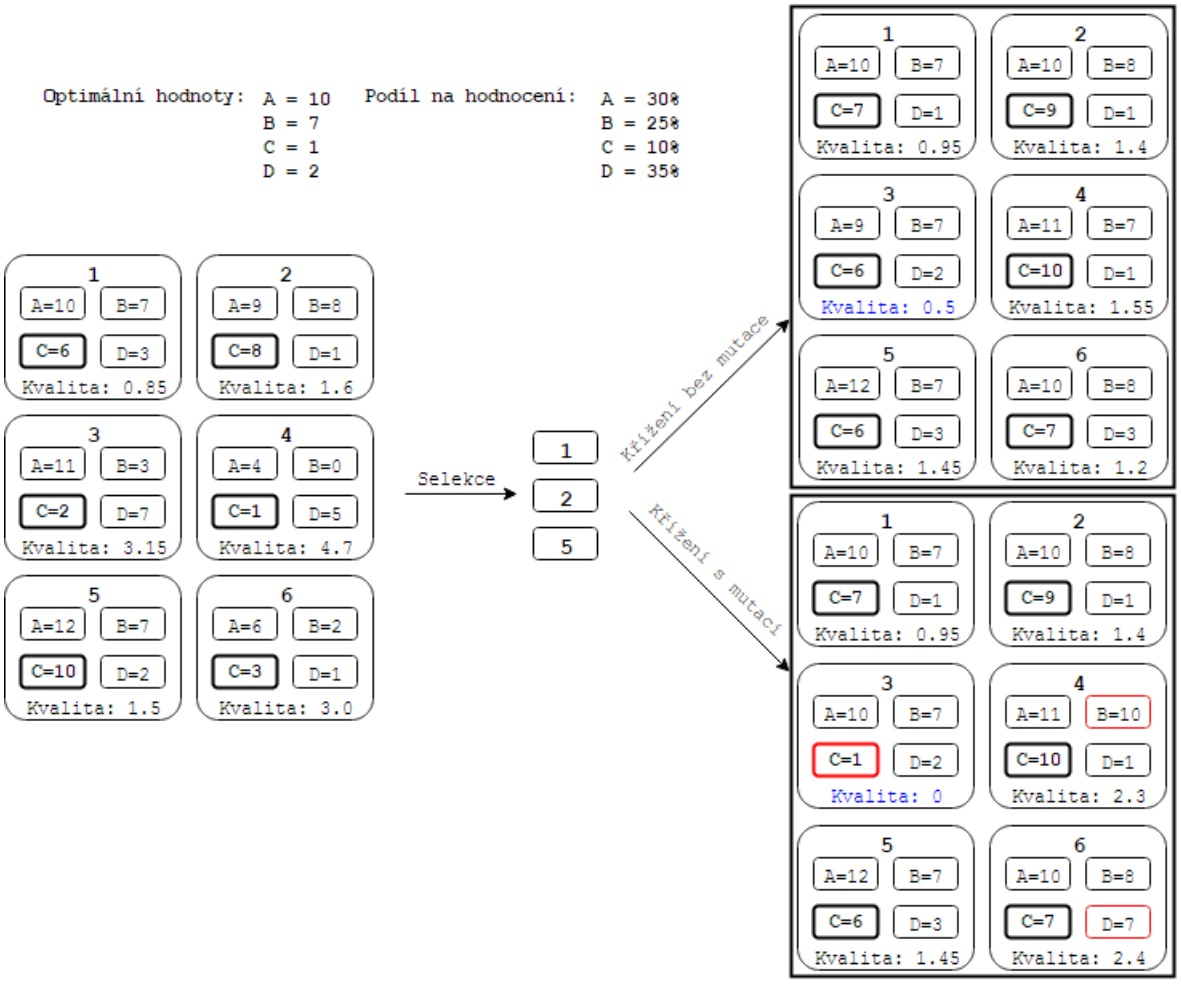



Obrázek 3.8: Grafická reprezentace přímkového a obdélníkového křížení a jejich rozšíření.

### 3.3.1 Mutace genů

Mutace je následek náhodného jevu při křížení, který má za cíl znemožnit totální eliminaci určitých hodnot genu. Mutace mění hodnotu genu (nebo genů) a přitom může nebo nemusí brát vpotaz původní hodnotu (hodnoty). Na obrázku 3.9 je vidět populace o velikosti 6. Každý jedinec obsahuje nějaké geny, reprezentované malými obdélníčky s hodnotami. Tyto hodnoty de facto určují kvalitu jedince. Zvýrazněná hodnota je hodnota minoritního genu $C$, který do kvality příliš nezasahuje. Jak je vidět, selekcí projdou jedinci, kteří mají hodnotu $C > 5$, jelikož jejich ostatní geny ovlivnily řešení natolik, že jedince jako celek, dělají kvalitnějším. Do další generace tedy nepostoupil žádný jedinec, který by měl hodnotu $C \leq 5$. Algoritmus bez mutace tedy skončí s nejlepším možným výsledkem 0.5. Algortmus s mutací náhodně zmutoval gen $C$ a mohl tak dosáhnout jakékoli hodnoty genu (v našem případě to byla zrovna hledaná hodnota).

V předchozím příkladě jsme samozřejmě uvažovali takový způsob křížení, který nedovoluje měnit hodnoty genů mimo interval daný jedinci, kteří se křížení účastní. Mutace tedy není jediným mechanismem, který může zabránit eliminaci hodnot genu. V určitých případech mutace také přispívá k rychlejšímu nalezení správných hodnot genu. Pokud dojde k mutaci, která jedince znekvalitní, jedinec bude mít menší pravděpodobnost na to, aby se jeho geny dostaly do další generace a zmutovaný gen tedy zahyne společně s jedincem. Pokud ale dojde k mutaci, která jej učiní kvalitnějším, tak tento jedinec bude mít vyšší šanci na účast při křížení a případné elitářství a tím se hodnota genu uchová pro další generace.




Obrázek 3.9: Příklad fungování algoritmu s/bez mutace (červeně jsou označeny zmutované geny a modře nejlepší možné dosažené řešení).

# Kapitola 4

# Porovnávání zvuků

Jednou z kritických oblastí této práce je porovnávání zvuků mezi sebou. Proto zde rozeberu metody použité v pracích, které se zaměřují na podobné téma.

## 4.1 Spektrální normy a Spektrální centroidy

Johnson [5], který se odvolává na práce Yonga [6] a Laie [7] používá k porovnávání zvuků tzv. *Spektrální normy* a *Spektrální centroidy*. Z nich následně vypočítává kvalitu jedinců.

*Frekvenční spektrum* (neboli *Spektogram*) je označení pro frekvenční amplitudu vzorku[15], která se běžně získává ze zvukového signálu pomocí FFT. Jde o velice efektivní způsob měření frekvenčních pásem zvuku. Aby byla zachována rychlost FFT, je nutné, aby počet hodnot ve vzorku byl mocninou dvou.

Frekvenční spektrum nám o daném zvuku, z hlediska poslechových vlastností, řekne mnohem více, než samotný signál. Signál může pro skoro totožné zvuky vypadat vizuálně úplně jinak, to ale neplatí o jejich spektrogramech. Nevýhodou FFT je ale problém spektrálního úniku, o kterém se více píše v sekci 4.3.2.1.

FFT se obvykle aplikuje na malé úseky v řádech desítek milisekund. Tím získáme několik vrstev spektrogramů v čase, které zobrazují, jak se měnilo frekvenční složení signálu. Velikost spektra však pro různě dlouhé časové zůstává stejná.

Tím se dostáváme k rozlišení spektra. Rozlišení je dáno velikostí úseku a jeho vzorkovací frekvencí ($Fs$). Z velikosti úseku získáme počet složek (tzv. bins) spektrogramu jako $B = \frac{velikost}{2}$. Samotné rozlišení je pak dáno jako $FR = \frac{F_{max}}{B}$, kde $F_{max} = \frac{Fs}{2}$. $F_{max}$ označuje maximální hodnotu frekvence, která může být zaznamenána.

Pro $Fs = 44100Hz$ tedy platí, že maximální zaznamenaná frekvence činí $22050Hz$, což je hranice, kterou je schopen běžný člověk vnímat sluchem. Z toho plyne, že nižší vzorkovací frekvence vede k omezení frekvenčního rozsahu zvuku a tedy k jeho znekvalitnění vzhledem k lidskému sluchu.

Následující vzorce jsou převzaté z práce Johnsona, který se odkazuje na [6] a [7]. Pro spektogram $f_x$ definujeme jeho velikost jako $f_x(w, b)$, kde $w$ je úsek na který jsme FFT aplikovali a $b$ je složka FFT. Označme spektogramy syntetizovaného a cílového zvuku jako $f_x$ a $f_y$. Dále označme $W$ jako počet úseků, které mají být porovnávány, a $B$ jako počet složek.

Spektrální norma je pak dána vzorcem:

$$N(a, b) = \sum_{w=1}^{W} \sum_{b=1}^{B} |f_x(w, b) - f_y(w, b)|^2 \tag{4.1}$$

Což je mimochodem pouze součet všech kvadratických rozdílů. Kvadrát zde pomáhá k umocnění velkých rozdílů v jednotlivých bodech.

Spektrální centroid se počítá pro samostatný zvuk ($a$) a úsek ($w$) a je dán vzorcem:

$$C(a, w) = \frac{\sum_{b=1}^{B} f_a(w, b) \times b}{\sum_{b=1}^{B} f_a(w, b)} \tag{4.2}$$

Centroid má vztah ke světlosti[1] zvuku, jelikož vyšší frekvence, které mají za následek světlejší zvuky, zvyšují hodnotu centroidu. Nyní je možné spočítat rozdíly centroidů dvou zvuků jako součet absolutních rozdílů všech jednotlivých hodnot centroidů:

$$C_{dif}(a, b) = \sum_{w=1}^{W} |C(a, w) - C(b, w)| \tag{4.3}$$

Výsledný rozdíl dvou zvuků (resp. kvalitu syntetizovaného zvuku) pak lze spočítat jako:

$$e(a) = KN(a, b) + (1 - K)C_{dif}(a, b) \tag{4.4}$$

Konstantu $K$ je nutno zvolit jakožto váhu pro jednotlivé složky rozdílu. Johnson [5] ve své práci zpochybňuje hodnotu $\frac{1}{2}$, kterou zvolil Lai [7], jelikož pro ní nemá žádné odůvodnění. Rozhoduje se proto vytvořit sadu testů na zjištění optimální hodnoty

[1] Jako světlost se označuje barva zvuku, která je velmi výrazná a indikuje vysoké frekvence.

rozdělení vah a dochází k hodnotě 0.8. Pokud bychom uvažovali o použití této metody, tak by vzhledem k odlišnostem prací bylo nejvhodnější vytvořit vlastní testy a z výsledků odvodit optimální hodnotu $K$.

## 4.2 Akustické otisky

Při zpracovávání rešerše jsem narazil na pojem *Akustické otisky*. Jedná se o deterministicky generovanou reprezentaci zvuku, která není přímo závislá na bitové reprezentaci zvuku. V této reprezentaci se může odrážet mnoho faktorů a vlastností zvuku. Příkladem může být zero-crossing rate, směrodatná odchylka, odhadované tempo, zvuková intenzita, frekvenční spektrum, šířka frekvenčního pásma, ale i délka či metadata souboru, ve kterém je zvuk uložen. Nejde tedy o konrétní reprezentaci, ale spíše o oabstrakní pojem. *Akustickým otiskem* lze nazvat i *Spektrální normy* a *Spektrální centroidy*.

Akustické otisky se používají v programech na rozpoznávání hudby, jako je například *Shazam* [16]. Takové programy však neporovnávají pouze 2 zvukové nahrávky, ale i několik stovek či tisíc, aby dosáhly výsledku. Porovnávané otisky jsou uloženy v komplexní databázi. Důraz je zde kladen především na jednoduché porovnávání, efektivní vyhledávání a na návrh databáze. Tyto problémy nás však míjí.

## 4.3 MFCC

MFCC neboli Mel Frequency Cepstral Coefficients je způsob reprezentace spektrálních informací zvuku [17]. Toto reprezentace se následně používá především při porovnávání zvuku. Tato metoda se používá při analýze řeči a je dominantním nástrojem v této oblasti již dlouhou dobu (např. Young, Woodland & Byrne 1993). Úspěch této metody je založen na schopnosti reprezentovat amplitudové spektrum v kompaktní formě. Zároveň zohledňuje poslechové vlastnosti zvuku [18], což je jeden z požadavků této práce. Algoritmus je oblíbený především díky své výpočetní nenáročnosti a robustnosti. [19] Cílem tohoto projektu však není práce se zvukovým materiálem obsahující nahrávku řeči, ale práce s hudbou (resp. krátkými úseky hudebních zvuků). Podle Loughran & Walkera [17], Beth Logan [18], Mubarak & Ambikairajah [20] a David Pye [21] je ale vhodné používat MFCC i pro práci s hudbou. David Pye sice tvrdí, že MFCC je vytvořeno spíše pro rozpoznávání řeči, ale zároveň podotýká, že je to dobrý startovní bod i při porovnávání hudby. Mě osobně se MFCC při porovnávání zvuků osvědčilo a proto jsem se rozhodl ho pro tuto práci použít.

### 4.3.1 Reprezentace MFCC

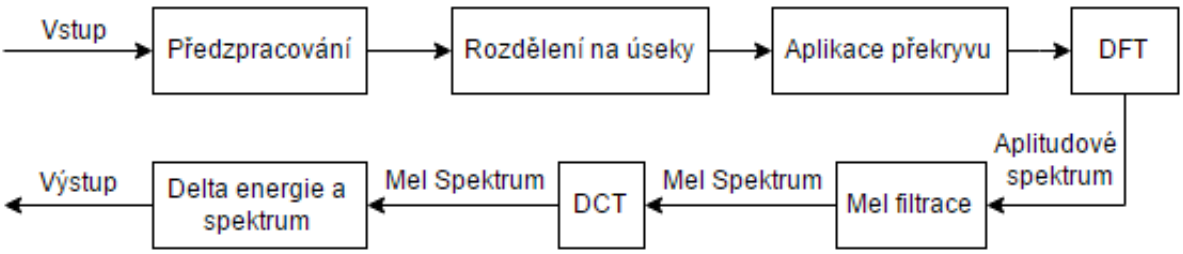
Jak již bylo řečeno, MFCC reprezentuje signál v kompaktní formě. Touto kompaktní formou je myšlen vektor o několika ($N$) neceločíselných hodnotách (tzv. feature = vlastnosti). N je volitelné a má významný vliv na rychlost, přesnost a robustnost algoritmu. Obecně lze říci, že čím je N menší, tím je algoritmus rychlejší a ztrácí na své přesnosti reprezentace. Poměr rychlost:přesnost je však nelineární a navíc se zde vyskytují i výjimky. K výslednému vektoru se obvykle přidává ještě jedna hodnota, která udává logaritmus energie úseku, případně další hodnoty, jako jsou výška, zero crossing rate, atd. [22]

### 4.3.2 Proces MFCC

Algoritmus pro výpočet MFCC se skládá z 5 kroků [**?**] [19] [**?**]:

1. Rozdělení signálů do úseků (frames)
2. Získání amplitudového spektra pomocí DFT
3. Namapování mocnin spektra na mel škálu a mel filtrace
4. Zlogaritmování hodnot získaných z kroku 3
5. DCT (pomocí IFT) umocněných hodnot získaných z kroku 4

Pro výpočet DFT předpokládám využití FFT.




Obrázek 4.1: Blokový diagram fází MFCC

#### 4.3.2.1 Rozdělení signálů do krátkých úseků a jejich překryv

Zvukový signál obvykle nebývá stacionární a proto je nutné rozdělit jej do několika stejně dlouhých vzorků, u kterých lze, alespoň po statistické stránce, předpokládat

stacionaritu. Tomuto procesu se říká okénkování (windowing) Je zřejmé, že stacionarita vzorků nebude úplná, ale vzorky budou mnohem více stacionární, než původní nedělený zvuk. S takovými krátkými vzorky už dokáže MFCC pracovat korektně. Otázkou však je, jak tyto vzorky rozdělit, s jakým překryvem a jak velké by měly být.

#### 4.3.2.2 Diskrétní Fourierova transformace v MFCC

Při provádění DFT se předpokládá, že zpracovávaný úsek je periodický a spojitý. Pokud jedna z podmínek neplatí, což se běžně, vzhledem k velikosti úseků, stává, může se tento problém projevit nechtěnými nárůsty hodnot v prvním a posledním bodě výstupu. Částečně se tomutu efektu zabraňuje pomocí překrývání úseků.

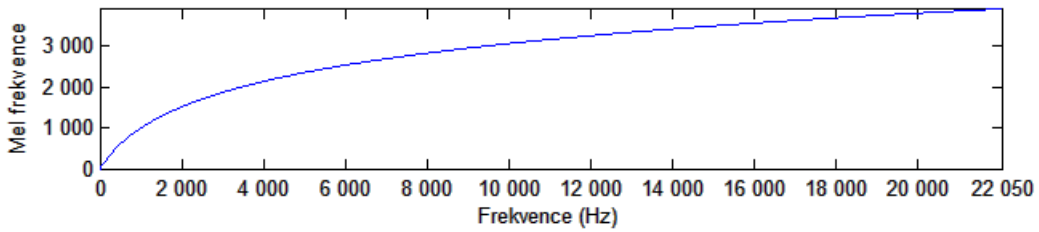
#### 4.3.2.3 Mel škála a mel filtrace

Mel škála slouží pro převod frekvence do melů. Označení mel je odvozeno od slova "melody". Jelikož člověk vnímá nízké a vysoké frekvence s rozdílným rozlišením, nehodí se frekvence pro reprezentaci výšky zvuku, pokud chceme porovnávat poslechové vlastnosti. Výška zvuku o nízké frekvenci je pro sluchové ústrojí lépe rozlišitelná, než je tomu u zvuků s vysokou frekvencí. V našem kontextu to znamená, že pokud budeme porovnávat dva zvuky mezi sebou, větší váhu budou mít při porovnávání nižší frekvence.

Pro přepočet frekvence ($f$) na mely ($m$) se běžně používá následující vzorec [22]:

$$mel(f) = 1125 \cdot ln(1 + \frac{f}{700}), \tag{4.5}$$

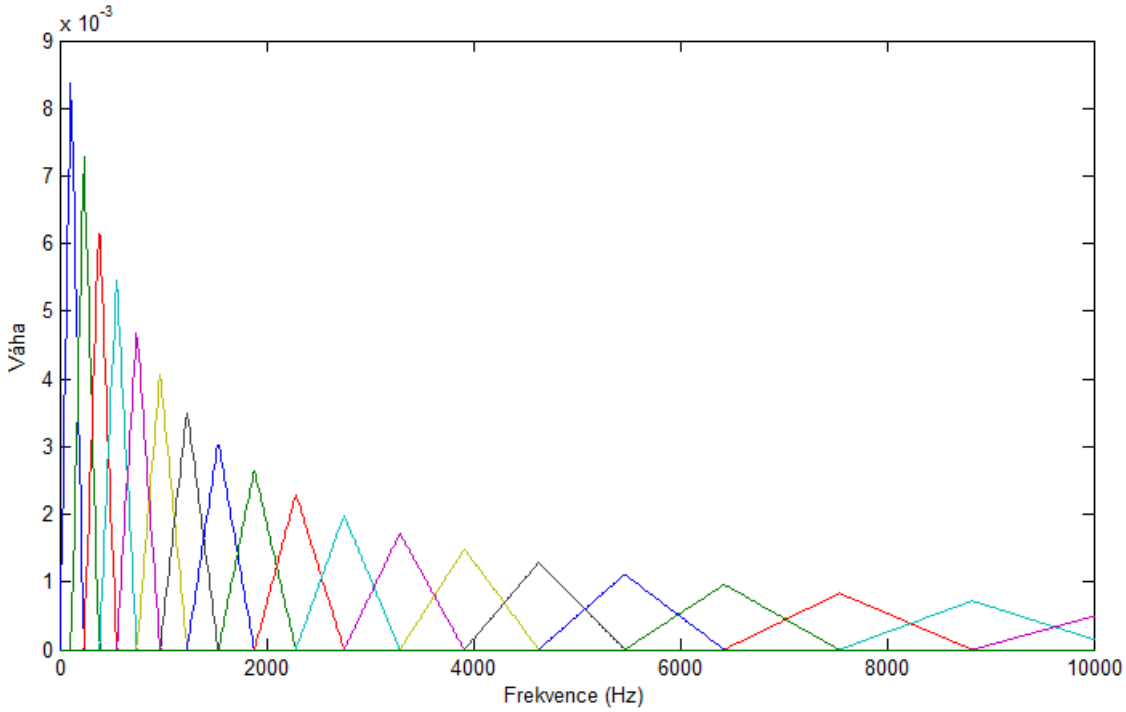
vzorce se však mohou lišit v konstantách, či v základu logaritmu.




Obrázek 4.2: Převod frekvence na mel frekvenci (resp. mel škála)

Po převodu se provádí filtrace. Tato filtrace je rozsahová a třídí jednotlivé hodnoty do skupin. Skupiny jsou rozděleny podle mel škály a obvykle jich je 23, ale je možné použít filtrů méně nebo i více. Proto se pokusím v sekci 8.2 otestovat, jaký

počet filtrů je pro naší úlohu nejvhodnější. Filtry jsou trojúhelníkové a vzájemně se překrývají. Překryv umožní zjemnit přechody mezi filtry podobně jak to činí překryv úseků při okénkování. Trojúhelníkový tvar pak zaručuje rovnoměrné rozprostření vah mezi jednotlivými filtry. Fitlrů je více druhů a liší se v překryvu filtrů a v jejich vahách.




Obrázek 4.3: Trojúhelníkové filtry mel filtrace podle Auditory Toolbox (Matlab)

Počet filtrů by měl být volen také vzhledem k frekvenčnímu rozsahu, který mají filtry obsáhnout. Podle Sigurdssona [35] dochází při volbě velkého frekvenčního rozsahu k újmě na robustnosti MFCC (podobně to samozřejmě platí i naopak). Sám jsem se o tomto tvrzení přesvědčil v sekci 8.2. Aby bylo výsledné řešení robustní, měla by spodní hranice začínat kolem 100-200 Hz a horní hranice by se měla pohybovat kolem 4.6 - 8 kHz.

#### 4.3.2.4 Diskrétní kosinová transformace

Úkolem DCT je převést zlogaritmované hodnoty energií z mel filtrace do časové oblasti. Výsledek se již označuje jako Mel Frequency Cepstrum Coefficient. Definujme $N$ jako počet filtrů a $C$ jako počet výsledných koeficientů, pak platí [30]:

$$c_i = \sum_{j=1}^{N} E_j \cdot cos(\frac{\pi \cdot i \cdot j - \frac{1}{2}}{N}), i = 1, \ldots, C \tag{4.6}$$

## 4.4 Porovnávání zvuků sluchovým ústrojím

Ač je tato metoda velmi subjektivní, její výsledky budou vzhledem k cíli pravděpodobně nejlepší. Jelikož jde o nalezení sluchové shody, je sluchové ústrojí tím nejlepším, co může zvuk porovnat. Z tohoto důvodu jsem ve vypracovaném programu umožnil uživateli ovlivňovat chod algoritmu pomocí upřednostnění vybraných syntetizovaných zvuků pomocí úpravy jejich kvality. Ve výsledku jde přeci o to, aby byl výsledek uspokojivý pro člověka, nikoliv pro stroj, který ho bude numericky vyhodnocovat.

# Kapitola 5

# Návrh a realizace genetického algoritmu

V této kapitole si vysvětlíme aplikaci genetického algoritmu na náš konkrétní problém. Dále také zvolíme konkrétní metody a parametry tohoto algoritmu vzhledem k rozboru v kapitole 3.

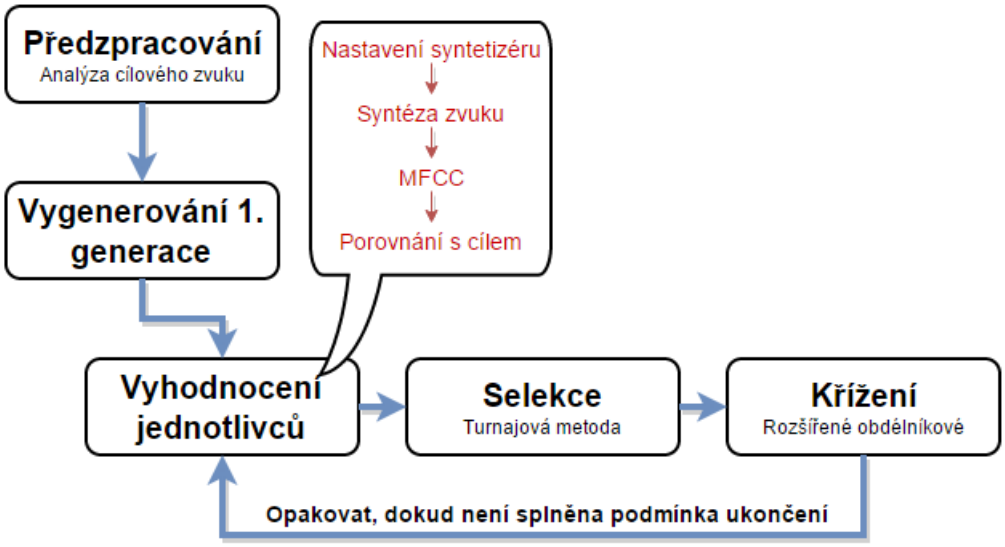
## 5.1 Genetické algoritmy v kontextu resyntézy zvuku

Převedeme-li algoritmus do kontextu našeho problému, budeme za geny považovat jednotlivé parametry konfigurace syntetizéru. Jedinec pak v sobě nese genetickou informaci (genom), která koresponduje s hodnotami parametrů syntetizéru. Kvalitu jedince hodnotí vyhodnocovací algoritmus, který vyhodnotí zvuk vytvořený syntetizérem, kterému se nastaví parametry dle genu daného jedince.

Jednotlivé parametry syntetizéru lze reprezentovat jako hodnoty od 0 do 1 a stejně lze reprezentovat jednotlivé geny jedince. Každý gen je tedy ekvivalentem jednoho parametru syntetizéru. Ne všechny hodnoty parametrů jsou spojité. Syntetizér může obsahovat parametry diskrétní, tedy takové, které mají omezený počet předem definovaných hodnot. Tento problém je nutno řešit omezením hodnot genů, které budou odpovídat hodnotám parametru. Toto omezení lze reprezentovat pomocí typů genů. Typy pokryjeme tři možnosti rozdělení hodnot:

- Boolean - pouze hodnota 0/1
- Choice - n diskrétních hodnot s rovnoměrným rozdělením mezi hodnotami 0 a 1
- Continuous - hodnota od 0 do 1

Pro představu, typem *Boolean* může být přepínač pro zapnutí a vypnutí zesilovače, typ *Choice* může sloužit k výběru typu filtru a typ *Continuous* jako ovladač hlasitosti výstupu.




Obrázek 5.1: Diagram genetického algoritmu v kontextu resyntézy zvuku

## 5.2 Selekce - turnajová metoda

V sekci 3.2 jsme rozebraly základní metody selekce. Jako nejvhodnější se zdá metoda *Turnajů*. Proč tomu tak je? Metoda automaticky nediskvalifikuje méně kvalitní jedince, je jednoduše implementovatelná, efektivní, rychlá, schopná paralelního běhu a vysoce škálovatelná. Dále nevyžaduje žádné škálování kvality ani řazení, jako tomu je u *Vážené rulety* nebo *Stochastického univerzálního výběru*. Existují však i selekční metody, které umožňují obecně o něco rychlejší konvergenci ke kvalitnímu řešení, tyto metody jsou ale výpočetně a implementačně náročnější. [25]

Z podstaty genetického algoritmu je zřejmé, že velká míra selekční intenzity vede k příliš velké preferenci kvalitních jedinců a tedy k příliš vysoké míře chamtivosti, což většinou u algoritmů tohoto typu není žádoucí. Zároveň bychom se měli snažit o co nejmenší míru ztráty diverzity a co největší míru variance.

Z grafu zjistíme, že střední intenzity, relativně nízké ztráty diverzity a varianci získáme, když zvolíme velikost skupiny 4-8. Johnson ve své práci [5] zjišťuje, jaká velikost skupiny je pro účely optimalizace parametrů syntetizéru nejvhodnější a dochází k hodnotám 7-8 včetně 10 % elitismu (viz dále). Vzhledem k tomu, že hodnoty byly posuzovány podle rychlosti konvergence k maximální povolené hranici chyby (odlišnost cílového a syntetizovaného zvuku) je zřejmé, že selekční intenzita je důležitější než

diverzita a variance. Diverzita a variance jsou však důležitými prvky v případě, že operujeme s větším počtem genů nebo pokud se příliš nevydaří generování první generace populace. Volbu velikosti skupiny tedy nechám na uživateli s předpokladem, že implicitní hodnota bude nastavena na 7. S touto hodnotou budu také dále pracovat při svých experimentech (viz. 8.5).

Nakonec je třeba rozhodnout o velikosti populace. Zde platí pravidlo, že čím větší populace, tím méně generací je třeba k tomu, aby bylo dosaženo kvalitního řešení. Nevýhodou je větší výpočetní náročnost, která pro vybraný selekční algoritmus roste lineárně. Totéž však neplatí pro změnšování počtu generací. Navíc s velikostí populace je potřeba měnit i ostatní parametry (např. množství elitních jedinců a šance na mutaci), aby se algoritmus nestal příliš chamtivým. Vzhledem k doporučené velikosti turnajů a elitismu je třeba volit velikost populace relativně vysokou, aby nedošlo k příliš velké ztrátě variance mezi jedinci. Velikost populace by se také měla volit vzhledem k počtu genů.

V paperu[36] je možné se dozvědět, že obecně je vhodné používat 100 jedinců pro každou generaci. Větší počet vede ke zbytečnému prodloužení běhu algoritmu, zatímco nižší počet může vést ke zhoršení výsledků. Johnson ve svém projektu také pracuje s hodnotou 100, proto nebylo třeba se příliš rozhodovat. Tato hodnota se navíc při testování osvědčila.

## 5.3 Výběr metody pro křížení

Vzhledem k okolnostem, které byly zmíněny v sekci 3.3 je třeba vybrat takovou metodu křížení, která dovolí pohyb hodnot genů i mimo hranice rodičů nebo tento problém ošetřit jinak. Při testování se mi však osvědčila metoda *rozšířeného obdélníkového křížení*, jelikož pomáhá řešit problém s okrajovými hodnotami bez nutnosti přidání regulátorů. Zároveň tato metoda nabízí o něco komplexnější řešení, než *přímkové křížení*, a generuje tedy rozmanitější jedince.

U *rozšířeného obdélníkového křížení* je potřeba zvolit koeficient $\alpha$, který určí maximální odchylku hodnoty po křížení od krajních hodnot genů rodičů. Pokud zvolíme příliš velkou $\alpha$, bude porušena podstata křížení, nízká hodnota povede k znehodnocení vedlejšího efektu tohoto křížení. Obecně se doporučuje zvolit $\alpha = 0.2$. Jelikož je zřejmý efekt volby příliš vysoké či nízké hodnoty, není třeba tuto hodnotu vystavovat testování. Hodnotu jsem při obecném testování zkoušel měnit a s původní hodnotou jsem byl nakonec spokojen.

## 5.4 Mutace genů

Mutace může a nemusí brát v úvahu původní hodnotu genu. Pokud se hodnota nezohlední, jde o naprosto náhodnou mutaci a gen tak může získat jakoukoli hodnotu z intervalu hodnot tohoto genu. Mutace také může změnit hodnotu několika genů a to například přehozením hodnot, ale tento způsob pro náš případ není vhodný. Pokud se hodnota zohledňuje, může se tak stát hned několika způsoby. Tyto způsoby se liší v pravděpodobnostním rozdělení.

Příkladem může být Gaussovo rozdělení, Cauchyho rozdělení nebo třeba Adaptivní Levyho rozdělení [27]. Jelikož se algoritmus spoléhá spíše na křížení a mutace je pouze pomůckou, rozhodl jsem se použít základní verzi mutace, která mění hodnotu genu náhodně. Taková forma mutace je navíc nejběžnější a ve své práci ji používá i Johnson [5] a to se šancí 5%, což je běžně využívaná hodnota. Vyšší šance by mohla způsobovat nechtěný zánik kvalitních jedinců, nižší šance by znehodnotila efekt mutace.

## 5.5 Geny, parametry a jejich kontext při syntéze zvuku

Jak jsem již podotkl, rozhodl jsem se pokusit genetický algoritmus podpořit tak, že u některých genů bude znám kontext v jakém gen působí. Lze si to představit na jednoduchém příkladu: Řekněme, že člověk má gen, který určuje jeho výšku. Algoritmus, který zajišťuje evoluci a křížení o tomto genu ví. Cílem algoritmu je vytvořit člověka, který bude vysoký 2 metry. Algoritmus nyní dostane 2 jedince o výšce 1.4 ($A$) a 2.1 ($B$) metru. Tyto jedince má zkřížit. Každá metoda křížení je náhodná a tak může vzniknout jedinec, který je různě vysoký. Pokud ale algoritmus bude vědět o genu, který ovlivňuje výšku, pak také dokáže říci, že hodnota genu jedince $B$ je vhodnější, než hodnota jedince $A$. Algoritmus pak může brát gen jedince $B$ s větší vahou a při křížení se přikloní spíše k hodnotě genu tohoto jedince.

Nápad se to jeví jako zajímavý, ale v praxi jen velmi obtížně proveditelný. Zaměřil jsem se na rozpoznání hlasitostní ADSR obálky, filtraci frekvence a posun oktávy. Ostatní parametry, jako tvar vlny oscilátoru, atp. jsou příliš komplexní (resp. mají komplexní vliv na signál), než aby je bylo možno jednoduše rozpoznat.

I Kdyby bylo možné rozpoznat kontext všech parametrů, pak jde sotva jen o polovinu práce. Aby bylo možné získané informace nějak použít, musel by se syntetizovaný zvuk porovnávat s cílovým na takové úrovni, aby byly zjištěny jeho nedostatky v rámci kontextů. Toho je možné dosáhnout v případě, že rozdíl mezi syntetizovaným a cílovým zvukem je v jednom kontextu (např. liší se pouze v ADSR obálce). Jakmile je ale rozdíl ve více kontextech (např. liší se ADSR obálka a spodní frekvenční filtr), můžeme rozdíl

pouze odhadovat.

Pro zajímavost zde alespoň uvedu, jak bylo postupováno při snaze o získání kontextu parametru. V prvé řadě se vygeneruje nějaké náhodné nastavení syntetizéru. Následně se určí jeden parametr, se kterým se bude manipulovat a pro který se bude zjišťovat kontext. Tento parametr se pro jeden zvuk nastaví náhodně na velikost 0 - 0.4 a pro druhý 0.6 - 1. Pak se vzhledem k tomuto nastavení syntetizují dva zvuky. Tyto zvuky se porovnají a program se pokusí zjistit kontext. Pokud se mu to podaří, práce je u konce a je možno určit kontext dalšího parametru. V opačném případě se vytvoří nový pár zvuků a práce začíná od začátku. Pokud se nepodaří zjistit kontext po syntéze pěti párů zvuků, program zjišťování vzdá a označí kontext jako nezjistitelný typ.

Zda parametr ovládá hlasitostní ADSR obálku se zjišťuje pomocí obálky hlasitosti zvuku. Obálka se získá z Hilbertovy transformace velikostí amplitud signálu. Rozdíl mezi obálkami základního a porovnávaného zvuku se provádí pomocí procentuálního rozdílu mezi obálkami. Procentuální rozdíl se rozokénkuje a následně se zjistí variance a průměr pro každé okno. Pokud je průměr rozdílů a variance dostatečně velká, pak se s největší pravdepodobností jedná o rozdíl při syntéze v použité hlasitostní obálce. Pokud je průměr dostateně velký a variance malá, pak jde pravděpodobně o rozdíl v celkové hlasitosti.

Filtraci frekvencí lze zjistit pomocí porovnání nejmenší a největší zaznamenané frekvence (s tím, že je použit nějaký rozumný práh amplitudy, který je nutné překročit). Pokud se dostatečně liší, je pravděpodobné, že parametr ovlává danou frekvenční filtraci.

Změna oktávy se hodnotí dle posunu nejsilnější frekvence v celém signálu. Pokud je posun přibližně o celou oktávu dolu nebo nahoru, pak lze říci, že parametr ovládá posun oktávy.

To je však teorie. V praxi by bylo třeba provést mnohem důkladnější analýzu signálu, jelikož dochází k rozdílné syntéze zvuku za stejných podmínek (viz 6.4). Z časových důvodů jsem se rozhodl tomuto problému více nevěnovat.

# Kapitola 6

# Porovnávání syntetizovaného a cílového zvuku pomocí MFCC

Pro genetický algoritmus je porovnávání syntetizovaného a cílového zvuku kritickým bodem, ač přímo nesouvisí s vlastním algoritmem. Je prakticky nutné, aby tento proces byl rychlý, ale zároveň přesný a opakovatelný (resp. deterministický). Dále je vhodné, aby byly zvuky porovnávány z poslechového hlediska, nikoliv pouze z hlediska vlastností signálu či dokonce pouze podle reprezentace tohoto signálu. Jak již bylo zmíněno, právě z těchto důvodů jsem vybral pro porovnávání zvuků reprezentaci MFCC.

Pro výpočet MFCC bude použita Java knihovna *OpenIMAJ*. Jelikož MFCC využívá DFT využijeme zde FFT, jejíž implementace se nachází v Java knihovně *jTransforms*.

## 6.1 Volba počtu koeficientů v MFCC

V sekci [**?**] jsem popsal, jak vypadá reprezentace MFCC. Jde tedy o vektor čísel o velikosti $N$. Při zpracování řeči obvykle platí $N = 13$. U nižších hodnot dochází k výraznému poklesu přesnosti porovnávání. U vyšších hodnot dochází většinou ke stagnaci nebo zhoršování přesnosti [**?**] [26].

Jelikož je porovnávání zvuků v této práci kritickým bodem, je nutné zvolit hodnotu $N$ korektně, vzhledem k tomu, že významně ovlivňuje přesnost a rychlost této metody. Pokud bychom se měli řídit doporučeními, která platí pro porovnávání řeči, zvolíme $N = 13$. Je však nutné brát vpotaz, že řeč a hudba se liší. Hudba má obvykle větší dynamický rozsah než řeč a navíc se v této práci zaměřujeme především na krátké souvislé úseky hudebních zvuků. Podle Mubarak & Ambikairajah [**?**] může u hudby vyšší hodnota $N$ zaručit vyšší přesnost. Ve zmíněné práci se však zaměřují na dlouhé hudební nahrávky, které svou nestacionaritou spíše odpovídají řeči. Loughran & Walker

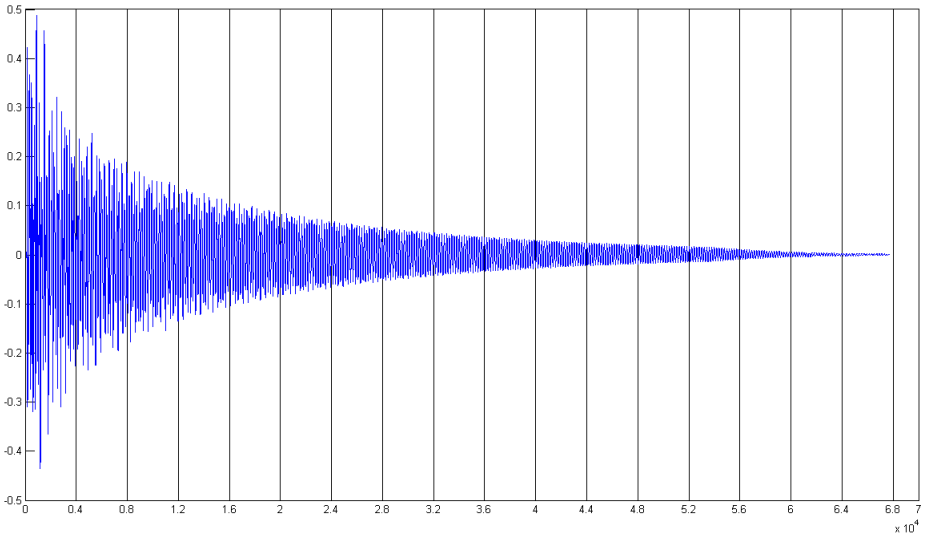
[?] naopak dokazují, že pro krátké zvuky produkované pomocí hudebních instrumentů, je vhodné se držet osvědčené hodnoty 13.

Rozhodl jsem se tedy držet doporučené hodnoty 13, ale pro zajímavost jsem v sekci 8.2.2 provedl jednoduché experimenty, které by měly objasnit případné pochybnosti.

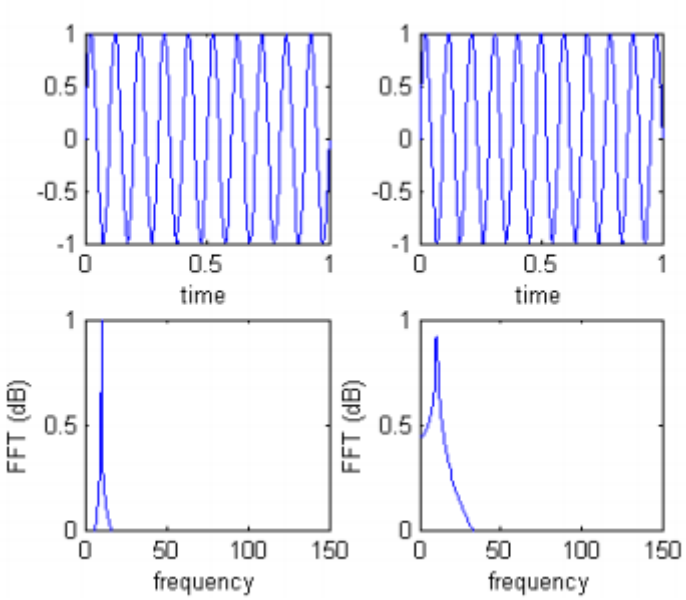
## 6.2 Rozdělení signálů do krátkých úseků a jejich překryv

Již máme vybraný algoritmus pro porovnávání zvuků, nyní zbývá zvolit konkrétní zpracování. První problém, který je třeba vyřešit je volba délky vzorků pro okénkovací funkci. Aby byla zachována efektivita FFT, je nutné volit délku vzorku tak, že počet hodnot v tomto vzorku by měl být roven některé mocnině dvou. [22] Obvykle se délka vzorku volí mezi 10-30ms [32] [22], což odpovídá 441-1323 hodnotám při vzorkovací frekvenci 44.1 kHz. Mocninám dvou v tomto rozmezí odpovídá hodnota 512 (11.61ms) a 1024 (23.22ms). Rozhodnout mezi těmito dvěma hodnotami nám pomůže [23]. Z této práce vyplývá, že nejvhodnější je použití vzorků o délce 20ms, čemuž je nejbližší našich 1024 vzorků.

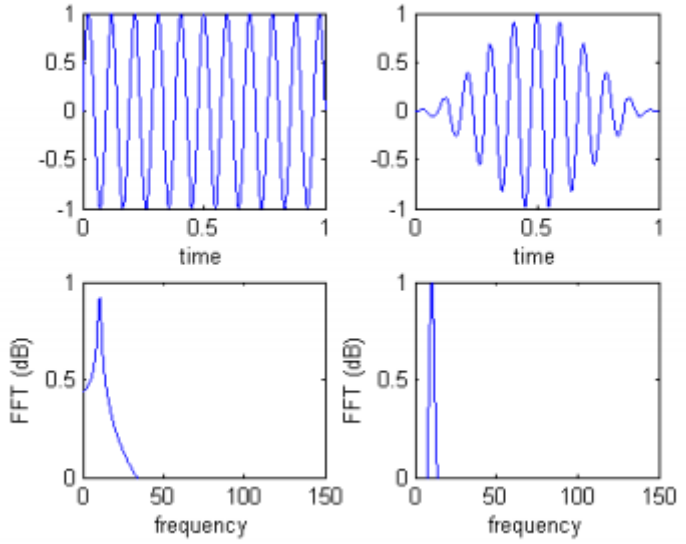
Nyní zbývá zvolit okénkovací funkci. Nejjednodušším způsobem, jak rozdělit zvuk na vzorky, je obdélníkově bez překryvu (viz 6.1). Takové rozdělení je ale vhodné použít pouze v případě, že je zpracovávaný signál periodický, což v našem případě neplatí. Pokud je signál neperiodický, může při takovémto rozdělení docházet ke spektrálnímu úniku (viz 6.2 a 6.3). [31]




Obrázek 6.1: Obdélníkové okénkování




Obrázek 6.2: Porovnávní spektrálního úniku periodického (vlevo) a neperiodického (vpravo) signálu před/po aplikaci FFT. [31]




Obrázek 6.3: Porovnávní spektrálního úniku neperiodických signálů při použití nevhodného (vlevo) a vhodného (vpravo) rozdělení úseků před/po aplikaci FFT. [31]

| **Název** | **Vhodné pro signály** | **Frekvenční rozlišení** | **Spektrální únik** | **Amplitudová přesnost** |
|---|---|---|---|---|
| Barlett | Náhodné | Dobré | Ucházející | Ucházející |
| Blackman | Náhodné / složené | Špatné | Nejlepší | Dobré |
| Flat top | Sinusové | Špatné | Dobré | Ucházející |
| Hanning | Náhodné | Dobré | Dobré | Ucházející |
| Hamming | Náhodné | Dobré | Ucházející | Ucházející |
| Kaiser-Bessel | Náhodné | Ucházející | Dobré | Dobré |
| Žádné (obdélníkové) | Přechodné a synchronní | Nejlepší | Špatné | Špatné |
| Tukey | Náhodné | Dobré | Špatné | Špatné |
| Welch | Náhodné | Dobré | Dobré | Ucházející |

Tabulka 6.1: Okénkovací funkce a jejich vlastnosti [31]

Omezit spektrální únik lze pomocí použití vhodné okénkovací funkce. Nejběžnější okénkovací funkce a jejich vlastnosti jsou uvedeny v tabulce 6.1.

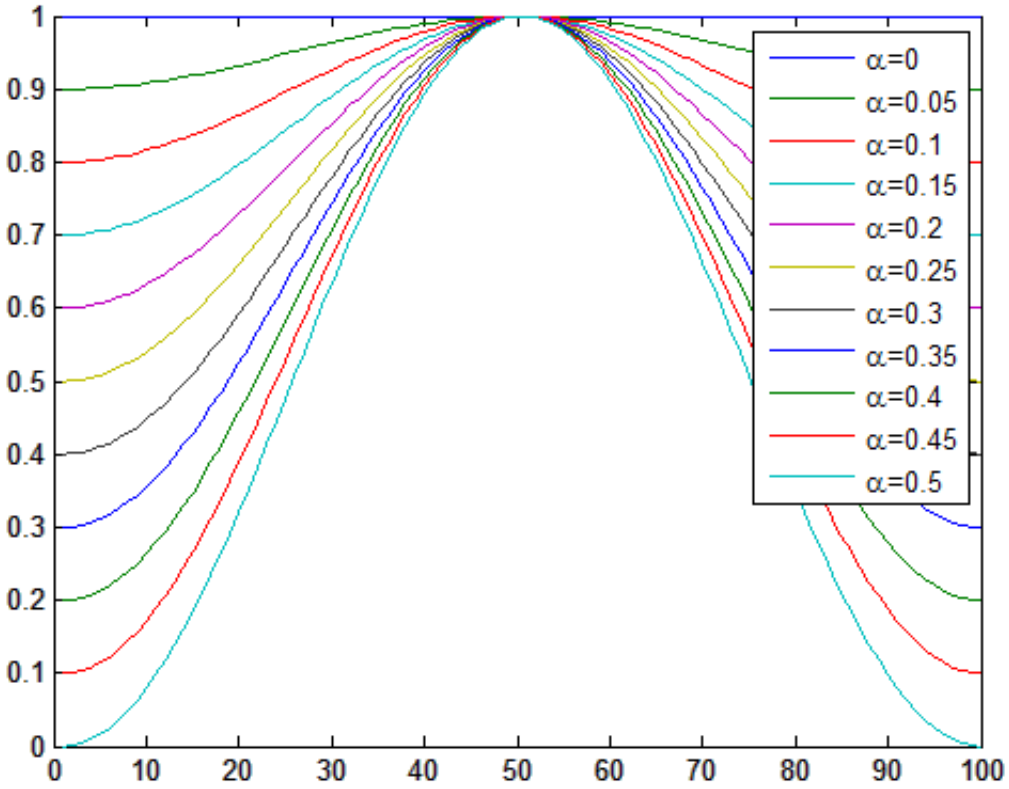
Pro naše využití se jeví jako nejlepší Hanningova okénkovací funkce, případně Barlettova, Welchova či Hammingova. Všechny splňují požadavek na náhodný signál, mají dobré frekvenční rozlišení a omezují spektrální úniky, zároveň jsou ucházející co se amplitudové přesnosti týče. Jelikož jsem nebyl schopen nalézt jedinou práci, kde by pro MFCC využívali jinou okénkovací funkci než-li Hammingovu, je volba této funkce pravděpodobně nejvhodnější.

Výpočet hodnot pro Hammingovu funkci je dán následujícím vzorcem [22]:

$$w(n,\alpha) = s(n) \cdot ((1-\alpha) - \alpha \cdot cos(\frac{2\pi}{N-1} \cdot n)), 0 \leq n \leq N-1, \tag{6.1}$$

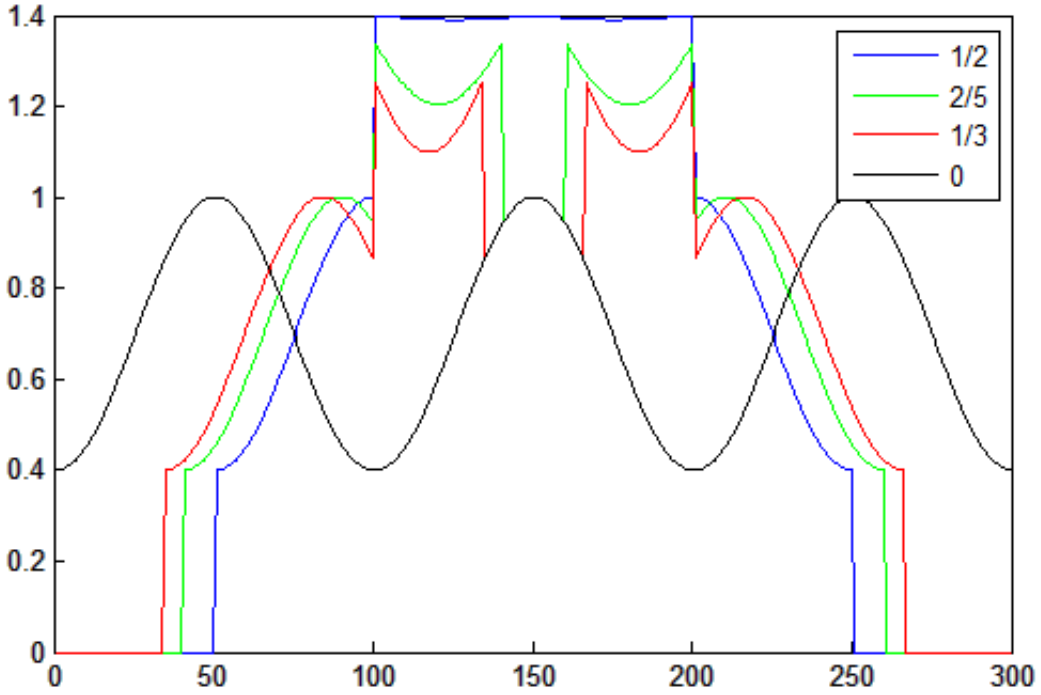
kde N je počet hodnot ve vzorku a s(n) je hodnota ze vzorku na pozici n. Hodnota $\alpha$ ve vzorci určuje tvar Hammingovy funkce. Jak je vidět na obrázku 6.4, okrajové hodnoty jsou násobeny menší hodnotou, než hodnoty ve středu vzorku. Právě tato vlastnost je důležitá pro eliminaci spektrálního úniku. Pokud by platilo $\alpha = 0$, pak bychom dostali již zmíněné obdélníkové rozdělení. Příliš nízká $\alpha$ tedy vede k omezení eliminace spektrálního úniku. Příliš vysoká *alpha* ($\alpha > 0.5$) zase vede k zanedbání či dokonce vyřazení okrajových hodnot vzorku. Z toho plyne, že je nutno volit $\alpha$ mezi 0.25 a 0.5, aby byla zachovány vlastnosti této funkce. Tomu odpovídá i doporučení Janga [22], který tvrdí, že v praxi se běžně používá hodnota $\alpha = 0.46$.

S volbou neobdélníkové okénkovací funkce přichází nutnost překryvu vzorků, pro které se bude MFCC počítat. Kdyby nedošlo k překryvu, došlo by k zanedbání okrajových hodnot vzorku. Překryv je nutno volit tak, aby korespondoval s tvarem křivky




Obrázek 6.4: Hammingova funkce s různými koeficienty

zvolené okénkovací funkce. Jang ve své knize [22] píše, že se pro tyto účely používá hodnota překryvu $\frac{1}{2}$ - $\frac{1}{3}$ velikosti vzorku. Pro toto tvrzení však neuvádí žádné podklady. Můžeme se snažit překryvem pokrýt všechny hodnoty signálu stejnou měrou, v tom případě bychom pro Hammingovu funkci měli zvolit překryv o hodnotě $\frac{1}{2}$ (viz 6.5). Jelikož však nic kromě doporučení a intuice nenasvědčuje tomu, jaká hodnota by se měla použít, provedl jsem testy (viz 8.3), které by měly tento problém osvětlit.




Obrázek 6.5: Různé překryvy Hammingovy funkce

## 6.3 Porovnávání MFCC dvou zvuků

MFCC z knihovny *OpenIMAJ* vrací pro každé zvukové okno 13 koeficientů, přičemž první z nich je suma všech zlogaritmovaných energií (viz 4.3.2.4). Existuje mnoho způsobů, jak porovnávat vektory. Mezi nejpoužívanější a nejznámější patří například *Euklidovská vzdálenost*, *Manhattanská vzdálenost*, *Test dobré shody* (tzv. *Chi kvadrát*), *Sorensenova vzdálenost* a *Korelační vzdálenost*. Při pátrání, jak nejlépe porovnávat MFCC jsem nejčastěji narážel práce jež využívají ***Euklidovskou vzdálenost*** (viz [32], [33] a [29]).

Na obrázku 6.6 je možné vidět, že první koeficient je znatelně hodnotově odlišný od ostatních. Jeho výpovědní hodnota je přitom nižší, než je tomu u ostatních koeficientů. Jelikož jde o sumu energií, tato hodnota nese pouze informaci o hlasitosti. Pokud bychom tedy chtěli využít *Euklidovskou vzdálenost*, tento koeficient se na ní bude

podílet většinově, což není žádané. Je však obecně známé, že se tato hodnota při porovnávání zanedbává (pokud není chtěné porovnávat především hlasitost). S využitím zmíněné vzdálenostní metody by tedy neměl být problém. Přesto jsem však možnost zahrnutí prvního koeficientu do porovnávání nechal na uživateli a do programu připojil přepínač, který toto zajišťuje.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| -223.52 | 20.8927 | -1.7138 | 7.0949 | 2.1351 | 2.9897 | 3.5673 |

| 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|
| 2.1862 | 3.5450 | 1.8767 | 2.5123 | 1.9926 | 1.9796 |

Obrázek 6.6: Typický příklad hodnot MFCC

Jelikož je MFCC vlastně histogram, mohli bychom jej porovnávat také pomocí *Earth Mover's Distance* (EMD). To také doporučuje práce Beth Logan [34] o které jsem se již zmínil. Tento algoritmus hledá nejmenší náročnost přechodu mezi dvěma stavy. Výhodou tohoto algoritmu je komplexnější pohled na celý problém, nevýhodou je časová náročnost výpočtů ($O(N^2)$). Algoritmus jsem nakonec nevyužil, viz 6.4.1.

## 6.4 Rozdílná syntéza zvuku, totožné parametry syntetizéru

V sekci 2.3.1 jsem rozebrali z jakých částí se může syntetizér skládat. Některé tyto součásti mohou způsobit, že při syntéze je pokaždé výstupem trochu odlišný zvuk. Například nízkofrekvenční oscilátor může začínat na hodnotě 0, po syntéze prvního zvuku se může nacházet na hodnotě 1, odkud pokračuje syntéza dalšího zvuku a ta tedy začíná v 1. Je tedy zřejmé, že v takovém případě dojde k syntéze odlišného zvuku.

Další součásti v sobě mohou mít čítače či nějaký náhodný efekt, který způsobí, že zvuk bude pokaždé znít trochu jinak. Míra odlišnosti je závislá na efektu, některé mohou zvuk pouze lehce pozměnit tak, že rozdíl je neslyšitelný, jiné mohou změnit například hlasitost či výšku v průběhu, což je při porovnávání velký problém.

Tento typ nedeterminismu způsobuje, že zvuk syntetizovaný pomocí stejného nastavení parametrů, bude algoritmem ohodnocen pokaždé jinak. To může přispět k nepřesnosti celého programu.

Problém by se dal vyřešit uzavřením a opětovným otevřením syntetizéru, ale ani to není zárukou, že vygenerované zvuky budou totožné (např. zmíněné náhodné efekty a

procesy). Problém lze částečně vyřešit také mnohonásobnou syntézou totožného zvuku, kde výsledkem bude jeden zvuk, který je nejvíce podobný zvuku cílovému. Nevýhodou je lineární růst časové náročnosti. Tento princip se nakonec při testování osvědčil a je v programu implementován.

S touto nesnází souvisí další problém. Při syntéze zvuku dochází v programu k ustřižení zvuku po určité době. Děje se tak z toho důvodu, že některé nastavení syntetizérů může mít za následek syntézu příliš dlouhého zvuku, který zřejmě není shodný s cílovým. Po tomto ustřížení zůstane v syntetizéru zvukové reziduum, které je nutno odstranit, jinak by bylo připojeno k dalšímu syntetizovanému zvuku. Odstranění probíhá tak, že dojde ke zpracování dlouhého úseku přes syntetizér. Tento úsek je následně zahozen a syntetizér by měl být "čistý".

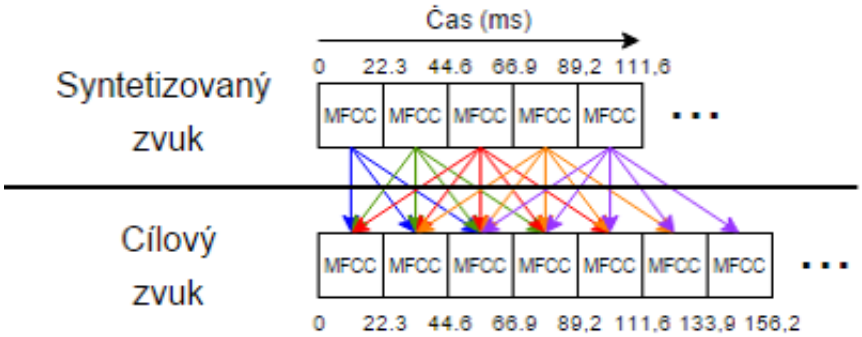
### 6.4.1 Posun zvuku v čase

Rozdílná syntéza může také způsobit posun zvuku v čase. Syntetizovaný zvuk tedy začne a skončí o něco později nebo dříve. Případně může dojít k prodloužení zvuku a to jak nechtěně, tak i pomocí rozdílného nastavení parametrů. Jelikož algoritmus porovnává okna o velikosti 23.22ms, tak i několika milisekundový posun zvuku může způsobit porovnávání nesprávných (nepárových) oken mezi sebou. V tomto by nám mohl pomoci zmíněný algoritmus *Earth Moving Distance*, jelikož je schopný porovnávat několik nepárových oken a najít pár, který si je nejbližší.

Nakonec jsem se rozhodl uchýlit k řešení, které nabízí ve své práci Muda [29]. K porovnávání MFCC využívá *Dynamické borcení časové osy* (DTW - *Dynamic time warping*), což se nakonec zdá být běžnou praxí. Tato metoda slouží k porovnávání nepárových oken podobně, jako to dělá EMD. Běžná časová náročnost je, podobně jako u EMD, $O(N^2)$. Algoritmus lze ale omezit na prohledávání omezeně vzdálených oken. Navíc je žádoucí, aby zvuk nebyl posunutý příliš. Vzhledem k velikosti oken je tedy vhodné zvolit posun maximálně kolem $\pm 60$ ms, což odpovídá zhruba velikosti 5 oken. Dalo by se tedy říci, že EMD a MFCC + DTW jsou velice podobné přístupy. Vzhledem k doporučení a k tomu, že jsem již více seznámen s funkcionalitou MFCC, jsem se rozhodl EMD v této práci nevyužít.

## 6.5 Normalizace zvuku

Během průběžného testování programu jsem narážel na problém špatné optimalizace parametrů v důsledku nízké hlasitosti cílového zvuku. Podobně na tom byl ve své práci Johnson, jelikož své výsledky porovnával pomocí spektrálních norem. Johnson se nakonec rozhodl zvuky pro porovnávání normalizovat, což problém vyřešilo.




Obrázek 6.7: Dynamické borcení časové osy pro maximální vzdálenost dvou oken

Problém se přestal vyskytovat, jakmile jsem z porovnávání odstranil první koeficient MFCC. Právě tento koeficient má za následek především porovnávání hlasitostí. Jelikož je možnost zapojení tohoto koeficientu do porovnávání volitelná, umožnil jsem programu normalizovat zvuky pomocí přepínače. Tuto volbu nechávám na uživateli. Vzhledem k běžné praxi však nebudu implicitně uvažovat použití prvního koeficientu a tedy ani normalizace.

# Kapitola 7

# Program pro aproximaci parametrů syntetizéru a resyntézu zvuku

V rámci této práce byl vytvořen program, který je schopen resyntézy zvuku. Resyntéza je provedena optimalizací parametrů pomocí genetického alegoritmu. Tyto parametry jsou následně využity pro samotnou syntézu zvuku.

Kromě základní optimalizační funkce program:

- spravuje a zobrazuje vstupy a výstupy
- programově a uživatelsky komunikuje s virtuálním syntetizérem ve formátu VST
- obsahuje GUI pro ovládání vstupů, výstupů, manipulaci s programem a vizualizaci syntetizéru
- využívá algoritmus MFCC pro vyhodnocování podobnosti zvuků
- analyzuje zvuky a výsledky analýzy využívá ke zrychlení procesu genetického algoritmu
- informuje o stavu
- generuje kombinace parametrů, zvuků podobných cílovému

Program byl vyvíjen v programovacím jazyku Java a pro grafickou část byla využita knihovna SWT. Program by měl být multiplatformní, ale byl vyvíjen a testován na platformě *Windows*. Ostatní platformy by měly být z principu podporovány, ale funkčnost nebyla otestována.

Java, jako programovací jazyk, byla vybrána hned z několika důvodů:

- V porovnání se skriptovacími jazyky, jako je LAPSDA je rychlá a navíc v posledních letech prodělala významné změny, které ji činí rychlejší, než tomu bylo v předchozích letech. [5]
- Má objektově orientovanou architekturu, což mi usnadnilo vývoj genetického algoritmu a rychlejší prototypování analžzy zvuku a zkoušených porovnávacích algoritmů.
- Je multiplatformní.
- Existuje pro ni mnoho knihoven s algoritmy pro analýzu zvuku a propojení s VST.

## 7.1 Vstupy a výstupy programu

Základním vstupem pro celý program je v prvé řadě zvuk, který má program resyntetizovat, tedy zvuk cílový. Formát vstupního souboru jsem omezil na typ **WAV se vzorkováním 44100Hz, o 16 bitech na jeden vzorek se znaménkem a se zápisem v pořadí little-endian**. Počet kanálů by neměl mít vliv, ale program byl testován na 2 kanálech.

Aby mohl být tento cílový zvuk resyntetizován, musí být použit syntetizér. Program nevyužívá žádný obecný syntetizér ani žádný svůj vlastní, syntetizér je jeden ze základních vstupů programu. Syntetizér se programu předává jako **VST knihovna** specifická pro danou platformu. Není zaručeno, že všechny syntetizéry budou plně funkční, jelikož komunikace probíhá skrze knihovny, které samy o sobě nezaručují 100% funkčnost pro všechny syntetizéry. Jak již bylo řečeno, syntetizéry se rozdělují na 2 typy. V této práci se zaměřujeme na ty syntetizéry, které mají jako vstup posloupnost MIDI událostí, tedy na VST instrumenty.

Tím se dostáváme k dalšímu vstupu programu a tím jsou právě tyto MIDI události. Tento vstup by bylo možné odhadnout z cílového zvuku, avšak to je již nad rámec této práce a navíc neexistují metody, které by obecně fungovaly s velkou přesností. Dalším vstupem je tedy posloupnost MIDI událostí a to jako **soubor ve formátu MID**.

Vedlejšími vstupy programu jsou 2 celá čísla určující násobnost syntézy (viz 6.4 - implicitně 1) a velikost turnajové skupiny (impliticně 7).

Výstup programu je dvojí. Primárním výstupem je seznam všech vygenerovaných kombinací parametrů včetně ohodnocení zvuků, pro jejiž syntézu byly dané parametry využity v syntetizéru. Seznam je dostupný skrze tabulku v GUI, kde jsou vidět pouze ohodnocení. Po kliknutí na ohodnocení dojde k načtení parametrů tohoto ohodnocení do další tabulky, kde jsou vidět konkrétní hodnoty parametrů. Pomocí ovládacích prvků lze nastavit hodnoty parametrů přímo do syntetizéru. Následně je možné zvuk nechat

znovu syntetizovat pomocí nastavených parametrů a poslechnout si jak zní a případně ho uložit. Tím se dostáváme k sekundárnímu výstupu programu a tím je právě soubor se zvukem. **Tento soubor má stejný formát jako zvuk vstupní**.

## 7.2 Volba frameworku pro komunikaci s VST

Pro komunikaci s VST je nutné vytvořit buď vlastní knihovnu nebo použít již existující. Tato knihovna by měla poskytovat přístup k několika nativním metodám pro práci s VST. Tyto nativní metody jsou dále implementovány v knihovně specifické pro operační systém. Mezi tyto metody patří především:

- long loadPlugin(String pluginFile) - načte VST z cesty uvedené v parametru a vrátí jeho pointer
- void setParameter(int index, float value), float getParameter(int index) - nastaví / vrátí hodnotu parametru VST
- int numParameters(long pluginPtr) - vrátí počet parametrů VST
- int numInputs(long pluginPtr), int numOutputs(long pluginPtr) - vrátí počet vstupů/výstupů VST (resp. počet kanálů)
- void processReplacing(MidiMessage[] messages, float[][] inputs, float[][] outputs, int blockSize, long pluginPtr) - zpracuje předané MIDI zprávy a vstup do výstupu pro danou velikost na daném VST

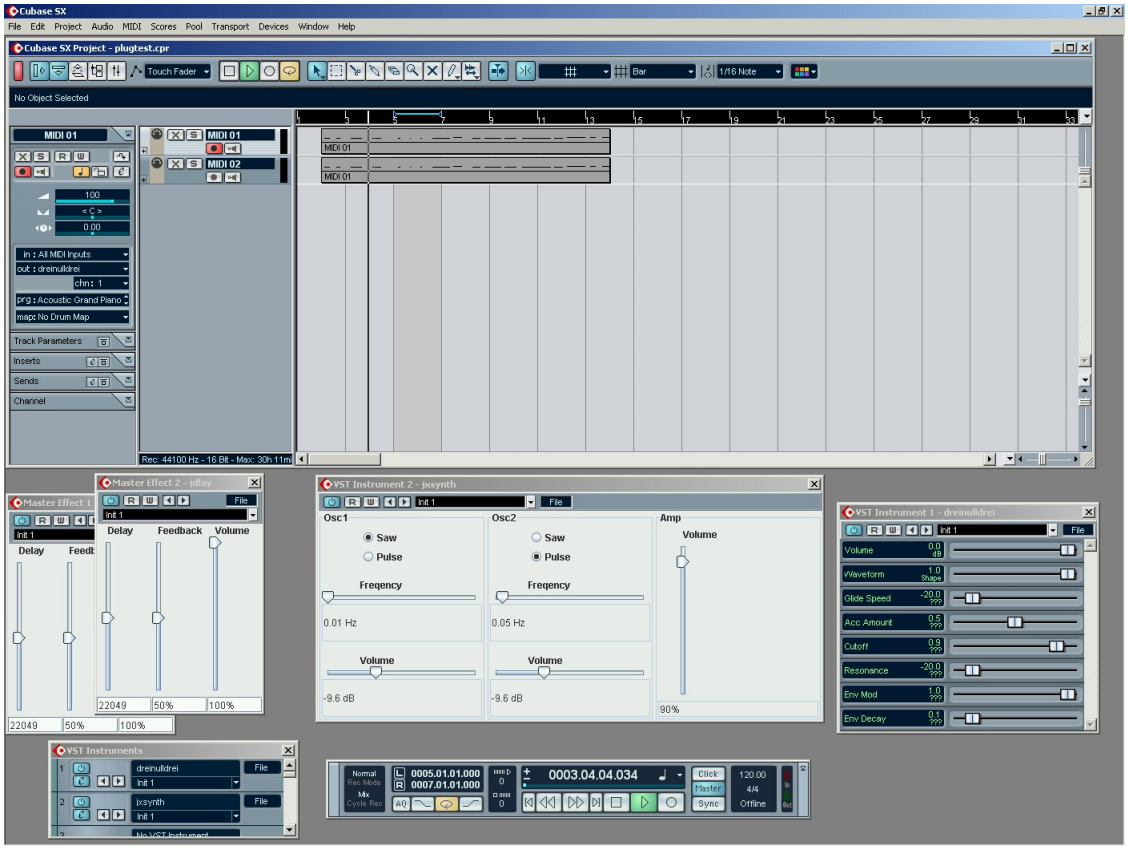
Vyzkoušel jsem tedy hned několik knihoven a stanovil jsem si několik podmínek, které musí být splněny.

- První podmínka byla možnost hostování syntetizéru ve formátu DLL, jelikož pracuji převážně na platformě Windows.
- Další podmínkou byla možnost zpracování syntézy zvuku pomocí nahostovaného syntetizéru za daných parametrů.
- Poslední podmínkou bylo umožnění programového ovládání parametrů syntetizéru.

Výhodou, kterou by mohl framework nabízet, by bylo načtení původního grafického uživatelského rozhraní syntetizéru. Pokud by žádný framework nesplňovat povinné podmínky, byl bych nucen vytvořit framework vlastní, případně upravit nějaký již existující. K tomu však nedošlo.

### 7.2.1 Nástroj jVSTwRapper

První framework, který jsem vyzkoušel se jmenuje *jVSTwRapper*. Tento framework se zdál velmi propracovaný a komplexní. Spíše než frameworkem bych jej nazval přímo open-source nástrojem pro práci s hudbou a syntetizéry. Avšak ani po několika hodinách se mi jej nepodařilo zprovoznit tak, abych s ním dokázal pracovat, jak je třeba. Navíc s největší pravděpodobností nepodporuje původní grafické rozhraní syntetizérů, ale využívá pro parametry své vlastní. Nechal jsem si tedy tento nástroj v záloze a vyzkoušel jiné řešení.




Obrázek 7.1: *jVSTwRapper*, který nabízí široké možnosti v hostování syntetizérů [24]

### 7.2.2 Framework JVST

Další framework, který jsem vyzkoušel, se jmenuje *JVST*. Na první pohled se zdál až příliš primitivní. Vydán byl pouze v jedné verzi a tou je v0.0.1. Když jsem zkoumal kód, uvědomil jsem si, že by mohl splňovat základní požadavky, které jsem si stanovil. Dokázal načíst a nahostovat VST, i když v základu nedovoloval programovou manipulaci s ovládacími prvky. Dokonce jsem pomocí tohoto frameworku úspěšně syntetizoval vzuk.

Ale ani to nebylo jednoduché. Jelikož bylo třeba, aby byla data zvuku dodána v určitém formátu, musel jsem nejprve pochopit podstatu zápisu zvuku do různých formátů. Stejně tak bylo třeba pochopit, jak má framework tato data přijímat a co dělat například v případě, mám-li k dispozici více kanálů, atp., jelikož byl k dispozici pouze jeden příklad vysvětlující použití v jednom specifickém případě. Po několika neúspěšných pokusech o zahákování událostí jsem ale práci s *JVST* vzdal. Tento framework nabízí původní grafické rozhraní syntetizéru, čehož jsem nakonec využil.

### 7.2.3 Framework JVSTHost

Framework *JVSTHost* umožňuje jednoduše nahostovat syntetizér z *Microsoft* knihoven typu DLL a to včetně původního grafického rozhraní, ale měl by stejně dobře fungovat i na ostatních systémech. Dále umožňuje relativně jednoduché zahákování událostí manipulace se syntetizérem a jeho programové ovládání. Menším problémem byla kompilace pod systémem *Windows*. Tento framework splňuje všechna kritéria, podporuje VST ve verzi od 2.0 do 2.4, má vlastní zpracování výjimek a je poskytován s jednoduchým GUI pro MIDI klávesy. Bohužel původní GUI k syntetizéru chybí.

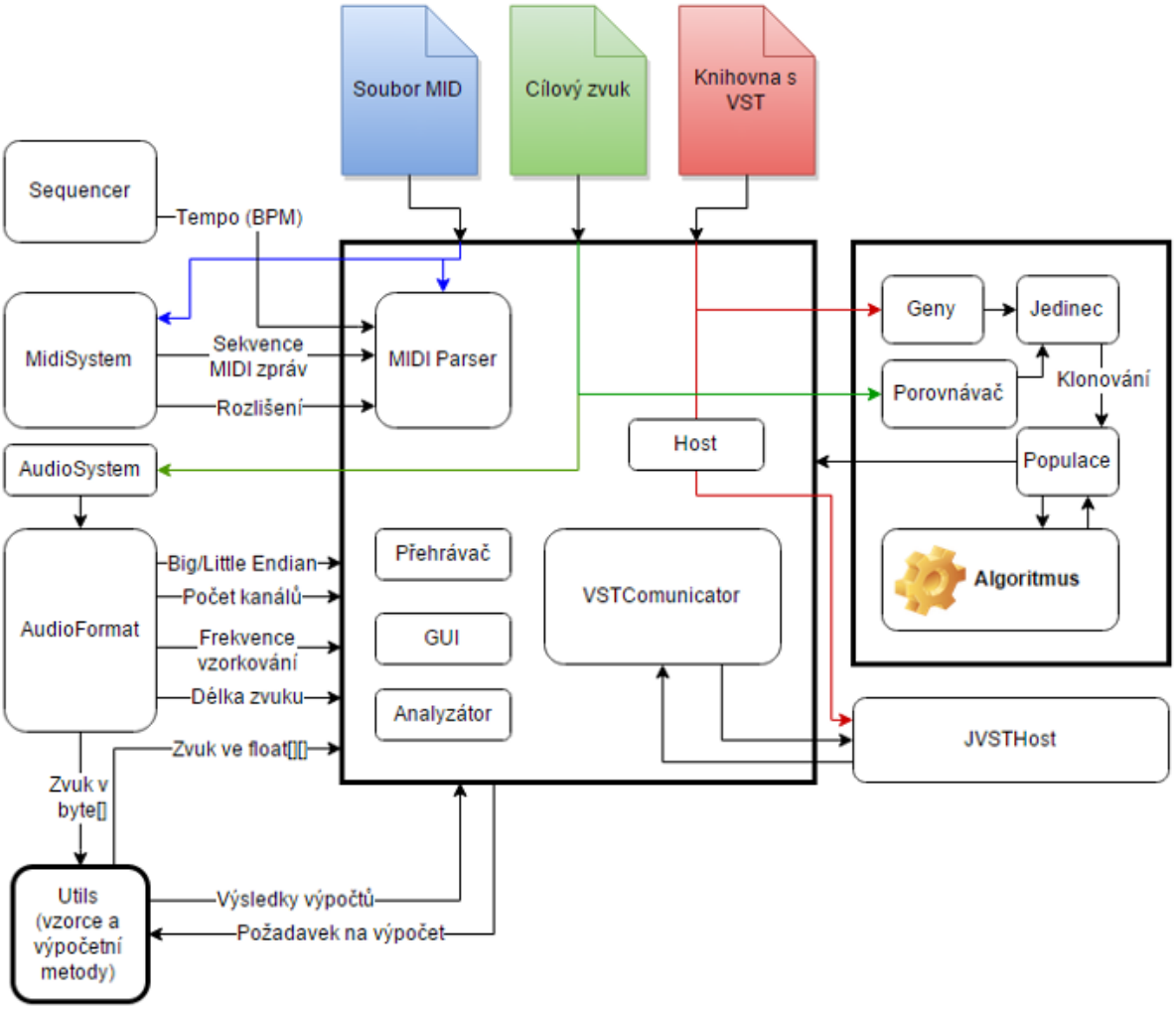
Nakonec jsem se rozhodl využít dva frameworky. *JVST*, které bude poskytovat grafické rozhraní syntetizéru a JVSTHost, který využiji pro interní komunikaci programu.

Co se podpory verzí VST týče, od roku 2008 je k dispozici SDK pro vývoj VST ve verzi 3.0. V únoru 2011 byla vydána verze 3.5, která umožňuje vyprodukovat VST i ve verzi 2.x. V prosinci 2013 Steinberg ukončil distribuci SDK verze 2.0. Vzhledem k faktu, že lze vyprodukovat VST ve verzi 2.x s SDK verze 3.5, nebudu se příliš zaobírat omezeností knihovny pro komunikaci s VST. Nakonec vytvořit knihovnu pro komunikaci s VST není až tak složitá záležitost, v této práci se tím zabývat ale nebudu.

## 7.3 Komunikace s VST

Pro komunikaci s VST jsem si tedy vybral vybral hned dva frameworky s tím, že jejich fungování zkombinuji. Prvním je *JVSTHost*, ten však obstarává pouze koncové body komunikace a druhý je *JVST*, který VST pouze zobrazuje. Program využívá pouze některé základní metody, které framework poskytuje. Tyto metody popisuje diagram komunikace programu B.2. Většina základních operací je 1:1, ostatní jsou o něco komplexnější a u nich se na chvíli zastavím.

Metoda *setParameterValue*, kterou volá program na základě změny hodnoty skrze GUI či genetický algoritmus musí volat metodu *setParameter* na objektu JVST, tak na objektu JVSTHost. Objekt JVST pouze změní svůj aktuální vzhled, dle nastaveného parametru. JVSTHost změní hodnotu parametru jako takového.




Obrázek 7.2: Zjednodušený diagram částí programu (tučně ohraničené části jsou součástí vlastní implementace)

Metoda *getParameterStepValue* se využívá k získání nejmenší hodnoty, o kterou se může parametr změnit. Toho je dosaženo postupným zvyšováním hodnoty o 0.01 od 0 a kontrolou, zda se hodnota již změnila. Změna hodnoty proběhne v případě, že byla dosažena alespoň polovina hodnoty následující. Algoritmus tedy předpokládá, že jakýkoli parametr s více jak 50 hodnotami ($\frac{1}{(0.01 \cdot 2}$) je parametr se spojitými hodnotami. Získání této hodnoty je důležité pro genetický algoritmus u parametrů, které slouží jako přepínače.

Následují dvě metody, *processMidiEvents* a *sendMidiEvents*. Tyto metody se liší ve výstupu. Zatímco *processMidiEvents* vrací celý syntetizovaný zvuk ve formátu float[][], *sendMidiEvents* bere postupně MIDI události a nechává je postupně zpracovat ve stejně velkých dávkách, přičemž výsledek postupně přehrává. Dalším rozdílem, který by bylo vhodné zmínit je zpsůsob předávání MIDI událostí.

U *sendMidiEvents* se bere jedna událost za druhou a algoritmus si pamatuje tick poslední zpracované události. Následující událost způsobí zastavení zpracovávání na dobu, která se rovná rozdílu posledního a aktuálního ticku. Tato doba je následně přepočtena na milisekundy pomocí vzorce 2.3. Během pozastavení algoritmu běží v programu další vlákno, které zajišťuje postupné přehrávání výstupu ve stejně velkých dávkách. Dobu po jakou budou dávky zpracovávány pro jednu událost v tomto případě určuje doba pozastavení algoritmu, protože po opětovném rozběhnutí algoritmu dojde k odeslání další události, Je nutno podotknout, že události se mohou překrývat a zpracovávání pak probíhá pro více událostí najednou. V případě, že žádná další událost nebude odeslána, bude program nadále přehrávat zbytkový[1] zvuk. Dalo by se tedy říci, že o výstup se zde stará vlákno, které si neustále bere výstup o stejné velikosti a tento výstup přehrává, přičemž časové pauzy mezi událostmi jsou řízeny pozastavením algoritmu.

Jakmile se však budeme chtít vyhnout pozastavením, nelze využít tuto metodu zpracovávání. Doba zpracovávání se v případě pozastavování rovná délce syntetizovaného zvuku, což může být několik vteřin. Pokud bychom pak chtěli získávat syntetizovaný zvuk pro několik stovek kombinací poskytnutých genetickým algoritmem, trvalo by to i několik hodin. Proto je nutné se pozastavením vyhnout. Toho lze docílit změnou velikosti zpracovávaných dávek. Tato velikost je vypočtena ze vzorce 2.3 a následně přepočtena dle vzorkovací frekvence. Co se zbytkového zvuku týče, ten se připojí k výsledku po zpracování všech dávek, které odpovídají MIDI událostem. Konec je však ustřižen dle délky vstupního cílového zvuku.

[1] Zbytkovým zvukem je myšlena například ozvěna nebo release (u ADSR obálky tónu).

# Kapitola 8

# Testování programu a experimenty

V této kapitole rozebereme, jak proběhlo testování programu, shrneme si jeho výsledky a popíšeme některé získané poznatky. Zároveň si ukážeme výsledky experimentování s parametry MFCC, okénkovací funkce a selekční metody GA.

## 8.1 Testování programu

Testování programu jsem porovedl celkem na 16 zvucích a 6 syntetizérech. Testování bylo zaměřeno kvalitativně, nikoliv kvantitativně. Veškeré výsledky jsem zaznamenal do přiložených souborů (viz D). Výsledkem testování je nejpodobnější syntetizovaný zvuk ve formátu WAV, hodnoty jedinců v průběhu syntézy v tabulce Excel a ruční zápis s poznámkami.

### 8.1.1 Nastavení testů

Pro testování byly využity následující konstanty (odůvodnění jejich hodnot naleznete v předchozích kapitolách):

- Velikost populace: 100
- Velikost turnajové skupiny: 7
- Počet syntetizovaných vzorků pro každé nastavení: 5
- Počet generací: 20
- Elitismus: 10%

- Mutace: 5%
- Parametr překročení pro křížení[1]: 20%
- Parametr pro Hammingovu funkci okénkování: 0.46
- Velikost okna pro MFCC: 23.22ms
- Počet MFCC: 12 (resp. 1+12) - nevyužitá suma energií
- Počet filtrů MFCC: 23
- Vzdálenost pro DTW: 5
- Normalizace: Ne
- Spodní frekvenční hranice filtrů MFCC: 133.3 Hz
- Horní frekvenční hranice filtrů MFCC: 6855.5 Hz

Pro testování bylo využito 12 jednotónových zvuků a 4 třítónové [2]. Jednotónové zvuky jsou rozděleny do dvou skupin. Jedna skupina byla syntetizována ze čtvrťové noty C2 a druhá ze čtvrťové noty E3 při BPM=120. Třítónové zvuky byly syntetizovány taktéž ze čtvťových not, ale tentokrát C3, F3 a B3 (resp. H3). Testy byly provedeny několikrát a byly vyřazeny výsledky, které se dlouhodobě neopakovaly.

BPM=120 je standartní hodnota a čtvrťová nota má při ní pro účely testování vhodnou délku. Předpokládam, že výška noty, či BPM by neměly mít vliv na výsledky programu.

K syntéze 12 jednotónových cílových zvuků bylo využito 6 syntetizérů (pokaždé s jiným nastavením) a pro každý syntetizér 2 jednotónové zvuky (jeden C2 a druhý E3):

- DVS Megabass
- Kicker
- M-Box
- Monolisa
- DSK AsianZ DreamZ
- S7-Synth

[1] Viz rozšířené obdélníkové křížení 3.2

[2] Tím je myšlen počet not použitých při syntéze.

Pro syntézu třitónových zvuků byly použity syntetizéry Monolisa, DVS Megabass, M-Box a S7-Synth.

Každý z vybraných syntetizérů se vyznačuje specifickými zvuky, které dokáže syntetizovat. Testy jsem se snažil pokrýt co největší škálu syntetizérů, abych zjistil, zda je řešení v této oblasti robustní.

**DVS Megabass** Syntetizér pro jednoduchou syntézu basových zvuků s možností změny hlasitostní obálky pomocí jednoho parametru. Je také možné nastavit tvar vlny oscilátoru (2 parametry), filtr (2 parametry) a poupravit efekt (2 parametry). Celkem tedy tento syntetizér má 7 nastavitelných parametrů.

**Kicker** Úderový syntetizér s nastavitelnou výškou a hloubkou tónu (2 parametry), filtrací (2 parametry) a zesilovačem (3 parametry). Celkem tedy 7 nastavitelných parametrů.

**M-Box** Velice komplexní syntetizér pro tvorbu zvonivých a řehtačkových zvuků. Obsahuje parametry pro nastavení výšek, rychlostí, obálek, podtónů, hlasitosti, dozvuku a typu zvuku (25 zvonivých a 6 řehtačkových). Celkem má 17 nastavitelných parametrů, které však ovládají 25 vnitřních parametrů. Což je problém, jelikož program pracuje s vnitřními parametry, nikoliv s těmi uživatelsky nastavitelnými. Proto nedochází k vizuální změně hodnot parametrů při práci programu. Tento syntetizér jsem chtěl vyzkoušet právě z tohoto důvodu, zajímalo mě, zda si s tím program poradí. Zároveň produkuje vyšší tóny vzhledem k předchozím dvěma syntetizérům.

**Monolisa** Tento syntetizér ovládá 34 parametrů, přičemž 6 ovládá oscilátor, 5 filtr, 5 obálku, 5 zesilovač, 6 nízkofrekvenční oscilátor a 7 je určeno pro další efekty (hlasitost, lazení, atp.). Pomocí nich je schopen vyprodukovat velkou škálu bzučivých zvuků, přes nízké basové po velmi vysoké Tento syntetizér jsem jsem vybral, abych vyzkoušel, jak si program poradí s větším počtem parametrů a s nízkofrekvenčním oscilátorem, u kterého jsem předpokládal problémy.

**DSK AsianZ DreamZ** Syntetizér schopný napodobit 7 asijských nástrojů. Syntetizér obsahuje parametry pro ovládání obálky (5) a filtr (4). Celkem tedy 10 parametrů. Syntetizér jsem do testování zahrnul, abych otestoval, zda je program schopný pracovat se zvuky nástrojů a ne pouze s typickými syntetizačními zvuky.

**S7-Synth** Standartní syntetizér s možností ovládání tvaru vlny oscilátoru, filtrací a obálkou. Syntetizér produkuje zvuky typické pro jednotlivé tvary signálu oscilátoru. K

dispozici je celkem 10 parametrů, 4 na ovládání obálky, 2 pro filtraci, 1 pro tvar vlny oscilátoru, 1 pro lazení a poslední ovládá hlasitost.

### 8.1.2 Výsledky testů

Výsledky budu prezentovat numericky i slovně, přičemž slovní hodnocení je subjektivní a hodnocené pomocí poslechu. Všechny výsledky jsou uloženy v příslušných složkách a souborech na přiloženém DVD.

Jako referenční řešení jsem se rozhodl nechat program ohodnotit zvuk, který bude syntetizovaný z manuálně nastavených parametrů syntetizéru tak, aby tento zvuk odpovídal poslechově co nejvíce cílovému zvuku. Pro syntézu jsem tedy použil přibližné hodnoty parametrů, které byly použity při původní syntéze cílového zvuku, a tím tak simulovat reálné ruční hledání parametrů[3].

Nejlepší hodnotu normalizované kvality jedince, který vytvořil program jsem označil jako $A$ (syntetizovaný zvuk). Kvalitu jedince, jehož nastavení proběhlo manuálně dle výše uvedených podmínek, jsem označil $M$ (kontrolní zvuk). Kvalita je normalizována vzhledem k délce zvuku. S délkou roste počet oken, které jsou porovnávány, a tedy i výsledná vzdálenost (resp. kvalita) zvuků. Tímto způsobem je možné hodnocení porovnávat mezi sebou. Tabulka 8.1 dále ukazuje v jaké generaci bylo výsledku dosaženo a subjektivní hodnocení podobnosti ve škále 1-10, kde 10 znamená nejlepší a 1 nejhorší shoda zvuků. Hodnocení prováděly dvě osoby nezávisle na sobě a výsledkem je průměr hodnocení. Pod tabulkou je dále slovní hodnocení.

#### 8.1.2.1 Výsledky jednotónových zvuků

**1** Syntetizovaný zvuk prakticky nerozeznatelný od cílového. Parametry odhalily velmi slabou odchylku v nastavení tvaru vlny oscilátoru, která je však neslyšitelná.

**2** Rychlejší nástup syntetizovaného zvuku, což ho činí o málo hlasitějším. Jinak jsou zvuky poslechově nerozeznatelné.

**3** U syntetizovaného zvuku je tón o něco hlubší. Kontrolní zvuk má sice horší ohodnocení kvality, ale výsledný zvuk je poslechově podobnější, než ten, který syntetizoval program automaticky.

[3] I kdyby hodnoty parametrů byly naprosto shodné s hodhotami, které byly využity pro syntézu, nezaručuje to výslednou hodnotu kvality 0, jelikož se zvuk pokaždé syntetizuje trochu jinak.

| Číslo | Syntetizér | A | M | Generace | Hodnocení |
|---|---|---|---|---|---|
| 1 | DVS Megabass | 1.144 | 1.097 | 20 | 8 |
| 2 | DVS Megabass | 0.517 | 0.486 | 17 | 9 |
| 3 | Kicker | 0.653 | 1.045 | 20 | 8 |
| 4 | Kicker | 0.925 | 0.723 | 19 | 9 |
| 5 | M-Box | 0.817 | - | 19 | 7 |
| 6 | M-Box | 1.169 | - | 20 | 8 |
| 7 | Monolisa | 1.695 | 1.877 | 15 | 5 |
| 8 | Monolisa | 1.554 | 1.298 | 20 | 4 |
| 9 | DSK AsianZ DreamZ | 1.493 | 2.022 | 13 | 4 |
| 10 | DSK AsianZ DreamZ | 0.014 | 1.033 | 17 | 10 |
| 11 | S7-Synth | 0.262 | 0.124 | 20 | 8 |
| 12 | S7-Synth | 0.503 | 0.491 | 20 | 10 |

Tabulka 8.1: Výsledky testování jednotónových zvuků

**4** Cílový zvuk je o něco důraznější a pravděpodobně i kvůli tomu méně čistý než zvuk syntetizovaný. Jinak jsou zvuky totožné. Program opět ohodnotil jeho automaticky syntetizovaný zvuk jako lepší než ten, u kterého byly parametry nastaveny manuálně. Kontrolní byl opět poslechově blíže.

**5** Cílový zvuk se je opět o něco důraznější a hlasitější. Charakter zvuku je podobný.

**6** Syntetizovaný zvuk je znatelně tišší. Při zesílení však bylo zjištěno, že charakteristika je prakticky totožná. Manuálně nastavené parametry neumožnily syntézu kvalitního kontrolního zvuku. Problémem je, že vnitřní parametry syntetizéru nekorespondují s těmi, které uživatel může nastavovat (viz dále).

**7** Syntetizovaný zvuk je charakteristicky stejný, ale trpí kvůli špatně nastavenému nízkofrekvenčnímu oscilátoru. S tímto problémem jsem se potýkal i při manuálním nastavení parametrů.

**8** Špatná výška syntetizovaného zvuku, správně nastavený nízkofrekvenční oscilátor. Zvuk má také nižší hlasitost a pravděpodobně trpí špatně nastaveným filtrem.

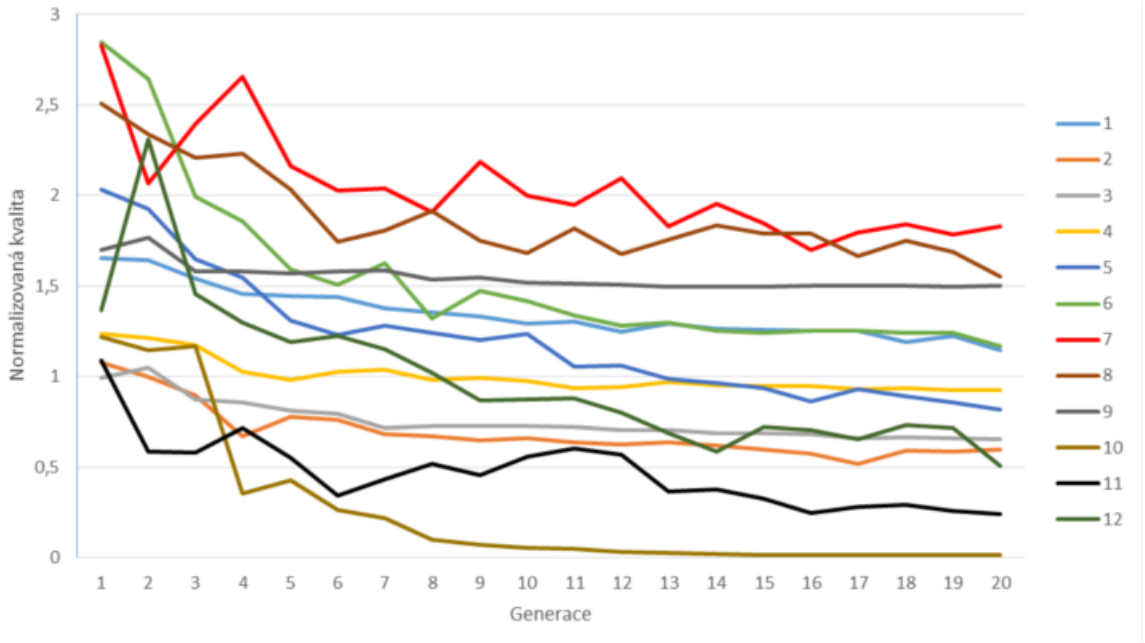
**9** Zvuk je svým způsobem podobný, ale jelikož byl programem zvolen špatný nástroj (parametr syntetizéru), je zvuk v tomto ohledu odlišný. Vzhledem k hodnotám $A$ a $M$ je chyba očividně na straně porovnávacího algoritmu, který dva poslechově stejné zvuky hodnotí hůře, než dva více rozdílné.

**10** Nerozeznatelné zvuky. Vzhledem k hodnotě *A* se není čemu divit.

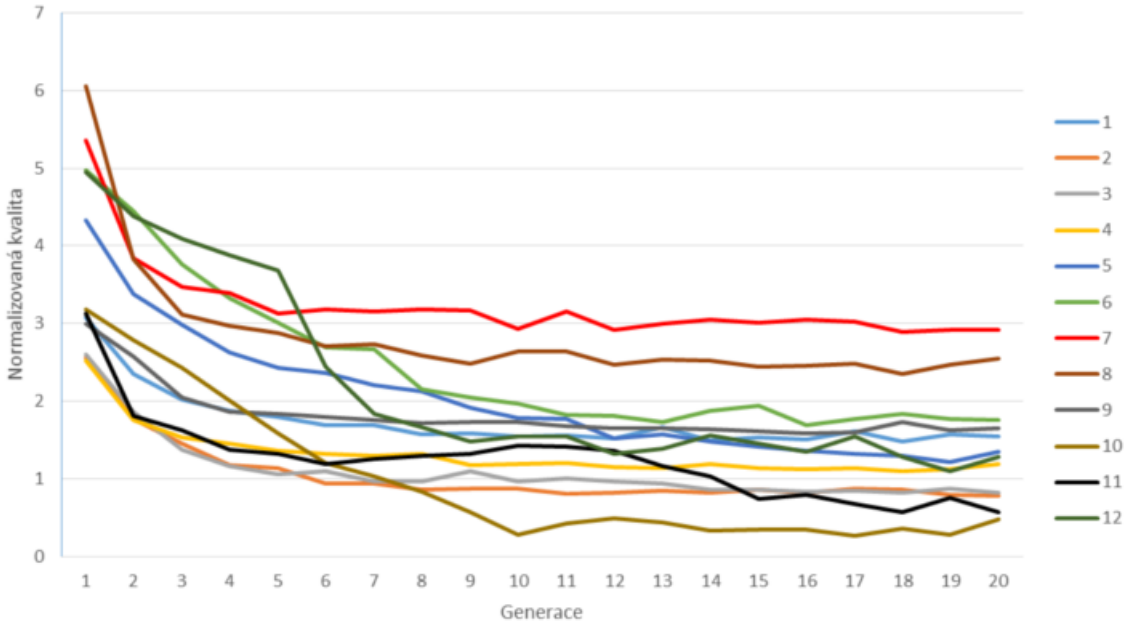
**11** Syntetizovaný zvuk měl okamžitý nástup, přičemž cílový zvuk měl nástup postupný. Charakter zvuku je totožný.

**12** Cílový a syntetizovaný zvuk jsou od sebe nerozeznatelné.

Fungování syntetizéru M-Box bylo v průběhu testování pochybné. Problémem je, že nastavitelné parametry nekorespondují s těmi, pomocí kterých je výsledný zvuk syntetizován (s těmi program pracuje). Dalším problémem je, že jedna z hodnot se pohybuje mimo běžný hodnotový rozsah. Testování to však neovlivnilo, jelikož jediným problémem byla nemožnost vytvořit korektní kontrolní zvuk, což není příliš důležité.




Obrázek 8.1: Vývoj kvality nejlepších jedinců v průběhu genetického algoritmu pro jednotlivé testovací případy jednotónových zvuků




Obrázek 8.2: Vývoj průměru kvality jedinců v průběhu genetického algoritmu pro jednotlivé testovací případy

| Číslo | Syntetizér | A | M | Generace | Hodnocení |
|---|---|---|---|---|---|
| 1 | Monolisa | 0.479 | 0.487 | 17 | 5 |
| 2 | DVS Megabass | 0.588 | 0.633 | 19 | 8 |
| 3 | M-Box | 1.870 | - | 20 | 4 |
| 4 | S7-Synth | 0.523 | 0.628 | 18 | 8 |

Tabulka 8.2: Výsledky testování třítónových zvuků

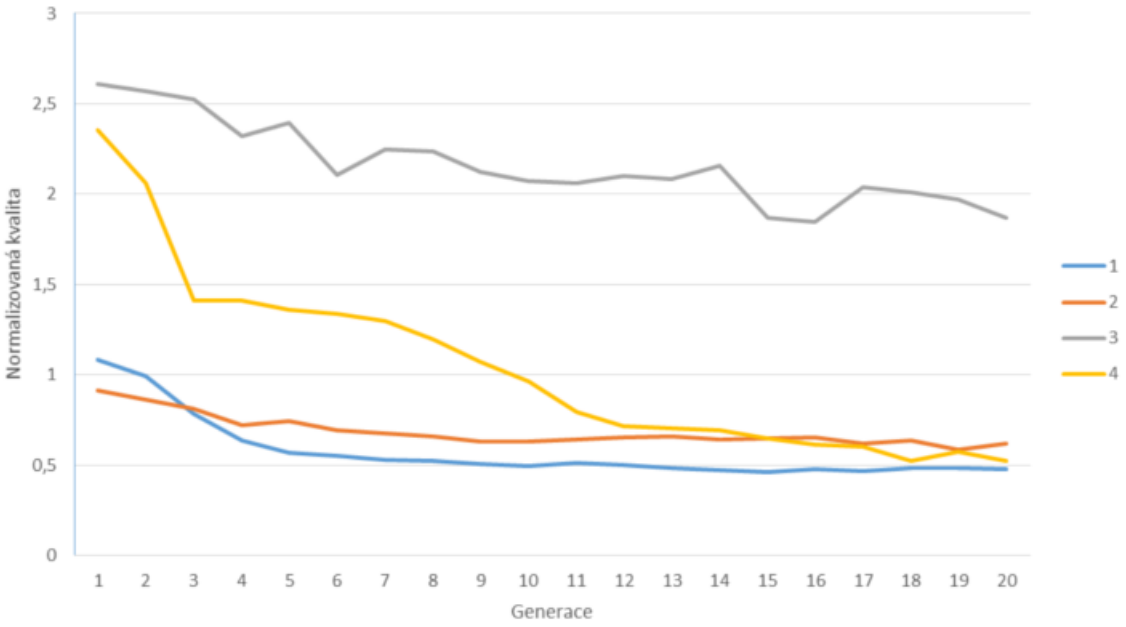
#### 8.1.2.2 Výsledky třítónových zvuků

**1** Syntetizovaný zvuk se vyznačuje pomalejším nástupem, rychlejším ústupem, nižší hlasitostí a nečistostí tónů. Odlišná je také výška, která je nižší. Problém s tímto syntetizérem je i u jednotónových zvuků.

**2** Syntetizovaný zvuk je proti cílovému hlubší a má menší důraz. Program však syntetizovaný zvuk ohodnotil lépe, než zvuk kontrolní, který byl poslechově nerozeznatelný.

**3** Syntetizovaný zvuk postrádal zvonivý efekt, byl hlubší a více na pozadí. Cílový zvuk byl proti němu čistý a jasný. Tomu odpovídá i normalizovaná hodnota kvality, která je značně vysoká.

**4** Cílový zvuk má proti syntetizovanému okamžitý nástup, charakter mají je však velmi podobný. Syntetizovaný zvuk má nástup velmi pozvolný. Zvuky se také liší v hlasitosti.



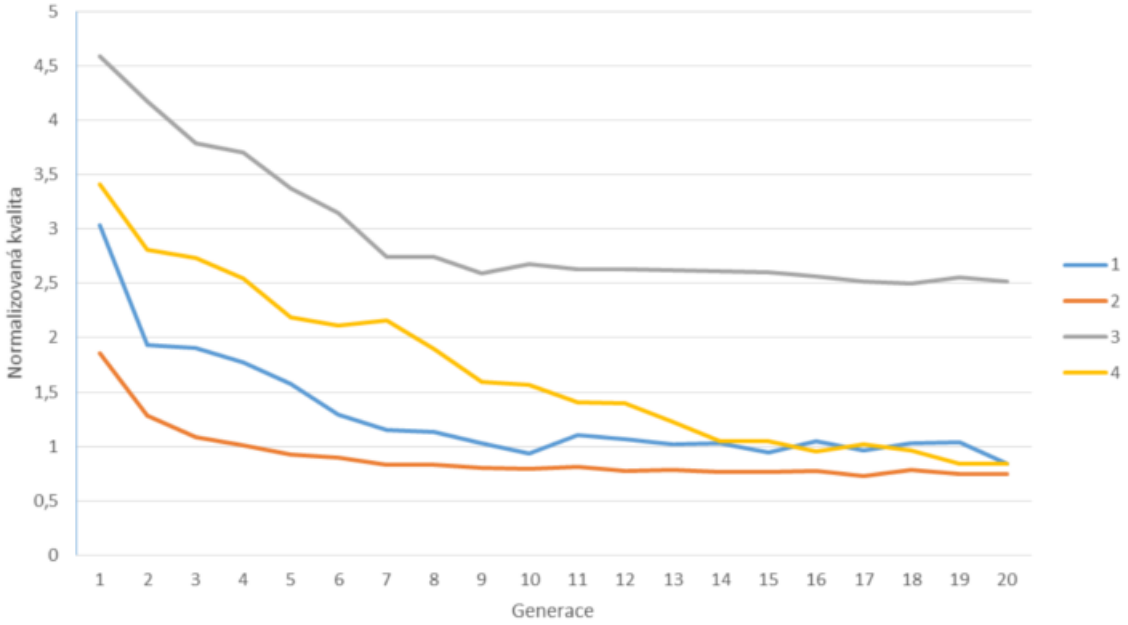

Obrázek 8.3: Vývoj kvality nejlepších jedinců v průběhu genetického algoritmu pro jednotlivé testovací případy

### 8.1.3 Shrnutí testování programu

Z testování je očividné, že větší úspěšnost na reprodukci podobného zvuku mají syntetizéry s méně parametry a ty, které produkují menší škálu zvuků. To je zřejmě dané množstvím možných kombinací parametrů, tedy především tím, že algoritmus může spíše najít takové řešení, které mu přijde lepší. S tím souvisí určitá nepřesnost porovnávací metody MFCC. V určitých případech hodnotí poslechově jasně odlišný zvuk lépe, než zvuk, který je od cílového nerozeznatelný prakticky nerozeznatelný.

U většiny testovacích případů je také vidět, že většina nejlepších řešení se vyskytuje v posledních generacích, je tedy otázkou, jak moc by se řešení zlepšilo, pokud by algoritmus pokračoval dále, např. do generace 50. Podle všeho by to ale příliš nepomohlo. Algoritmus většinou našel uspokojivé řešení do 20 generace (největší pokrok je pak vidět kolem generace 5) a pokud ne, slouzl k takovému řešení, které bylo ohodnocené lépe, než zvuk kntrolní, přitom bylo očividně poslechově nepřesné.

Program má také problém při použití syntetizéru s nízkofrekvenčním oscilátorem, jelikož ten může měnit povahu zvuku natolik, že je při každé samostatné syntéze




Obrázek 8.4: Vývoj průměru kvality jedinců v průběhu genetického algoritmu pro jednotlivé testovací případy

velmi odlišný. S tímto problémem se částečně vypořádá možnost syntetizovat zvuk pro každého jedince několikát, nejde však o dokonalé řešení.

Obecně lze říci, že syntéza podobného zvuku dopadá lépe u méně komplexních syntetizérů a u zvuků, které jsou dostatečně hlasité. Kritickým bodem je porovnávání, které dokáže svést algoritmus na špatnou cestu.

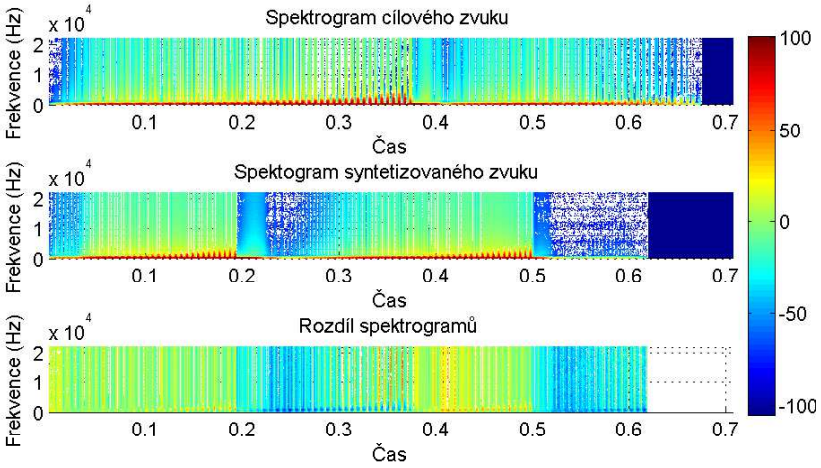
#### 8.1.3.1 Ostatní poznatky

**Rychlost** Doba zpracování je závislá na délce cílového zvuku se kterou roste lineárně. Záleží také na počtu syntetizovaných zvuků pro jednoho jedince, s tímto parametrem rychlost roste také lineárně. Počet filtrů MFCC rychlost znatelně neovlivňuje, stejně tak počet parametrů syntetizéru.

**Třítónové zvuky** Třítónové zvuky byly zapojeny do testování z důvodu předpokladu lepších výsledků. Algoritmus měl ke srovnání delší časový úsek a větší škálu zvuků (vzhledem k výšce tónů). Úspěchu však dosaženo nebylo. Numerické výsledky jsou podobné jako u jednotónových zvuků, nicméně hodnocení pomocí poslechu dopadlo hůře. Důvodem mohou být malé rozdíly v tichých oknech mezi jednotlivými tóny, které tak přispěly k lepší normalizované kvalitě.

**Hlasitost a hlasitostní obálka** Na otestovaných vzorcích je možné si všimnout, že často neodpovídá hlasitost či hlasitostní obálka cílového a syntetizovaného zvuku. To je způsobeno ignorováním prvního koeficientu MFCC, který nese informaci o celkové energii daného časového okna. Je tedy otázkou, zda tento koeficient vynechat či nikoliv. Toto rozhodnutí se pravděpodobně bude řídit uživatelským cílem syntézy. Pokud nám jde o charakteristiku zvuku, je lepší koeficient vynechat. Pokud je důležitá především hlasitost, měl by tento koeficient být zahrnut.

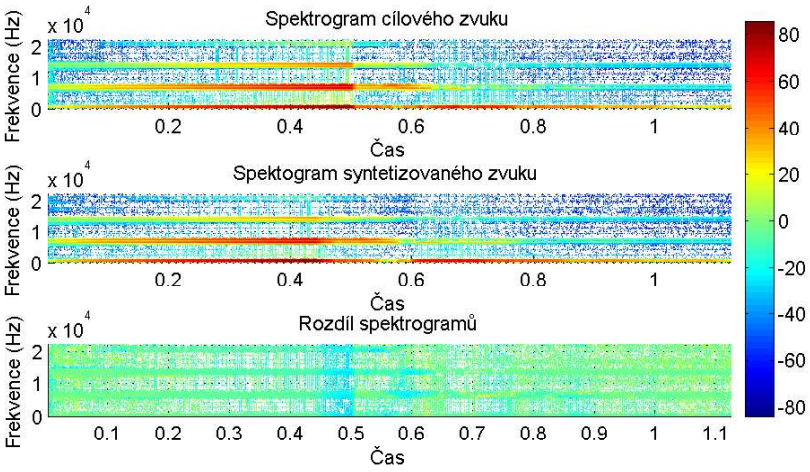
**Porovnání spektrogramů** Pro zajímavost jsem pro některé testované zvuky vytvořil spektogramy, aby bylo možné zobrazit vztah mezi kvalitou a rozdílem mezi spektogramy. Z obrázků 8.6, 8.5 a B.3 je tento vztah zřejmý. Zdá se tedy, že porovnávání spektogramů může být vhodnou součástí porovnávání zvuků. To potvrzuje ve své práci i Johnson a mnozí další.




Obrázek 8.5: Porovnání spektrogramů cílového a syntetizovaného zvuku s normalizovanou kvalitou 1.554 (8.)

## 8.2 Experimenty s MFCC

Jelikož by některé parametry MFCC mohly mít významný vliv na zlepšení výsledků porovnávání, rozhodl jsem se je podrobit testům. Konkrétně se zaměříme na počet filtrů, frekvenční rozsah filtrace a počet koeficientů. Vliv změny parametru by mohl být na různé zvuky (nízké/vysoké) odlišný, proto je třeba vybrat dobré reprezentativní vzorky. Proto jsem pro testování vybral zvuky, které jsme využili v testování 8.1 a zároveň ty, které mají nízkou i vysokou frekvencí (1, 6, 11). U těchto zvuků je také možné dobře sledovat případné zhoršení či zlepšení. I tak je však nutné počítat s ome-




Obrázek 8.6: Porovnání spektrogramů cílového a syntetizovaného zvuku zvuku s normalizovanou kvalitou 1.169 (6.)

zenou věrností, jelikož zde neprovádíme kvantitativní testy, ale pouze experimentujeme s nastavením hodnot.

Změna parametrů může mít vliv škálu hodnot kvality. Vyšší hodnota tedy automaticky neznamená horší zvuk a naopak. Proto budeme hodnotit zvuky především poslechově. Zároveň může mít změna parametru vliv na dobu vykonávání a rychlost konvergence. Tyto hodnoty je možné porovnávat s původními výsledky.

Testy budou provedeny za stejného nastavení, jako testy původní. Rozdíl nastává pouze u parametru, který je testován. Pokusy byly také několikrát opakovány, ale pokaždé bylo dosaženo stejného výsledku.

### 8.2.1 Změna počtu filtrů MFCC

V původních testech 8.1 je počet filtrů stanoven na 23. Výsledky těchto testů jsou brány jako referenční.

| Číslo | Filtrů | Doba vykonávání | Rychlost konv. | Hodnocení |
|---|---|---|---|---|
| 1 | 10 | Stejná | Rychlejší | Stejné (8) |
| 6 | 10 | Stejná | Rychlejší | Lepší (9.5) |
| 11 | 10 | Stejná | Rychlejší | Stejné (8) |
| 1 | 15 | Stejná | Rychlejší | Lepší (10) |
| 6 | 15 | Stejná | Rychlejší | Horší (7) |
| 11 | 15 | Stejná | Rychlejší | Lepší (9.5) |
| 1 | 30 | Stejná | Stejná | Stejné (8) |
| 6 | 30 | Stejná | Stejná | Stejné (8) |
| 11 | 30 | Stejná | Nekonverguje | Horší (5) |

Tabulka 8.3: Porovnání výsledků testování při změně počtu filtrů MFCC

Z testování vyplynulo, že změna počtu filtrů ovlivňuje rychlost konvergence a také samotný výsledek. Zatímco při zvýšení počtu filtrů z 23 na 30 může docházet k syntéze méně podobných zvuků a ke zhoršení konvergence, zmenšení počtu na 15 ovlivňuje výsledek pozitivně a to jak rychlostí konvergence, tak celkovým poslechovým hodnocením. Pokud počet filtrů ještě zmenšíme na hodnotu 10, vylepšení dosáhneme v jednom případě ze tří. Ač by počet filtrů měl mít vliv na dobu vykonávání, tento efekt se neprojevoval. Na provedených experimentech je vidět, že zmenšení počtu filtrů má v našem případě spíše pozitivní efekt, zatímco zvýšení má efekt negativní.

Těmito experimenty jsme chtěli především dokázat, zda může mít změna počtu filtrů MFCC vliv na celkový výsledek. Tento předpoklad byl potvrzen. Je však otázkou, zda zjištěné hodnoty platí obecně. Pravděpodobné je, že pro každý zvuk bude vhodné nastavení rozdílné.

### 8.2.2 Změna počtu koeficientů

V původních testech 8.1 je počet koeficientů stanoven na 13 (resp. 1+12). Výsledky těchto testů jsou brány jako referenční.

| Číslo | Koeficientů | Doba vykonávání | Rychlost konv. | Hodnocení |
|---|---|---|---|---|
| 1 | 8 | Stejná | Stejná | Lepší (9) |
| 6 | 8 | Stejná | Stejná | Stejné (8) |
| 11 | 8 | Stejná | Rychlejší | Stejné (8) |
| 1 | 10 | Stejná | Stejná | Lepší (9) |
| 6 | 10 | Stejná | Stejná | Horší (7) |
| 11 | 10 | Stejná | Stejná | Stejné (8) |
| 1 | 15 | Stejná | Stejná | Stejné (8) |
| 6 | 15 | Stejná | Stejná | Stejné (8) |
| 11 | 15 | Stejná | Stejná | Lepší (9) |

Tabulka 8.4: Porovnání výsledků při změně počtu koeficientů MFCC

Experimenty s počtem koeficientů nakonec neukázaly žádné příliš rozdílné výsledky. Je zřejmé, že rozdílu bychom dosáhli při větším rozptylu od původní hodnoty, to by však skoro jistě vedlo spíše ke zhoršení. Ne nadarmo je doporučena hodnota 13 jako optimální.

### 8.2.3 Změna frekvenčního rozsahu

V původních testech 8.1 je frekvenční rozsah nastaven na 133 - 6 855 Hz. Výsledky těchto testů jsou brány jako referenční.

| Číslo | Frekv. rozsah | Doba vykonávání | Rychlost konv. | Hodnocení |
|---|---|---|---|---|
| 1 | 16 - 6 855 Hz | Stejná | Stejná | Stejné (8) |
| 6 | 16 - 6 855 Hz | Stejná | Stejná | Stejné (8) |
| 11 | 16 - 6 855 Hz | Stejná | Stejná | Stejné (8) |
| 1 | 133 - 10 000 Hz | Stejná | Stejná | Horší (7) |
| 6 | 133 - 10 000 Hz | Stejná | Stejná | Stejné (8) |
| 11 | 133 - 10 000 Hz | Stejná | Nekonverguje | Horší (3) |

Tabulka 8.5: Porovnání výsledků při změně frekvenčního rozsahu MFCC

Z experimentů je zřejmé, že rozšířením frekvenčního rozsahu filtrů dochází k horším výsledkům. Rozšíření směrem dolů nemůže být proti původní hodnotě příliš velké, proto zde nejsou vidět rozdíly. Rozšíření do vyšších frekvencí však ukazuje zhoršení. To se projevuje zejména u zvuku s číslem 11, kde bylo dosaženo velmi špatných výsledků. Nezdá se tedy, že by rozšíření frekvenčního rozsahu mělo produkovat lepší řešení.

Frekvenční rozsah je úzce spjat s počtem filtrů, proto jsem mimo tento experiment vyzkoušel zvýšit počet filtrů, aby se tak větší rozsah rozprostřel mezi více filtrů. Lepších výsledků jsem ale nedosáhl.

## 8.3 Experimenty s prolínáním okénkovací funkce

V sekci 6.2 jsme rozebrali vliv koeficientu $\alpha$ na Hammingovu okénkovací funkci. Nyní se pokusíme s touto hodnotou experimentovat a budeme sledovat rozdíly proti referenčnímu řešení 8.1, které využívalo hodnotu $\alpha = 0.46$. Jelikož nemá smysl volit hodnotu vyšší než 0.5 (dochází k ignorování některých okrajových hodnot), zaměříme se především na hodnoty nižší než 0.46.

V experimentu nebudeme sledovat dobu vykonávání, jelikož změna koeficientu na ní nemůže mít vliv.

| **Číslo** | **Alpha** | **Rychlost konv.** | **Hodnocení** |
|---|---|---|---|
| 1 | 0.1 | Stejná | Stejné (8) |
| 6 | 0.1 | Stejná | Stejné (8) |
| 11 | 0.1 | Stejná | Stejná (8) |
| 1 | 0.25 | Stejná | Stejné (8) |
| 6 | 0.25 | Stejná | Stejné (8) |
| 11 | 0.25 | Stejná | Stejné (8) |

Tabulka 8.6: Porovnání výsledků při rozdílném překryvu Hammingovy okénkovací funkce

Výsledky experimentu ukázaly, že na hodnotě překryvu prakticky nezáleží. Vzhledem k počtu vzorků si však nedovolím toto tvrzení generalizovat. V našem případě to však říci lze.

## 8.4 Resyntéza reálných zvuků

Zkusil jsem také reprodukovat několik zvuků na jiných syntetizérech, než na kterých byly původně vytvořeny. Z těchto experimentů vyplynulo, že taková resyntéza je možná, ale obvykle je dosaženo horších výsledků. Velmi záleží na výběru syntetizéru a na tom, zda syntetizér vůbec dokáže zvuk kvalitně reprodukovat. Algoritmu je jedno s jakým syntetizérem pracuje, vždy se pouze snaží najít co nejlepší řešení.

Tento způsob resyntézy je vhodný zejména při prozkoumávání možností syntetizéru. Algoritmus je schopen rychle a s relativně velkou věrohodností odpovědět na otázku, zda je možné syntetizovat podobný zvuk na daném syntetizéru.

## 8.5 Velikost turnajové skupiny

Jelikož není zřejmé, jaká velikost turnajové skupiny je nejvhodnější pro řešení našeho problému, rozhodl jsem se s hodnotou experimentovat a sledovat rozdíly mezi různými nastaveními. Velikost skupin u referenčního řešení je 7. Při změně parametru velikosti skupiny si můžeme být jisti, že zůstane zachována doba běhu, proto není nutné ji porovnávat.

| **Číslo** | **Velikost skupiny** | **Rychlost konv.** | **Hodnocení** |
|---|---|---|---|
| 1 | 4 | Nekonverguje | Horší (4) |
| 6 | 4 | Stejná | Stejné (8) |
| 11 | 4 | Nekonverguje | Horší (4) |
| 1 | 10 | Stejná | Stejné (8) |
| 6 | 10 | Stejná | Stejné (8) |
| 11 | 10 | Nekonverguje | Horší (4) |

Tabulka 8.7: Porovnání výsledků při rozdílné velikosti turnajové skupiny

Z experimentů je zřejmé, že při nižší hodnotě turnajové skupiny (4) algoritmus k žádnému kvalitnímu řešení nemusí konvergovat. Obdobně to platí i při zvětšení skupiny na 10. Je tedy vidět, že selekční metoda je vysoce náchylná k velikosti skupiny. Příliš nízká hodnota propouští do dalších generací mnoho nekvalitních jedinců. Pokud je hodnota naopak příliš vysoká, algoritmus se může zaseknout na lokálním minimu, které se ani zdaleka neblíží dobrému řešení.

# Kapitola 9

# Závěr

Na závěr provedeme shrnutí a stručné vyhodnocení provedené práce a navrhneme doporučení a zaměření pro budoucí projekty v oblasti resyntézy zvuku.

## 9.1 Provedená práce

Cílem této diplomové práce bylo prozkoumat oblast syntézy a resyntézy zvuku, vytvořit programu, který by dokázal aproximovat zvuk na daném syntetizéru a následně provést testování.

Před začátkem celé práce bylo nutné se seznámit a porozumět základním principům a postupům syntézy zvuku. Tyto informace lze nalézt v **kapitole 2**. Zde jsme také provedli rozbor tří existujících prací, na které je později odkazováno.

V **kapitole 3** jsme rozebrali jednu z hlavních součástí této práce a tou je genetický algoritmus. Popsali jsme zde jeho fungování a možnosti a stanovili důsledky volby některých parametrů.

Dále jsme provedli rešerši metod pro porovnávání zvuků v **kapitole 4**. Zde jsme rozebrali detailně především MFCC, jelikož jsme tuto metodu využili pro porovnávání zvuků v této práci. Zaobírali jsme se ale i metodou Spektrálních norem a centroidů, kterou využil Johnson.

Následně jsme se v **kapitole 5** zaobírali využitými metodami GA (selekční metoda pomocí turnajů, rozšířené obdélníkové křížení a mutace) a nastavením parametrů genetického algoritmu. Především jsme však popsali, jak genetický algoritmus implementovat tak, aby se dal aplikovat na náš problém.

V **kapitole 6** jsme určili parametry pro MFCC. Zaobírali jsme se také jeho aplikací,

tedy okénkovacími funkcemi a jejich překryvy. K našemu účelu se jako nejvhodnější jevila Hammingova okénkovací funkce. Rozebrali jsme možnosti porovnávání MFCC. Nakonec jsme vybrali porovnávání pomocí *Euklidovské vzdálenosti*, ač se tato volba může potýkat s problémy nevyváženosti jednotlivých koeficientů MFCC. Nakonec jsme také rozebrali problém "odlišné syntézy zvuku za stejných podmínek"a "posunu zvuku v čase". První problém jsme částečně vyřešili pomocí vyčištění výstupu syntetizéru. Na druhý problém jsme aplikovali DTW, který by jej měl řešit úplně.

Program, který byl vytvořen jsme popsali v **kapitole 7**. Definovali jsme zde přesné vstupy a výstupy a zaobírali jsme se výběrem frameworku pro komunikaci s VST. Nakonec byla zvolena kombinace dvou frameworků. *JVST*, který je schopen zobrazit grafiku syntetizéru a *JVSTHost*, který slouží jako komunikační vrstva. Aby bylo zřejmé, jak program funguje, popsali jsme také komunikaci, která v něm probíhá mezi jednotlivými částmi.

Nakonec jsme v **kapitole 8** provedli kvalitativní testování na 16 rozdílných zvucích a 6 syntetizérech. Shrnuli jsme zde výsledky tohoto testování a také popsali další provedené experimenty a ostatní poznatky, které vyšly najevo během testování. **Výsledky ukázaly, že resyntéza je nejenom možná, ale zároveň i relativně spolehlivá**. Záleží však na délce a charakteru zvuku, komplexnosti syntetizéru a také na očekávání uživatele.

## 9.2 Návrhy do budoucna

**Porovnávání zvuku** MFCC a DTW se jako porovnávací algoritmus osvědčily, ale jsou zde jisté nedostatky. Algoritmus MFCC někdy hodnotí poslechové vzdálenější zvuky lepší hodnotou než ty, které jsou si blíže. Do budoucna doporučuji především analýzu vhodnosti zvolených metod na různé typy zvuků.

**Hlasitostní obálka a první koeficient MFCC** Vzhledem k výsledkům testů by bylo vhodné se v budoucích pracech zaměřit na první koeficient MFCC, který nese informaci o energii daného časového okna. Tato informace by mohla být vhodně využita při porovnávání zvuku tak, aby GA ve výsledku generoval kvalitnější řešení, které by hlasitostně více odpovídalo.

Možným řešením je vytvořit takový systém, který by prvnímu koeficientu přidělil menší váhu. Například by bylo možné nejprve optimalizovat parametry bez tohoto koeficientu, následně uzamknout veškeré parametry, které se netýkají hlasitostní obálky a nakonec zkusit optimalizovat zbylé parametry za použití prvního koeficientu. V tomto případě je však nutné znát kontext parametrů, tedy vědět, které parametry odpovídají hlasitostní obálce (pokud je to vůbec možné). Pokud by tato znalost poskytnuta nebyla,

uzamčení by proběhnout nemuselo a systém by se spoléhal pouze na to, že GA nebude příliš měnit parametry, které neodpovídají hlasitostní obálce, jelikož by teoreticky měly produkovat méně kvalitní jedince, kteří by neprošli selekcí.

**Nastavení koeficientů MFCC** V sekci 8.2 jsme provedli testování, které mělo odhaliv možné vlivy na výsledky při změně některých parametrů MFCC. Neprovedli jsme však komplexní analýzu, ve které by se uvažoval vliv jednotlivých parametrů na sebe navzájem. Proto by bylo v budoucnu možné provést kvantitativní testy, které by ohdalily, jaké hodnoty parametrů jsou vhodné na různé typy zvuků, jelikož je předpoklad, že pro různé zvuky se hodí různá nastavení.

**Nízkofrekvenční oscilátor** Testování ukázalo, že je obtížné automaticky resyntetizovat zvuk na syntetizéru s nízkofrekvenčním oscilátorem. Ten způsobuje, že zvuk při každé samostatné syntéze zní jinak, protože stav oscilátoru je se různí a je závislý na interních pochodech syntetizéru (resp. na času). Problém by bylo možné částečně vyřešit podobně, jako u hlasitostní obálky. Program by však musel vědět, co vše oscilátor ovlivňuje, zda zesilovač, filtr nebo výšku oscilátoru, popř. úplně něco jiného. Pokud by byl nízkofrekvenční oscilátor připojen například k zesilovači, mohl by program sledovat změnu hlasitosti v *sustain* části obálky zvuku a ty následně porovnávat. Jde však o velice komplexní analýzu, která nebude fungovat v případě nedostatku informací o syntetizéru.

**Resyntéza na odlišném syntetizéru** V budoucnu by bylo vhodné řádně otestovat, možnosti resyntézy cílového zvuku na jiném syntetizéru, než na jakém byl zvuk původně produkován. Zároveň by se tak definovaly vztahy mezi syntetizéry, resp. jejich skupinami, které by určovaly, kam až je možno zajít a jaký výsledek očekávat, což by pomohlo při prozkoumávání možností syntetizérů a zvuků obecně.

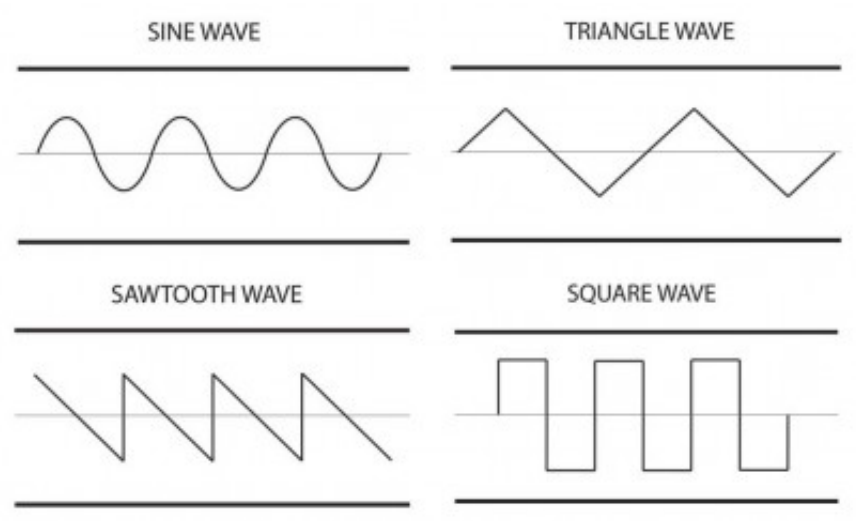
# Příloha A

# Zkratky

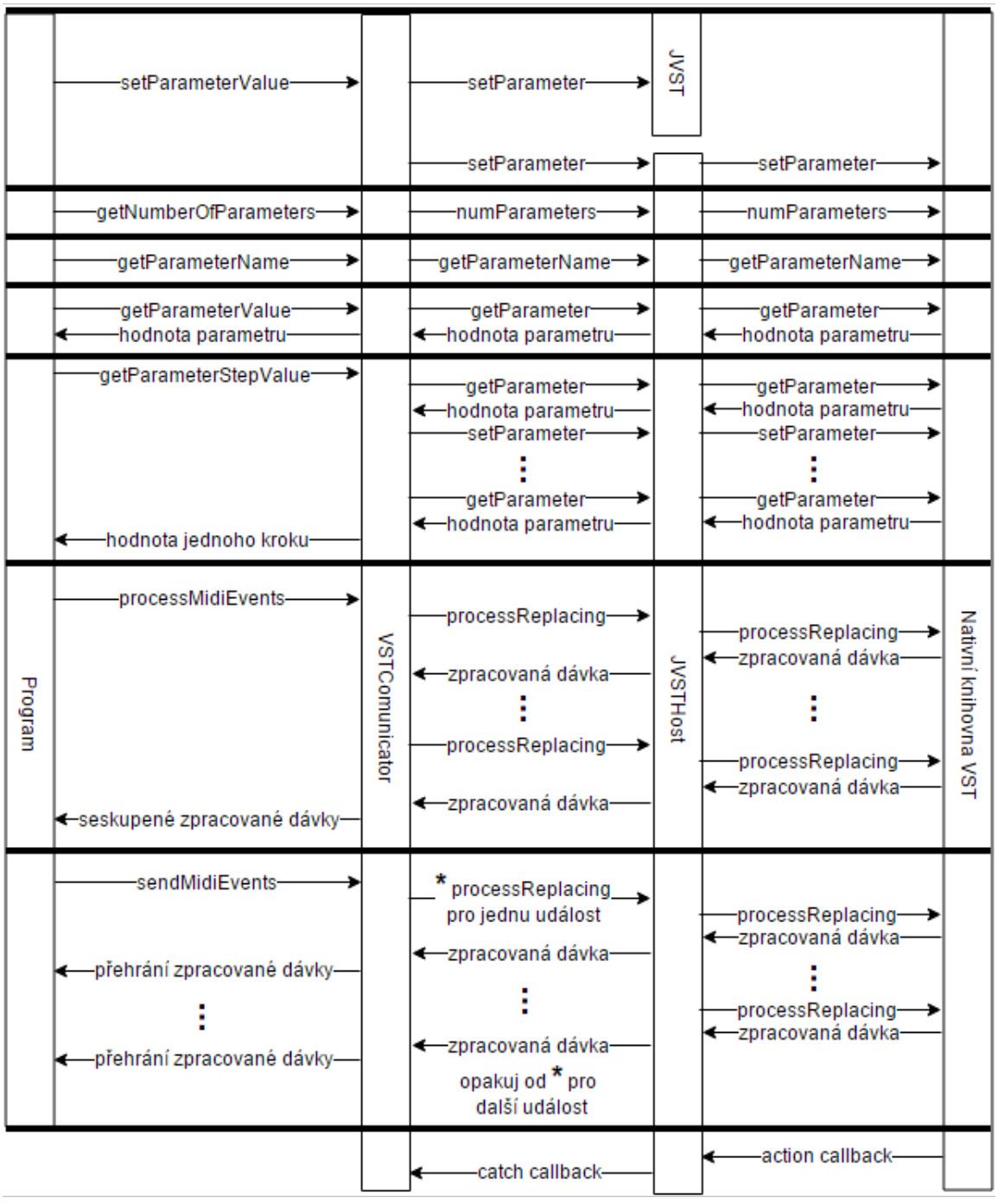
- VST - Virtual Studio Technology
- BPM - Beats per Minute
- PPQ - Pulses per Quarter
- FM - Frequency Modulation
- AM - Amplitude Modulation
- DFM - Double Frequency Modulation
- GA - Genetic Algorithm
- SA - Simulated Annealing
- MFCC - Mel-frequency Cepstral Coefficients
- FT - Fourier Transform
- DFT - Discrete Fourier Transform
- DCT - Discrete Cosine Transformation
- IFT - Inverse Fourier Transform
- EMD - Earth Mover's Distance
- DTW - Dynamic Time Wrapping

# Příloha B

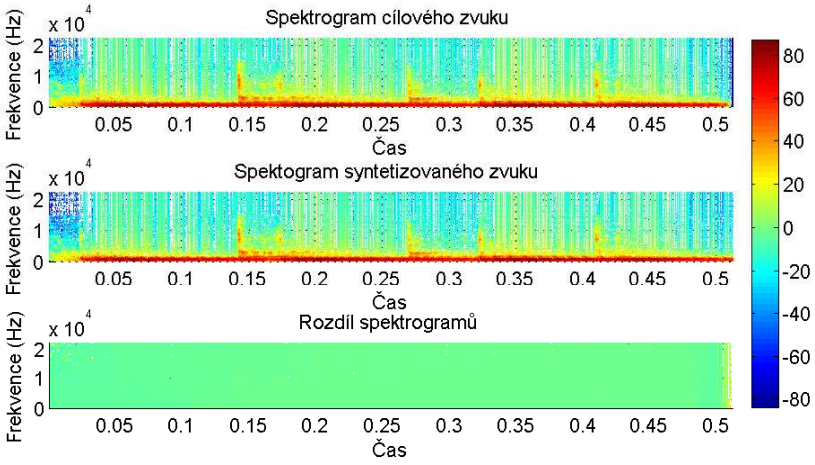
# Obrázky




Obrázek B.1: Základní typy vln oscilátoru [37]




Obrázek B.2: Diagram komunikace programu, komunikátoru a knihoven




Obrázek B.3: Porovnání spektrogramů cílového a syntetizovaného zvuku s normalizovanou kvalitou 0.014 (10.)

# Příloha C

# Manuál k programu

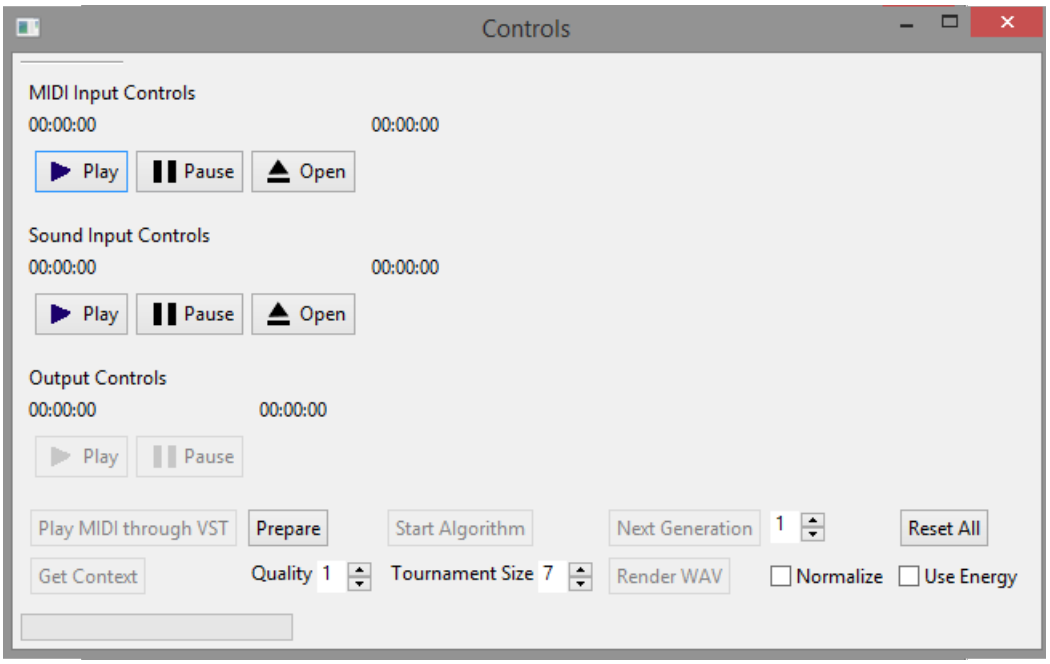
Při tvorbě programu nebyl příliš velký důraz kladen na vzhled. Zaměřil jsem se pouze na to nejpodstatnější a to bylo umožnit uživateli ovládat program, vidět výsledky a manipulovat s nimi. Základem programu jsou čtyři okna.

První okno C.1 zobrazuje původní grafické rozhraní syntetizéru, kde je možno manipulovat s parametry původním způsobem.


Obrázek C.1: Ukázka vykresleného původního rozhraní syntetizéru

Další okno C.2 kontroluje většinu funkcí programu. K dispozici jsou tlačítka pro načtení **MIDI** a **cílového** zvuku. Tyto soubory je také možno nechat přehrát. Dále okno obsahuje tlačítko **Start Algorithm**, které spustí první fázi algoritmu, tedy vygenerování první generace populace a jeho ohodnocení. Tlačítko **Next Generation** umožňuje provést další krok algoritmu a to je selekce, křížení a mutace. Jde tedy o kompletní vygenerování nové generace. Počet generací, které budou generovány je možno ovládat pomocí **číselníku** vedle tlačítka **Next Generation**, které slouží pro generování další generace. Talčítko **Reset All** vrátí program do původního stavu a smaže

všechny generace, které byly vygenerovány. Číselník **Quality** určuje kolikát proběhne generování zvuku pro jedno nastavení (viz 6.4). Číselník **Tournament Size** určuje velikost turnajové skupiny pro genetický algoritmus. K dispozici je také **stavová lišta**, která zobrazuje, jak velká část z další generace je připravena. Dále je k dispozici tlačítko **Play MIDI through VST**, které syntetizuje zvuk dle aktuálně nastavených parametrů pomocí aktuálně načteného souboru MIDI. Nakonec je také možné nechat uložit zvuk syntetizovaný podle aktuálně nastaveného syntetizéru do formátu WAV pomocí tlačítka **Render WAV**. Nakonec je k dispozici přepínač **Normalize**, který umožní normalizaci všech zvuků a **Use Energy**, který zaručí využití prvního koeficientu MFCC v porovnávací funkci.

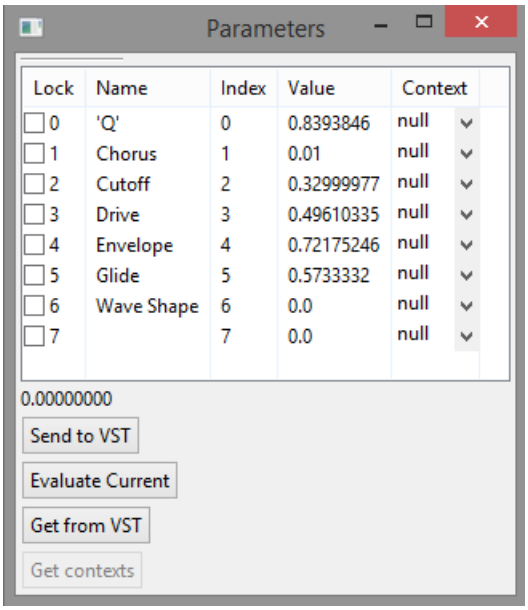



Obrázek C.2: Ovládací okno programu

Důležitým oknem je okno s parametry C.3. V tomto okně se zobrazují parametry syntetizéru v tabulce v surové podobě. První sloupec tabulky obsahuje **zaškrtávátka**, pomocí kterých je možné **uzamknout parametr** tak, že s ním nebude manipulováno v důsledku genetického algoritmu. Takto je možno pomoci algoritmu s rychlejším nalezením řešení, případně algoritmus určitým způsobem omezit. Nutno podotknout, že hodnota se uzamkne na aktuálně nastavené hodnotě v syntetizéru. Ve druhém sloupci jsou zobrazeny **názvy parametrů**, dále pak **index parametru** ve třetím sloupci a **hodnota parametru** ve sloupci čtvrtém. Hodnota parametru je závislá od předchozích akcí, jelikož okno může zobrazovat jak hodnoty parametrů syntetizéru, tak hodnoty jednotlivých jedinců z populace. Pátý sloupec sloužil pro testování.

Tlačítko **Send to VST** odešle hodnoty parametrů do syntetizéru. Takto je možno nastavit syntetizéru parametry některého vybraného jedince z populace (viz následující okno programu). Tlačítko **Evaluate Current** slouží pro testovací účely jako rychlé vy-

hodnocení aktuálního nastavení. Poslední tlačítko **Get from VST** slouží pro nastavení aktuálních hodnot ze syntetizéru.




Obrázek C.3: Okno s parametry s ovládacími prvky pro ně

Poslední okno zobrazuje jedince z populace v tabulce rozdělené **po sloupcích do generací a seřazeni vzestupně podle své kvality**. Zároveň se v posledních řádcích tabulky vyskytují řádky s **celkovým součtem kvality jedinců**, **minimum**, **maximum** a **průměr** z celé jedné generace. **Kliknutím na kvalitu jedince lze nechat zobrazit hodnoty jeho genů v okně s parametry** (následně je možné tyto hodnoty přenést pomocí tlačítka **Send to VST** do syntetizéru). Podobně je možné kliknout na hodnotu maxima či minima, které jsou propojené s jedinci. **Dvojklikem na kvalitu jedince je také možné změnit hodnotu kvality**. Tato změna bude brána vpotaz při vytváření další generace (resp. při selekci). Poslední funkcí je možnost kopírování výsledků doschránky pomocí zkratky CTRL+C, vložení je pak možné například do excelu.

Typicky se program ovládá následujícím způsobem:

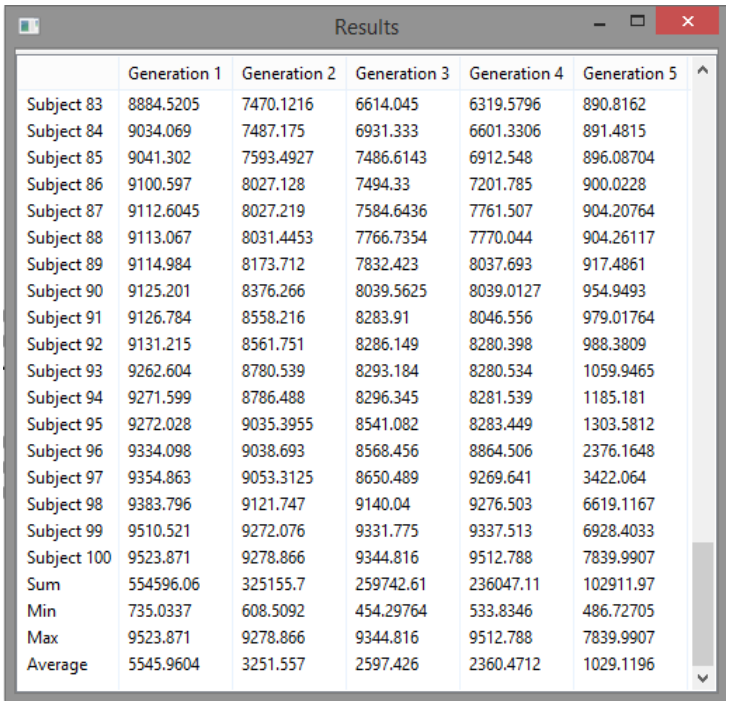
1. Nechá se načíst MIDI soubor.
2. Nechá se načíst soubor s cílovým zvukem.
3. Nastaví se hodnota "Quality"a stiskne se tlačítko "Start Algorithm".
4. Nastaví se hodnota pro počet generací a stiskně se tlačítko "Next Generation".
5. Jedinci se zobrazují postupně do tabulky s výsledky.

6. Vyhledá se jedinec s požadovanou kvalitou (nejlepší se většinou vyskytují v pozdějších generacích na prvních pozicích).
7. Klikne se na kvalitu
8. V okně s parametry se klikne na tlačítko "Send to VST"a pomocí tlačítka "Play MIDI through VST"v hlavním ovládacím okne se nechá zvuk přehrát. Zároveň je také možno nechat přehrát cílový zvuk a poslechově tyto dva zvuky porovnat.
9. Takto se pokračuje, dokud se nenajde uspokojivý zvuk.
10. Nakonec je možné nechat ručně upravit parametry pomocí okna se syntetizérem.

Program je možné spustit pomocí souboru *Resynthesis.jar*, kterému je možno předat parametry uvedené v tabulce C.1.

| **Klíč** | **Hodnota** |
|---|---|
| vstfile | Cesta ke knihovně s VST |
| soundfile | Cesta k cílovému zvuku (*.wav) |
| midifile | Cesta k MIDI souboru (*.mid) |

Tabulka C.1: Parametry programu




| | Generation 1 | Generation 2 | Generation 3 | Generation 4 | Generation 5 |
|---|---|---|---|---|---|
| Subject 83 | 8884.5205 | 7470.1216 | 6614.045 | 6319.5796 | 890.8162 |
| Subject 84 | 9034.069 | 7487.175 | 6931.333 | 6601.3306 | 891.4815 |
| Subject 85 | 9041.302 | 7593.4927 | 7486.6143 | 6912.548 | 896.08704 |
| Subject 86 | 9100.597 | 8027.128 | 7494.33 | 7201.785 | 900.0228 |
| Subject 87 | 9112.6045 | 8027.219 | 7584.6436 | 7761.507 | 904.20764 |
| Subject 88 | 9113.067 | 8031.4453 | 7766.7354 | 7770.044 | 904.26117 |
| Subject 89 | 9114.984 | 8173.712 | 7832.423 | 8037.693 | 917.4861 |
| Subject 90 | 9125.201 | 8376.266 | 8039.5625 | 8039.0127 | 954.9493 |
| Subject 91 | 9126.784 | 8558.216 | 8283.91 | 8046.556 | 979.01764 |
| Subject 92 | 9131.215 | 8561.751 | 8286.149 | 8280.398 | 988.3809 |
| Subject 93 | 9262.604 | 8780.539 | 8293.184 | 8280.534 | 1059.9465 |
| Subject 94 | 9271.599 | 8786.488 | 8296.345 | 8281.539 | 1185.181 |
| Subject 95 | 9272.028 | 9035.3955 | 8541.082 | 8283.449 | 1303.5812 |
| Subject 96 | 9334.098 | 9038.693 | 8568.456 | 8864.506 | 2376.1648 |
| Subject 97 | 9354.863 | 9053.3125 | 8650.489 | 9269.641 | 3422.064 |
| Subject 98 | 9383.796 | 9121.747 | 9140.04 | 9276.503 | 6619.1167 |
| Subject 99 | 9510.521 | 9272.076 | 9331.775 | 9337.513 | 6928.4033 |
| Subject 100 | 9523.871 | 9278.866 | 9344.816 | 9512.788 | 7839.9907 |
| Sum | 554596.06 | 325155.7 | 259742.61 | 236047.11 | 102911.97 |
| Min | 735.0337 | 608.5092 | 454.29764 | 533.8346 | 486.72705 |
| Max | 9523.871 | 9278.866 | 9344.816 | 9512.788 | 7839.9907 |
| Average | 5545.9604 | 3251.557 | 2597.426 | 2360.4712 | 1029.1196 |

Obrázek C.4: Okno s vysledky

# Příloha D

# Obsah přiloženého DVD

| **Cesta** | **Popis** |
|---|---|
| bin/Resynthesis.jar | Program pro resyntézu |
| src | Zdrojové kódy programu |
| lib | Využité knihovny |
| testing/*.xlsx | Tabulky s grafy |
| testing/sounds | Soubory k hlavním testům (vstupní cílové zvuky, VST, MID a příslušné výsledky) |
| testing/other | Výsledky ostatních experimentů |

# Literatura


[1] Jihočeská univerzita, *Vlnění a zvuk*. [online] Dostupné z: `http://mvt.ic.cz/dva/tef/tef-04.pdf`.

[2] REICHL, J.; VŠETIČKA, M., *Encyklopedie Fyziky*. [online] 2006-2015. Dostupné z: `http://fyzika.jreichl.com`.

[3] Songstuff Site Crew, *MIDI Message Format*. [online]. Dostupné z: `http://recording.songstuff.com/article/midi_message_format/`

[4] SIEVERS, Beau, *A Young Person's Guide to the Principles of Music Synthesis*. [online]. Dostupné z: `http://beausievers.com/synth/synthbasics/`

[5] JOHNSON, A., *Sound Resynthesis with a Genetic Algorithm*. [online] 2011. Dostupné z: `http://www.doc.ic.ac.uk/teaching/distinguished-projects/2011/a.johnson.pdf`

[6] YONG, S., *Automatic FM Synthesis Based On Genetic Algorithm*. [online] 2007.

[7] LAI, Yuyo., *Automated Optimization of Parameter for FM Sound Synthesis with Genetic Algorithms*. [online] 2006. Dostupné z: `http://forum.dmc.ntnu.edu.tw/~wocmat2006/pdf/2-3.pdf`

[8] TAN, B. T. G.; LIM, S. M. *Automated Parameter Optimization of Double Frequency Modulation Synthesis Using the Genetic Annealing Algorithm*. [online] 1996. Dostupné z: `http://www.physics.nus.edu.sg/~phytanb/automatedparameter.pdf`

[9] MCDERMOTT, James; GRIFFITH, Niall JL; O'NEILL *Michael. Evolutionary computation applied to sound synthesis*. [online] 2008. Dostupné z: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.330.8694&rep=rep1&type=pdf`

[10] RIIONHEIMO, J.; VÄLIMÄKI, V., *Parameter Estimation of a Plucked String Synthesis Model Using a Genetic Algorithm with Perceptual Fitness Calculation*. [online] 2002. Dostupné z: `http://asp.eurasipjournals.com/content/pdf/1687-6180-2003-758284.pdf`

[11] JEBARI, Khalid; MADIAFI, Mohammed, *Selection Methods for Genetic Algorithms*. [online] 2013. Dostupné z: http://ijes.info/3/4/42543401.pdf

[12] POHLHEIM, Hartmut, *GEATbx: Algorithms*. [online] 2006. Dostupné z: http://www.geatbx.com/

[13] MAGALHÃES-MENDES, Jorge, *A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem* . [online] 2013. Dostupné z: http://www.wseas.org/multimedia/journals/computers/2013/5705-156.pdf

[14] YOON, Yourim; KIM, Yong-Hyuk, *The Roles of Crossover and Mutation in Real-Coded Genetic Algorithms*. [online] 2012. Dostupné z: http://cdn.intechopen.com/pdfs-wm/30229.pdf

[15] BURRUS, Sidney C., *Fast Fourier Transforms. Rice University e-Book*. [online] 2010. Dostupné z: http://cnx.org/content/col10550/latest/

[16] DOSTÁL, F, *„Porovnání audio vzorků*. [online] 2014. Dostupné z: http://digilib.k.utb.cz/handle/10563/29906

[17] LOUGHRAN, Róisín; WALKER, Jacqueline; O'NEILL, Michael a O'FARRELL, Marion. *The Use of Mel-frequency Cepstral Coefficients in Musical Instrument Identification.*[online] 2013. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.331.2898&rep=rep1&type=pdf

[18] LOGAN, Beth. *Mel Frequency Cepstral Coefficients for Music Modeling*. [online] 2000. Dostupné z: http://ismir2000.ismir.net/papers/logan_paper.pdf

[19] TYAGI, V.; WELLEKENS, C. *On desensitizing the Mel-Cepstrum to spurious spectral components for Robust Speech Recognition, in Acoustics, Speech, and Signal Processing*. [online] 2005. IEEE International Conference on, vol. 1: 529–532. Dostupné z: http://www.researchgate.net/profile/Christian_Wellekens/publication/4136867_On_desensitizing_the_Mel-Cepstrum_to_spurious_spectral_components_for_Robust_Speech_Recognition/links/00b7d51500034affb9000000.pdf

[20] MUBARAK, Omer Mohsin; AMBIKAIRAJAH, Eliathamby a EPPS, Julien. *Analysis of an MFCC-based Audio Indexing System For Efficient Coding Of Multimedia Sources*. [online] 2005. Dostupné z:http://www.researchgate.net/publication/4216798_Analysis_of_an_MFCC-based_audio_indexing_system_for_efficient_coding_of_multimedia_sources

[21] PYE, David. *Content-Based Methods for the Management of Digital Music*. [online] 2000. Proceedings. 2000 IEEE International Conference on. Vol. 6. Dostupné z: http://www1.icsi.berkeley.edu/~dpwe/research/etc/icassp2000/pdf/4001_611.PDF

[22] JANG, Roger, *Audio Signal Processing and Recognition*. [online] Dostupné z: http://mirlab.org/jang/books/audiosignalprocessing/

[23] KELLY, Amelia C. and GOBL, Christer, *The effects of windowing on the calculation of MFCCs for different types of speech sounds*. [online] 2011. Dostupné z: https://www.academia.edu/1319601/The_Effects_of_Windowing_on_the_Calculation_of_MFCCs_for_Different_Types_of_Speech_Sounds

[24] jVSTwRapper, "Soubor:cubase_sx_win.jpg"(http://jvstwrapper.sourceforge.net/).

[25] RAZALI, Noraini M.; GERAGHTY, John, *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. [online] 2011. Dostupné z: http://www.iaeng.org/publication/WCE2011/WCE2011_pp1134-1139.pdf

[26] VIBHA, Tiwari. *MFCC and its applications in speaker recognition* [online] 2010. International Journal on Emerging Technologies 1.1 (2010): 19-22 Dostupné z: http://www.researchtrend.net/ijet/4_Vibha.pdf

[27] KOENIG, Andreas C., *A Study of Mutation Methods for Evolutionary Algorithms*. [online] 2002. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.2465&rep=rep1&type=pdf

[28] KLAUTAU, Aldebaro. *The MFCC*. [online] 2005. Dostupné z:http://www.cic.unb.br/~lamar/te073/Aulas/mfcc.pdf

[29] MUDA, Lindasalwa; BEGAM, Mumtaj, *Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques*. 2010. Dostupné z: http://arxiv.org/ftp/arxiv/papers/1003/1003.4083.pdf

[30] MCENNIS, Daniel *Java knihovna jAudio v 1.1.1.* 2015

[31] Universität Würzburg: Fakultät für Physik und Astronomie. *Understanding FFT Windows*. [online] 2003. Dostupné z: http://www.physik.uni-wuerzburg.de/fileadmin/11016800/Modulbeschreibungen/CMT/ANO14.pdf

[32] ZHU, Qifeng, ABEER Alwan. *On the use of variable frame rate analysis in speech recognition*. [online] 2000. Dostupné z: http://www.seas.ucla.edu/spapl/paper/zhu_icassp00.pdf

[33] KLABBERS, Esther, et al. *Speech synthesis development made easy: the bonn open synthesis system*. [online] 2001. Dostupné z: http://www.mirlab.org/conference_papers/International_Conference/Eurospeech%202001/papers/page521.pdf

[34] LOGAN, Beth; ARIEL Salomon. *A Music Similarity Function Based on Signal Analysis*. [online] 2001. Dostupné z: http://apotheca.hpl.hp.com/ftp/gatekeeper/pub/compaq/CRL/publications/logan/icme2001_logan.pdf

[35] SIGURDSSON, Sigurdur; KAARE, Brandt Petersen; TUE, Lehn-Schiøler. *Mel frequency cepstral coefficients: An evaluation of robustness of mp3 encoded music*. [online] 2006. Dostupné z: http://orbit.dtu.dk/fedora/objects/orbit:65367/datastreams/file_5801103/content

[36] ROEVA, Olympia; FIDANOVA, Stefka *Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling*. [online] 2013. Dostupné z : http://annals-csis.org/proceedings/2013/pliks/167.pdf

[37] Basic wave forms, "Soubor:wp-content/uploads/2012/06/basicwaveforms-405x245.jpg"(http://www.cameronmizell.com/).