Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Stosowana (IST)**
Specjalność: **Zastosowania Specjalistycznych Technologii Informatycznych**

PRACA DYPLOMOWA MAGISTERSKA

Opracowanie algorytmu generacji grafu DSP do rozwiązania problemu syntezy dźwięku

Automated generation of signal processing graphs for sound synthesis

Mateusz Bączek

Opiekun pracy
dr inż. Maciej Hojda

Słowa kluczowe: synteza, dźwięk, graf, optymalizacja

WROCŁAW, 2023

Streszczenie

Praca prezentuje metodę automatycznej konstrukcji grafu przetwarzania sygnałów dźwiękowych, które wykonują syntezę zadanego przez użytkownika dźwięku. Wytworzony w ramach pracy algorytm może zostać wykorzystany jako narzędzie w pracy inżynierów dźwięku, podczas tworzenia nowych instrumentów elektronicznych lub efektów specjalnych. W przeciwieństwie do technik wykorzystujących sieci neuronowe jako narzędzia syntezy, wynikiem działania algorytmu wytworzonego w ramach pracy jest zrozumiały dla człowieka graf przetwarzania sygnałów, przypominający konwencjonalne konfiguracje syntezatorów dźwięku wykorzystywane w programach do pracy nad dźwiękiem.

Słowa kluczowe: synteza, dźwięk, graf, optymalizacja

Abstract

Thesis explores a technique for automated design of sound synthesizers, which can be conceptualized as signal processing graphs. In contrast to approaches that rely on neural networks for sound synthesis, the thesis introduces an algorithm for dynamically generating signal processing graphs that are both comprehensive and modifiable by end-users. By enabling users to easily understand and modify the generated graphs, the algorithm offers a versatile and user-friendly solution for composers and sound enginners, allowing them to create novel musical instruments or audio effects by cooperating with a synthesizer-designing algorithm.

Keywords: synthesis, sound, audio, graph, optimisation

Spis treści

1. Wstęp 11
1.1. Cel pracy 13
1.1.1. Generowanie grafu przetwarzania sygnałów 13
1.1.2. Funkcja celu oceniająca podobieństwo barwy dźwieku 14
1.1.3. Problem optymalizacyjny 14
1.2. Zakres pracy, plan badań 14
1.2.1. Metody generowania grafu przetwarzania sygnałów oraz późniejsza modyfikacja grafu 14
1.2.2. Dobór funkcji błędu: różnica między wygenerowanym a docelowym sygnałem dźwiękowym 15
1.3. Struktura i zawartość pracy 15
2. Definicja problemu 17
2.1. Budowa grafu 18
2.1.1. Struktura grafu 18
2.1.2. Przypisanie parametrów do „wolnych wejść” 18
2.2. Funkcja celu 19
2.2.1. Wyliczanie współczynników MFCC [24] 19
2.3. Ograniczenia 20
2.4. Problem optymalizacji 20
3. Funkcja celu – porównanie barwy dźwięku 21
3.1. Porównanie barwy dźwięku w literaturze 21
3.1.1. Systematyzacja metod z literatury 22
3.1.2. Wybór funkcji celu do przetestowania 22
3.2. Proces testowania funkcji celu 22
3.3. Przekrój wartości funkcji celu dla prostego problemu syntezy typu FM 23
3.3.1. Analiza wyników 23
3.4. Optymalizacja parametrów dla predefiniowanych grafów syntezy FM oraz *analog modeling* 24
3.4.1. Synteza FM 24
3.4.2. Synteza *analog modeling* 24
3.4.3. Plan testów 25
3.4.4. Wyniki testów 25
3.4.5. Wybór funkcji celu na podstawie wyników 26
4. Algorytm rozwiązania 27
4.1. Wybór źródeł sygnału 28
4.1.1. Synteza FM 28
4.1.2. Synteza *analog modeling* 29
4.2. Wybór filtrów 30
4.3. Wybór efektów 30

5. Graf przetwarzania sygnałów 32
5.1. Podstawy syntezy dźwięku w syntezatorach modułowych 32
5.2. Wymagania 33
5.2.1. Węzły DSP 33
5.2.2. Połączenia między węzłami – modulacja parametrów węzłów 35
5.2.3. Graf przetwarzania sygnałów 36
5.2.4. Automatyzacja pracy ze środowiskiem eksperymentowym za pośrednictwem języka `Python` 36
5.3. Opis zaimplementowanego środowiska eksperymentowego 36
5.3.1. Przykłady użycia 36
5.3.2. Detale techniczne 37
6. Wyniki badań 39
6.0.1. Dźwięk fletu 39
6.0.2. Sampel z syntezatora *OP-1* 40
6.0.3. Transjent 40
7. Analiza wyników, możliwe drogi dalszych badań 41
7.1. Analiza wyników 41
7.2. Możliwe drogi dalszego rozwoju 41
Literatura 42
A. Instrukcja wdrożeniowa 45
B. Opis załączonej płyty CD/DVD 46

Spis rysunków

1.1. Zapis nutowy utworu *Opus One*, wygenerowany przez komputer *Lamus*. 11
1.2. Przykładowy spektrogram wygenerowany przez algorytm *Stable Riffusion* dla danych wejściowych `funk bassline with a jazzy saxophone solo`. 12
1.3. Syntezator *Mother 32* firmy *Moog*, po prawej stronie widoczny jest *patch bay* z podłączonymi przewodami, które nadpisują konfigurację połączeń między układami generującymi i przetwarzającymi sygnał dźwiękowy. 13
1.4. Zbiór parametrów konfigurujących syntezator dźwięku *Wavetable* w programie *Ableton* 13
1.5. Diagram blokowy pojedynczego głosu w syntezatorze *Minilogue xd* firmy *Korg* [28]. 14
2.1. Przykładowy węzeł w grafie, generujący sygnał sinusoidalny z możliwością modulacji. 17
2.2. Przykładowy graf DSP. Wolne wejścia, które nie są modulowane przez źródła sygnału w grafie są optymalizowanymi parametrami. 18
2.3. Schemat algorytmu obliczania współczynników MFCC, zaczerpnięty z [24]. Praca wykorzystuje gotową implementację algorytmu obliczającego współczynniki MFCC z pakietu `librosa` [33]. 19
3.1. Przykład trzech próbek dźwięku, które dla słuchacza brzmią identycznie, mimo znacznych różnic w kształcie fali. Źródło obrazka: [17]. 21
3.2. Prosty graf syntezy FM, zawierający jeden oscylator służący za sygnał nośny i jeden oscylator służący za sygnał modulujący. 22
3.3. Zmiany w wartościach testowanych funkcji celu podczas przesuwania różnych parametrów syntezy dźwięku. Kształt pierwszego wykresu wynika z zastosowania kwantyzacji dostępnych częstotliwości modulacji, aby wykluczyć nieharmoniczne stosunki częstotliwości modulacji i nośnej. Tego rodzaju praktyka jest wykorzystywana w syntezatorach FM [16], ponieważ ułatwia dostosowywanie parametrów syntezy. 23
3.4. Spektrogram, kształt fali oraz wizualizacja MFCC dla próbki dźwięku, którą ma imitować graf syntezy FM podczas testów różnych funkcji celu. 24
3.5. Graf wykonujący syntezę typu *analog modeling*, wykorzystany do testów funkcji celu. 24
3.6. Spektrogram, kształt fali oraz wizualizacja MFCC dla próbki dźwięku, którą ma imitować graf syntezy *analog_modeling* podczas testów różnych funkcji celu. . . . 25
3.7. Wykresy zmian funkcji celu podczas optymalizacji dla grafu syntezy FM. 25
3.8. Spektrogram dźwięku docelowego oraz dźwięków uzyskanych w procesie optymalizacji parametrów grafu FM. Czerwoną strzałką oznaczono składową harmoniczną (słabo widoczną na spektrogramie), która została poprawnie odtworzona przez algorytm optymalizacji. 25
3.9. Wykresy zmian funkcji celu podczas optymalizacji dla grafu syntezy *analog modeling*. **TODO: pełny wykres**. 26

3.10. Spektrogram dźwięku docelowego oraz dźwięków uzyskanych w procesie optymalizacji parametrów grafu *analog modeling*. **TODO: obrazek**. 26
4.1. Diagram algorytmu rozwiązania zaimplementowanego w ramach pracy. Algorytm oceny może wykorzystywać różne funkcje celu, finalnie zastosowano MFCC oraz *dynamic time wrapping*, proces wyboru funkcji celu opisuje rozdział 3. 27
4.2. Sekcje przetwarzania sygnałów oraz przykładowe węzły przetwarzania sygnałów, które są w nich powszechnie wykorzystywane. 27
4.3. Przykład wygenerowanej struktury grafu, oznaczono segmenty z diagramu 4.2. . . 28
4.4. Graf wykorzystujący gen `FM1`. 28
4.5. Graf wykorzystujący gen `FM2`. 29
4.6. Graf wykorzystujący gen `AN1`. 29
4.7. Graf wykorzystujący gen `AN2`. 30
4.8. Graf wykorzystujący gen `AN3`. 30
4.9. Przykładowy łańcuch efektów w grafie. 31
5.1. Przykładowy układ modułów w standardzie *Eurorack* [1]. W prawym dolnym rogu widoczne połączenia modulujące między modułami. 32
5.2. Przykładowy układ węzłów DSP w zaimplementowanym środowisku eksperymentowym. Układ wykonuje syntezę subtraktywną z modulowaną wartością częstotliwości granicznej filtru niskoprzepustowego oraz dodaje efekt pogłosu (*reverb*) [15]. 33
5.3. Węzeł DSP w zaimplementowanym środowisku eksperymentowym, generujący falę sinusoidalną z możliwością modulacji fazy. 34
5.4. Moduł syntezy *Mutable Instruments Elements*, umożliwiający ręczne ustawianie **oraz** modulację parametrów. Moduł wykonuję syntezę typu *physical modeling* [22]. 34
5.5. Przykładowa modulacja parametru `input_modulation` za pomocą sygnału sinusoidalnego, charakterystyczna dla syntezy typu FM [29]. 35
5.6. Spektrogram oraz wykres sygnału wygenerowanego za pomocą układ z rysunku 5.2.2. Widoczne dodatkowe składowe harmoniczne wpływające na barwę dźwięku. 35
5.7. Spektrogram oraz wykres sygnału wygenerowanego przez układ z rysunku 5.2.2 **po usunięciu** połączenia modulującego fazę oscylatora `#2`. Widoczna tylko jedna składowa harmoniczna: częstotliwość podstawowa. 35
5.8. Wynik wykonania kodu przedstawionego w listingu 5.2 w środowisku *Jupyter Notebook*, wizualizacja utworzonego grafu. 37
6.1. Spektrum fouriera i współczynniki MFCC dla dźwięku `flute.wav` wykorzystywanego do eksperymentów w [30]. 39
6.2. Spektrum fouriera i współczynniki MFCC dla dźwięku `op1_1.wav` wykorzystywanego do eksperymentów w [30]. 40
6.3. Spektrum fouriera i współczynniki MFCC dla dźwięku `transient.wav` wykorzystywanego do eksperymentów w [30]. 40

Spis tabel

Spis listingów

5.1. Implementacja węzła SineOscillator. 34
5.2. Utworzenie prostego grafu generującego sygnał sinusoidalny. 36
5.3. Typ danych zwracanych przez środowisko eksperymentalne. 37

Skróty

DAW (ang. *Digital Audio Workstation*) – oprogramowanie dostępne na komputery osobiste, służące do komponowania utworów muzycznych.

STFT (ang. *Short-time Fourier Transform*) – wariant transformaty Fouriera, wykonujący transformatę na ruchomym okno przesuwające się wzdłuż analizowanego sygnału. **STFT** pozwala na zwiększenie dokładności transformaty dla sygnałów o dużej zmienności w czasie. W kontekście syntezy audio, **SFTP** zwiększa dokładność z jaką rejestrowane są transjenty, czyli dynamiczne zmiany charakterystyki barwy dźwięku w czasie.

CV (ang. *Control Voltage*) – Sygnał sterujący parametrami syntezy dźwięku, standardowo wykorzystywanych w syntezatorach modułowych (przykładowo w standardzie *EuroRack*). Sygnał **CV** wykorzystuje się do przekazywania sygnałów kontrolnych między modułami.

VCO (ang. *Voltage Controlled Oscillator*) – komponent elektroniczny generujący sygnał dźwiekowy. Parametry generowanego sygnału sterowane są za pomocą napięcia kontrolnego (*CV*).

VCF (ang. *Voltage Controlled Filter*) – komponent elektroniczny wykonujący filtrację dźwięku w domenie częstotliwości. Parametry filtru sterowane są za pomocą napięcia kontrolnego (*CV*).

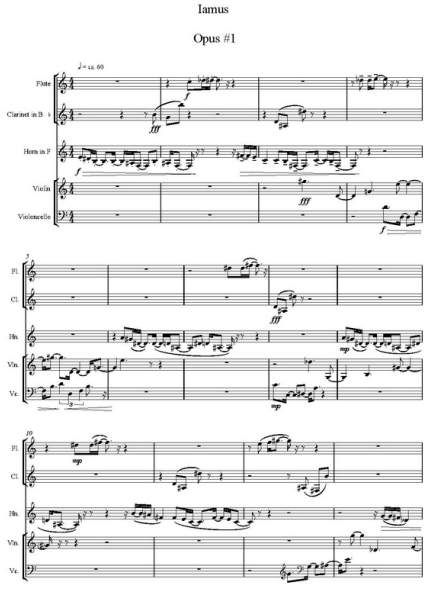
Rozdział 1

Wstęp

Rozpowszechnione algorytmy sztucznej inteligencji wspomagające pracę inżynierów dźwięku i kompozytorów można podzielić na trzy główne kategorie [14] :

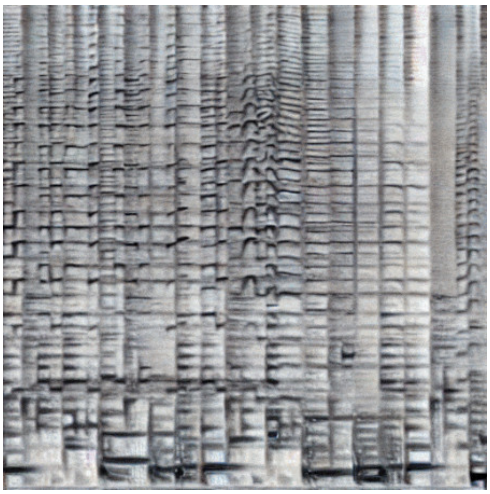
1. algorytmy generujące symboliczny zapis muzyki (nuty lub dane MIDI) (1.1), [40],
2. algorytmy generujące gotowy plik audio (1.2).
3. algorytmy symulujące brzmienie instrumentów muzycznych za pomocą sieci neuronowych [18].

Pierwsza grupa algorytmów znana jest już od lat 80, gdyż zagadnienie generowania zapisu symbolicznego wymaga mniej mocy obliczeniowej niż wytworzenie pełnego pliku audio. Powszechnie wykorzystywana jest w nich teoria muzyki, pozwalająca określić matematyczne relacje występujące w rytmach, melodiach i progresjach akordów. Wiedza dotyczącą teorii muzyki pozwala na wyznaczenie możliwej przestrzeni stanów, w której generowana jest kompozycja, natomiast modele matematyczne takie jak łańcuchy Markowa służą za mechanizmy decyzyjne, które „nawigują” w przestrzeni stanów.



Rys. 1.1: Zapis nutowy utworu *Opus One*, wygenerowany przez komputer *Lamus*.

Druga grupa algorytmów, generująca pliki audio, bazuje na klasie algorytmów wywodzących się ze *Stable Diffusion* [34]. Modele generujące pliki audio zgodne z opisem tekstowym (przykładowo „smutny jazz" bądź „muzyka taneczna w stylu Depeche Mode") szkolone są w taki sam sposób jak algorytmy *stable diffusion*, dane treningowe składają się z obrazów spektrogramów [20]. Po trenowaniu, model jest wstanie wygenerować spektrogram zawierający utwór muzyczny zgodny z promptem użytkownika (1.2). Wygenerowany przez model spektrogram jest konwertowany do sygnału dźwiękowego za pomocą odwrotnej transformaty Fouriera.



Rys. 1.2: Przykładowy spektrogram wygenerowany przez algorytm *Stable Riffusion* dla danych wejściowych `funk bassline with a jazzy saxophone solo`.

Metody opisane w rozdziale 1 można porównać pod względem ich przydatności dla użytkownika końcowego, czyli osoby zajmującej się produkcją nagrań muzycznych. Metoda pierwsza, generowanie zapisu symbolicznego, może wydawać się mniej zaawansowana niż generowanie całych plików dźwiękowych. Jednakże, z perspektywy użytkownika, zapis symboliczny jest bardziej praktyczny, ponieważ możliwe jest zaimportowanie go do programu DAW i późniejsza modyfikacja zapisu nutowego. Obecnie dostępne modele generujące pełne nagrania z muzyką nie umożliwiają szczegółowego edytowania parametrów wygenerowanego dźwięku, ponieważ operują bardzo wysokopoziomowo – syntezują muzykę na podstawie opisu słownego.

Podsumowując, wykorzystanie wygenerowanego przez komputer zapisu nutowego jest proste, ze względu na symboliczną naturę zapisu. Wykorzystanie wygenerowanego przez komputer **dźwięku** jest ograniczone ze względu na fakt, że do generowania złożonych sygnałów dźwiękowych wykorzystywane są techniki takie jak głębokie sieci neuronowe, w których utrudniona jest dokładna kontrola nad konkretnymi parametrami funkcjonowania sieci.

Niniejsza praca sugeruje nową metodę podejścia do problemu generowania sygnałów dźwiękowych, którego nie da się zaklasyfikować do żadnej z wyżej wymienionych (1) dziedzin komputerowej kompozycji muzycznej. Wynik pracy algorytmu implementowanego w ramach pracy magisterskiej jest **gotowym elektronicznym instrumentem muzycznym**, który może być wykorzystany w programie do komponowania muzyki. Algorytm nie generuje bezpośrednio sygnału dźwiękowego, lecz tworzy graf przetwarzania sygnałów, który jest zrozumiały dla użytkownika i **pozwala na precyzyjne dostosowanie parametrów syntezy**. Tego typu proces generowania grafów przetwarzania sygnałów dźwiękowych może być porównany z procesem projektowania instrumentu muzycznego.

Modyfikowanie ścieżki przetwarzania sygnału jest techniką często wykorzystywaną w muzyce elektronicznej, do tworzenia dźwięków o interesującej barwie bądź dynamice. Syntezatory dźwięku dostępne na rynku często wyposażone są w tzw. *patch bay* (1.3), pozwalający na mody-

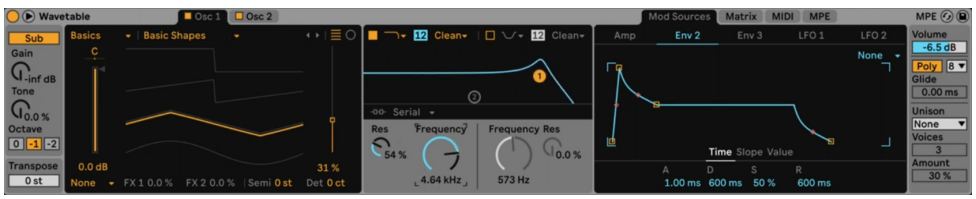
fikowanie grafu przepływu sygnałów wewnątrz syntezatora, bądź połączenie go z zewnętrznym sprzętem muzycznym bądź elektronicznym.

Rys. 1.3: Syntezator *Mother 32* firmy *Moog*, po prawej stronie widoczny jest *patch bay* z podłączonymi przewodami, które nadpisują konfigurację połączeń między układami generującymi i przetwarzającymi sygnał dźwiękowy.

1.1. Cel pracy

Celem pracy jest zbadanie, czy algorytmy optymalizacyjne są w stanie wytworzyć graf przetwarzania sygnałów audio, który wykona syntezę próbki dźwięku zadanej przez użytkownika. Problem poruszany w pracy można zakwalifikować do grupy zagadnień związanych z pojęciami *computer-aided design* oraz *generative artificial intelligence*, zastosowanymi w dziedzienie inżynierii dźwięku. Docelowo zaimplementowany algorytm będzie automatyzował pracę inżyniera dźwięku, tworząc i konfigurując grafy przetwarzania sygnałów dźwiękowych, dostępne w programach typu *digital audio workstation* (1.4). Badania obejmują dwa zagadnienia:

1. **opracowanie metody generowania grafu przetwarzania sygnałów oraz późniejszej modyfikacji grafu – jego struktury i parametrów,**
2. **dobór funkcji celu, na podstawie której algorytm optymalizujący będzie modyfikował graf przetwarzania sygnałów.**

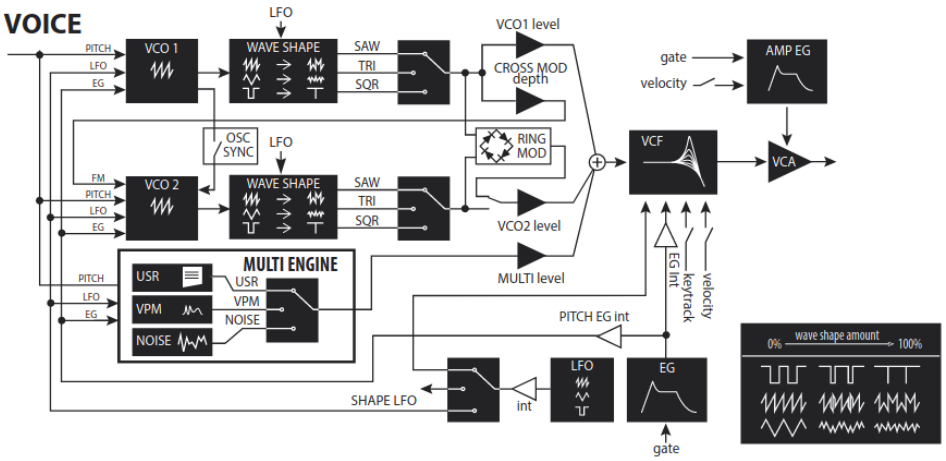


Rys. 1.4: Zbiór parametrów konfigurujących syntezator dźwięku *Wavetable* w programie *Ableton*

1.1.1. Generowanie grafu przetwarzania sygnałów

Proces syntezy dźwięku może być przedstawiony jako graf przetwarzania sygnałów, w którym każdy węzeł wykonuje na sygnale określoną operację. Przykładowy graf przetwarzania sygnału

dla syntezatora analogowego subtraktywnego przedstawiony jest na schemacie 1.5. Pierwsze zagadnienie sprowadza się do opracowania algorytmu pozwalającego na wygenerowanie grafu przetwarzania sygnałów DSP oraz jego późniejszą modyfikację. Przykładem modyfikacji grafu może być wprowadzanie do niego nowych źródeł modulacji bądź zmiana algorytmu generującego sygnał.



Rys. 1.5: Diagram blokowy pojedynczego głosu w syntezatorze *Minilogue xd* firmy *Korg* [28].

1.1.2. Funkcja celu oceniająca podobieństwo barwy dźwieku

Drugie zagadnienie obejmuje przetestowanie szeregu algorytmów, które można wykorzystać do zbudowania funkcji celu, która będzie optymalizowana poprzez „dostrajanie" parametrów i struktury grafu przetwarzania sygnałów dźwiękowych. Praca proponuje wykorzystanie współczynników cepstralnych sygnału (MFCC) w połączeniu z dynamicznym skalowaniem czasu (*dynamic time wrapping*, DTW) jako funkcji celu. Proces wyboru funkcji celu został opisany w rozdziale 3.

1.1.3. Problem optymalizacyjny

W pracy rozwiązywany jest problem optymalizacyjny, w którym struktura oraz parametry grafu przetwarzania sygnałów dostosowywane są tak, aby wygenerować zadaną próbkę dźwięku. Tak wytworzony graf może być wykorzystany jako elektroniczny instrument muzyczny.

1.2. Zakres pracy, plan badań

1.2.1. Metody generowania grafu przetwarzania sygnałów oraz późniejsza modyfikacja grafu

Głównym problemem przy generowaniu grafu przetwarzania sygnałów są ograniczenia nałożone na strukturę grafu, które należy spełnić, by graf był logicznie interpretowalny jako łańcuch przetwarzania sygnałów. Graf musi być grafem skierowanym, który nie zawiera pętli o dodatnim sprzężeniu zwrotnym. Struktura grafu powinna być możliwie jak najbardziej przejrzysta dla użytkownika. Automatyczna ewolucja może dążyć w kierunku wykorzystania nadmiarowej liczby bloków przetwarzania sygnału, jeśli funkcja celu nie będzie zawierała kary za zbyt złożone grafy. Podobne prace [31] wykorzystują podejście oparte o *mixed-typed carthesian genetic*

programming, które będzie punktem startowym dla pracy. Finalnie, badania dążą do wyznaczenia algorytmu o następujący właściwościach:

1. algorytm generuje grafy będące logicznie spójnymi łańcuchami przetwarzania dźwięku (skierowany, bez pętli o dodatnim sprzężeniu zwrotnym w natężeniu sygnału),
2. algorytm maksymalizuje wykorzystanie poszczególnych bloków przetwarzania w grafie, co minimalizuje finalny rozmiar grafu, czyniąc go bardziej czytelnym,
3. generowany graf posiada reprezentację umożliwiającą wykonanie krzyżowania dwóch grafów przetwarzania sygnału. Graf będący wynikiem krzyżowania nadal musi być poprawnym grafem przetwarzania sygnału.

Elementami grafu przetwarzania sygnałów są używane powszechnie w syntezie dźwięku algorytmy:

1. modulacja FM [37] [29],
2. synteza subtraktywna [29] [35],
3. algorytmy *physical modeling* [22] [29],
4. symulacja efektu pogłosu/echa [15] [36].

1.2.2. Dobór funkcji błędu: różnica między wygenerowanym a docelowym sygnałem dźwiękowym

Funkcja celu poszukiwana w ramach projektu musi określać, jak dobrze sygnał wygenerowany przez graf przetwarzania sygnałów pokrywa się z sygnałem docelowym. Porównanie sygnałów musi skupiać się na cechach sygnału, które są najbardziej słyszalne dla ludzkiego ucha. Jednocześnie funkcja nie powinna „karać" sygnałów, które są względem siebie przesunięte w fazie. Wśród algorytmów, które zostały wybrane do przetestowania w ramach projektów zawarte są:

1. algorytmy porównywania sygnałów oparte o transformatę Fouriera [23] [41],
2. techniki wykorzystywane do generowania „cyfrowych podpisów" sygnałów dźwiękowych (*sound fingerprinting*) [25],
3. algorytmy wykrywające spadek jakości dźwięku z perspektywy psychoakustycznej [26] [27].

1.3. Struktura i zawartość pracy

Praca podzielona jest na następujące części:

Definicja problemu (2)

Formalizuje i opisuje problem optymalizacyjny rozwiązywany w pracy.

Analiza i wybór funkcji celu (3)

Porównuje funkcje z dziedziny przetwarzania sygnałów, które pozwalają określić jak podobna jest barwa dźwięku dwóch sygnałów dźwiękowych. Uzasadnia wybór funkcji celu, która została zastosowana w pracy.

Algorytm rozwiązania (4)

Opisuje algorytm wykorzystany do rozwiązania problemu zdefiniowanego w rozdziale 2.

Graf przetwarzania sygnałów (5)

Opisuje zaimplementowane w ramach pracy środowisko eksperymentalne, pozwalające na wytwarzanie grafów przetwarzania sygnałów o dowolnej strukturze. Przedstawia zaimplementowane algorytmy syntezy i przetwarzania sygnałów dźwiękowych.

Wyniki padań (6)

Opisuje proces badawczy, w którym narzędzia wytworzone w rozdziałach 5 oraz 3 zostały wykorzystane do automatycznego wytworzenia grafu DSP, który naśladuje barwę zadanej próbki dźwięku. Porównuje uzyskane wyniki z podobną pracą badawczą [31].

Analiza wyników, dyskusja nad skutecznością działania algorytmu oraz możliwe drogi dalszego rozwoju (7)

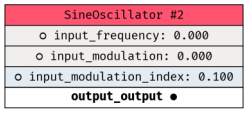
Rozdział podsumowuje uzyskane wyniki badań, podejmuje dyskusję nad ogólną skutecznością i przydatnością zaimplementowanego rozwiązania oraz kreśli potencjalne drogi dalszego rozwoju prac badawczych w podobnej tematyce. W czasie, gdy niniejsza praca była tworzona, zostały opublikowane badania dotyczące podobnego problemu [19], rozdział podejmuje dyskusję o różnicach w podejściu do problemu oraz potencjalnych zalet i wad każdego z podejść.

Rozdział 2

Definicja problemu

Jak opisano w zakresie pracy (1.2), w pracy rozwiązywany jest problem budowy grafu przetwarzania sygnałów oraz opracowywana jest funkcja celu, porównująca dwa sygnały pod względem ich barwy. Następnie algorytm generujący graf przetwarzania sygnałów oraz funkcja porównująca barwy sygnałów są wykorzystane do rozwiązania problemu optymalizacyjnego. Graf przetwarzania sygnałów można opisać jako zbiór połączonych węzłów generujących i przetwarzających sygnał dźwiękowy. Każdy węzeł opisany jest poprzez:

1. zbiór wejść,
2. zbiór wyjść,
3. operację matematyczną wykonywaną na sygnale.



Rys. 2.1: Przykładowy węzeł w grafie, generujący sygnał sinusoidalny z możliwością modulacji.

Przykładowo, dla syntezy subtraktywnej powszechnie wykorzystywane są następujące typy węzłów:

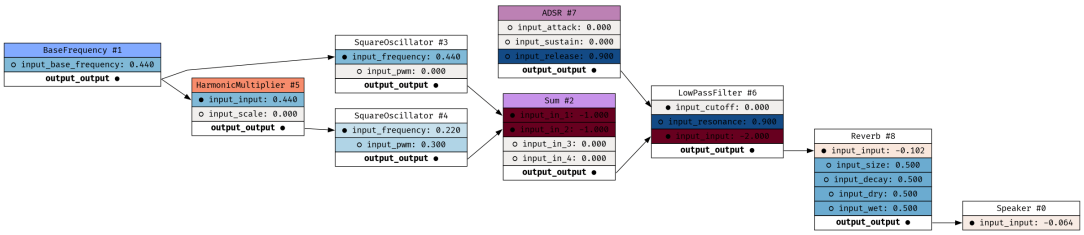
Oscylator (*VCO*):

1. Wejścia:
   - częstotliwość,
   - kształt fali.
2. Wyjścia:
   - wygenerowany sygnał.

Filtr (*VCF*):

1. Wejścia:
   - sygnał wejściowy
   - częstotliwość odcięcia,
   - rezonans.
2. Wyjścia:
   - przefiltrowany sygnał.

2.1. Budowa grafu



Rys. 2.2: Przykładowy graf DSP. Wolne wejścia, które nie są modulowane przez źródła sygnału w grafie są optymalizowanymi parametrami.

Pełny graf przetwarzania można opisać za pomocą zbioru węzłów oraz listy połączeń między węzłami:

$N = [n_1, n_2, .., n_n]$ - liczba węzłów,

$i_j = [p_1, p_2, \ldots, p_m]$ – Zbiór wejść (*inputs*) j-go węzła. $i_{l,m}$ oznacza m-te wejście l-go węzła.

$o_j = [p_1, p_2, \ldots, p_m]$ – Zbiór wyjść (*outputs*) j-go węzła,

$f_i(x)$ – operacja wykonywana na sygnale przez i-ty węzeł.

$C = [\{(j,k),(l,m)\}, \ldots]$: lista połączeń między węzłami, opisujący, które k-te wyjście j-go węzła podłączone jest do którego m-go wejścia l-go węzła. Przykładowo, dla diagramu 2.1, jednym z połączeń będzie $\{(1,0),(5,0)\}$, ponieważ zerowe wyjście węzła `BaseFrequency #1` podłączone jest do zerowego wejścia węzła `HarmonicMultiplier #5`.

Nie wszystkie wejścia w grafie muszą być podłączone do któregoś z wyjść, co jest widoczne na diagramie 2.1. Wejście, które nie zostało nigdzie podłączone przyjmuje jako wartość wejściową parametr liczbowy, optymalizowany na podstawie wartości funkcji celu (2.10). W przypadku schematu 1.5 takimi „wolnymi" wejściami są przykładowo sygnał określający częstotliwość odcięcia filtru sygnału lub parametry określające parametry generatora obwiedni (*EG*). W pracy wykorzystano algorytm genetyczny *differential evolution* [39] do wygenerowania struktury i parametrów grafu. Genotyp opisujący dany graf przetwarzania sygnałów składa się z dwóch części:

1. fragment decydujący o strukturze grafu, $S = [s_1, s_2, \ldots]$,
2. fragment decydujący o wartości parametrów w wolnych wejściach, $P = [p_2, p_2, \ldots]$.

2.1.1. Struktura grafu

Różne rodzaje syntezy dźwięku wykorzystują różnorodne struktury grafu przetwarzania sygnałów [28] [16]. Aby umożliwić dostosowanie grafu przetwarzania sygnałów do wykonywania różnych rodzajów syntezy, struktura grafu przetwarzania sygnałów jest dynamicznie modyfikowana przez algorytm optymalizacji. Algorytm generujący określoną strukturę grafu na podstawie genotypu opisany jest w rozdziale 4. Genotyp odpowiadający za strukturę grafu ma formę krotki liczb rzeczywistych S. Praca definiuje funkcję generującą strukturę grafu G_s z genotypu:

$$G_s(S) = N, C \tag{2.1}$$

2.1.2. Przypisanie parametrów do „wolnych wejść"

Po stworzeniu grafu o danej strukturze G_s, druga część genotypu wykorzystywana jest jako wartości poszczególnych parametrów P dla wolnych wejść w grafie przetwarzania sygnału.

$$\forall_{(l,m)}(l,m) \notin C, i_{l,m} = P_j \tag{2.2}$$

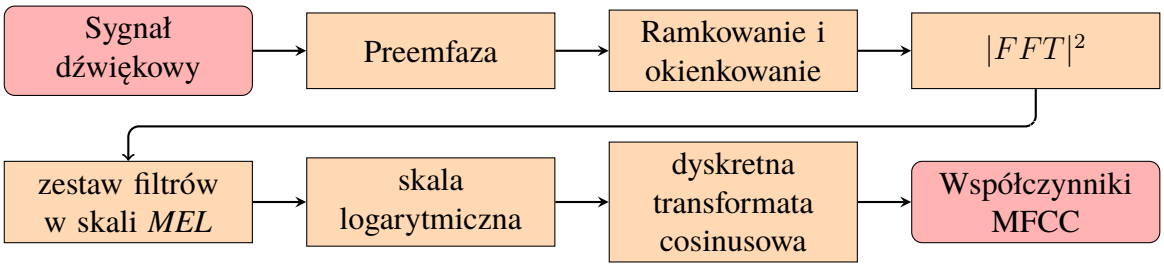
2.2. Funkcja celu

Wykorzystana w pracy funkcja celu F, oceniająca, jak sygnał wygenerowany ($\bar{x}$) przez algorytm jest bliski sygnałowi docelowemu (x) przedstawiona jest w następujący sposób:

$$F(x, \bar{x}) = DTW(MFCC(x),\ MFCC(\bar{x})) \tag{2.3}$$

Gdzie $MFCC$ oznacza *mel-frequency cepstrum coefficients* (2.2.1), natomiast DTW oznacza algorytm *dynamic time warping* [38]. Uzasadnienie wybranej funkcji celu opisane jest w rozdziale 3.

2.2.1. Wyliczanie współczynników MFCC [24]



Rys. 2.3: Schemat algorytmu obliczania współczynników MFCC, zaczerpnięty z [24]. Praca wykorzystuje gotową implementację algorytmu obliczającego współcznyniki MFCC z pakietu `librosa` [33].

Algorytm obliczania współczynników MFCC przedstawiony jest na rysunku 2.2.1. Pierwszym krokiem w algorytmie jest zastosowanie preemfazy, która wzmacnia składowe wysokoczęstotliwościowe i osłabia składowe niskoczęstotliwościowe:

$$x'_n = x_n - a\ x_{n-1} \tag{2.4}$$

W następnym kroku sygnał jest ramkowany, na każdą ramkę nakładane jest okno Hamminga:

$$Ham(N) = 0.54 - 0.46\cos(2\pi\frac{n-1}{N-1}) \tag{2.5}$$

Dla każdej ramki wyliczana jest transformata Fouriera, aby obliczyć widmo mocy sygnału $|FFT|^2$. Widmo przetwarzane jest przez zbiór filtrów H_m, których środki są rozmieszczone w równomiernych odstępach w skali mel. Typ i liczba filtrów zależy od implementacji algorytmu (w pracy wykorzystano [33]). Wyjście każdego z filtrów wykorzystane jest do obliczenia energii przefiltrowanego pasma:

$$S_m = \sum_{k=1}^{N} |X_r(k)|^2 H_m(k), \tag{2.6}$$

gdzie X_r oznacza widmo danej ramki, m jest numerem filtra.

W ostatnim kroku do obliczenia wartości współczynników MFCC wykorzystuje się dyskretną transformatę kosinusową. Aby lepiej przybliżyć wrażliwość ludzkiego ucha na głośność dźwięku, wykorzystuje się logarytm energii pasma:

$$c_i = \sqrt{\frac{2}{M}} \sum_{m_1}^{M} \log(S_m)\cos(\frac{\pi i}{M}(M - 0.5)), \tag{2.7}$$

Gdzie M to liczba użytych filtrów w zbiorze 2.6, a i jest numerem współczynnika.

2.3. Ograniczenia

W algorytmach DSP powszechnie wykorzystuje się ograniczenie wartości sygnału do przedziału $(-1.0, 1.0)$, gdy implementacja danego algorytmu wykorzystuje typ `float` do przechowywania wartości sygnałów. Środowisko zaimplementowane w ramach pracy oczekuje, że wartości sygnałów i parametrów sterujących będą znajdowały się w tym przedziale:

$$\forall p_j \in P, -1.0 \leq p_j \leq 1.0, \quad \forall s_i \in S, -1.0 \leq s_i \leq 1.0 \tag{2.8}$$

2.4. Problem optymalizacji

Sygnał wygenerowany przez graf przetwarzania sygnałów zależy od następujących parametrów:

$$\bar{x} = G(G_s(S), P) \tag{2.9}$$

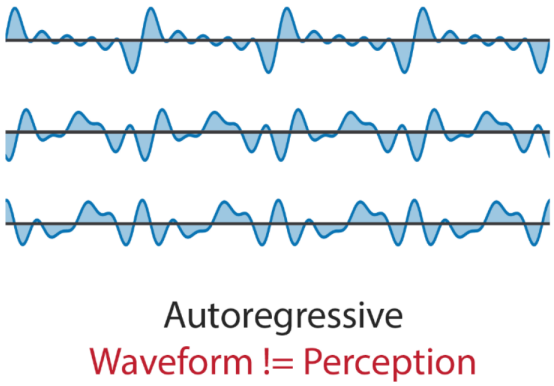
Gdzie G jest funkcją generującą sygnał dźwiękowy dla konkretnej struktury grafu G_s (2.1.1, 2.1), dla wartości parametrów przypisanych z P (2.1.2, 2.2). Dla parametrów S oraz P, ograniczonych przez 2.8, rozwiązywany jest problem optymalizacyjny:

$$\text{Minimize } F(x,\ G(G_s(S), P)) \tag{2.10}$$

Rozdział 3

Funkcja celu – porównanie barwy dźwięku

Aby stopniowo dostosować graf przetwarzania sygnałów zaimplementowany w rozdziale 5 do imitowania zadanej próbki dźwięku, należy wykorzystać funkcję celu, która maleje wraz ze wzrostem podobieństwa barwy dźwięku między próbki zadaną i sygnałem generowanym przez graf.



Rys. 3.1: Przykład trzech próbek dźwięku, które dla słuchacza brzmią identycznie, mimo znacznych różnic w kształcie fali. Źródło obrazka: [17].

3.1. Porównanie barwy dźwięku w literaturze

Żadna z prac przeanalizowanych podczas przeglądu literatury ([17], [19], [12], [20], [31], [11], [38]) nie wykorzystuje metod porównywania sygnału osadzonych jedynie w dziedzinie czasu, ponieważ nie są one skuteczne do porównywania dźwięków pod względem odczuć psychoakustycznych. Przykład różnych kształtów fali, które z perspektywy słuchacza brzmią jak taki sam dźwięk zademonstrowano na rysunku 3.

Ponieważ porównywanie barwy dźwięku instrumentów muzycznych nie należy do popularnych tematów badań, podczas przeglądu literatury wykorzystano również badania dotyczące rozpoznawania głosu, wykorzystujące współczynniki MFCC oraz *dynamic time warping* (DTW) [38].

3.1.1. Systematyzacja metod z literatury

Metody zaczerpięte z literatury wykorzystują różne podejścia do porównywania barwy dźwięku pomiędzy sygnałami. Podejścia te można usystematyzować za pomocą dwóch cech:

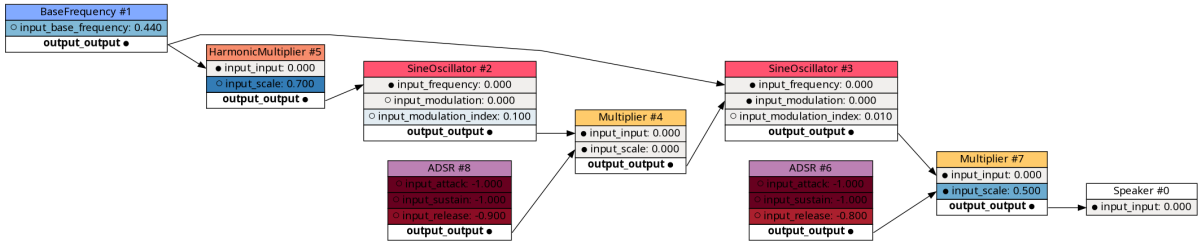
1. Rodzaj wykonanej transformacji z dziedziny czasu do dziedziny częstotliwości:
   - transformata Fouriera (w różnych konfiguracjach) [20] [12],
   - MFCC [19] [31] [38].
2. Dalsze przetwarzanie reprezentacji sygnału w domenie częstotliwości, w celu ułatwienia optymalizacji:
   - dostosowywanie wagi konkretnych próbek na podstawie metryki określającej siłę sygnału (na przykład *root-mean-square*, RMS) [11], aby wzmocnić istotność głośniejszych fragmentów dźwięku,
   - wykorzystanie *dynamic time warping*, aby funkcja celu przyzwalała na niedokładności w odwzorowaniu dokładnej dynamiki zmian w charakterystyce spektralnej [38].

3.1.2. Wybór funkcji celu do przetestowania

Na podstawie analizy metod z literatury opisanej w rozdziale 3.1.1 zostały wybrane wszystkie warianty funkcji celu wykorzystywane w przeanalizowanej literaturze:

1. Różnica w spektrum Fouriera,
2. Różnica w spektrum Fouriera liczona za pomocą DTW,
3. Różnica w MFCC,
4. Różnica w MFCC liczona za pomocą DTW,
5. Różnica w MFCC ważonym za pomocą RMS.

3.2. Proces testowania funkcji celu



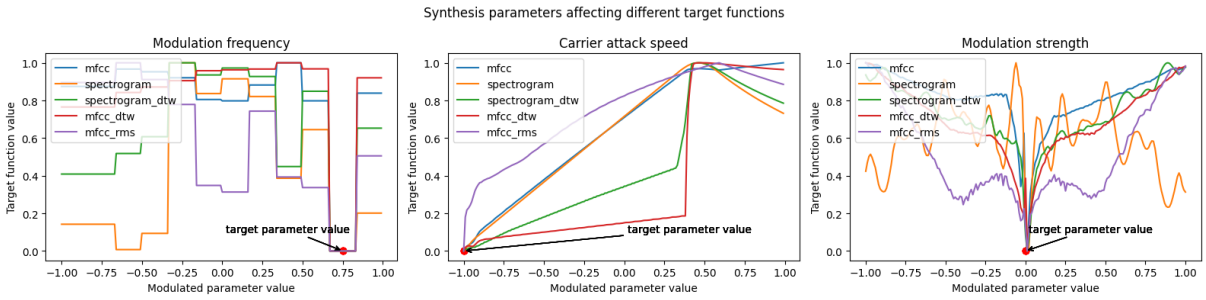
Rys. 3.2: Prosty graf syntezy FM, zawierający jeden oscylator służący za sygnał nośny i jeden oscylator służący za sygnał modulujący.

Metoda testowania została zaczerpnięta z [31]. Funkcje celu zostały wpierw przetestowane poprzez wykonanie zbioru przekrojów przez uproszczony problem syntezy typu FM. Następnie przeprowadzono próby automatycznego dostosowania parametrów dwóch grafów o predefiniowanej strukturze, dla syntezy FM oraz *analog modeling*.

3.3. Przekrój wartości funkcji celu dla prostego problemu syntezy typu FM

Testy obejmowały wygenerowanie wartości funkcji celu podczas modyfikowania pojedynczego parametru w grafie przetwarzania sygnałów przedstawionym na rysunku 3.2. Modyfikowane parametry odpowiadają za różne cechy barwy uzyskanego dźwięku:

- `HarmonicMultiplier/input_scale`: częstotliwość modulacji FM,
- `SineOscillator/input_modulation_index`: siła składowych harmonicznych,
- `ADSR/input_attack`: dynamika dźwięku.



Rys. 3.3: Zmiany w wartościach testowanych funkcji celu podczas przesuwania różnych parametrów syntezy dźwięku. Kształt pierwszego wykresu wynika z zastosowania kwantyzacji dostępnych częstotliwości modulacji, aby wykluczyć nieharmoniczne stosunki częstotliwości modulacji i nośnej. Tego rodzaju praktyka jest wykorzystywana w syntezatorach FM [16], ponieważ ułatwia dostosowywanie parametrów syntezy.

3.3.1. Analiza wyników

Wyniki testów zaprezentowane na wykresach 3.3 pozwalają na wyeliminowanie różnicy między spektrogramami jako funkcji celu, ponieważ w przypadku zmian częstotliwości modulacji posiada ona minimum globalne w niewłaściwej pozycji parametru. Późniejsze testy wykorzystują tylko funkcje celu wykorzystujące MFCC.

Częstotliwość modulacji FM

Wszystkie funkcje z wyjątkiem różnicy między spektrogramami pokazują poprawną, najniższą wartość dla właściwej wartości parametru.

Dynamika dźwięku

W przypadku wpływu zmian w dynamice dźwięku na wartości funkcji celu, zastosowanie DTW znacząco zmienia kształt funkcji celu, zależnie od wybranego rozmiaru okna DTW. Duży rozmiar okna powoduje zmniejszenie kary za niedokładne odwzorowanie dynamiki dźwięku.

Siła modulacji

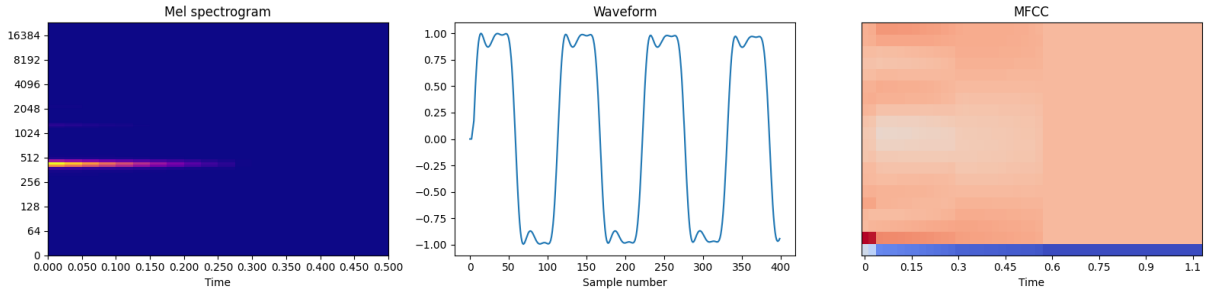
Różnica między spektrogramami jest najbardziej chaotyczna, nie maleje wraz ze zbliżaniem się do poprawnej wartości parametru. Pozostałe funkcje celu wykorzystujące *MFCC* mają lepszą charakterystykę – maleją wraz ze zbliżaniem się do poprawnej wartości parametru.

3.4. Optymalizacja parametrów dla predefiniowanych grafów syntezy FM oraz *analog modeling*

Drugą częścią procesu testowania funkcji celu z literatury było zweryfikowanie skuteczności każdej z funkcji w uproszczonym problemie optymalizacyjnym, polegającym jedynie na odnalezieniu właściwych parametrów dla predefiniowanej struktury grafu DSP. Wykorzystano dwa grafy DSP (3.5, 3.2), wykonujące różne rodzaje syntezy.

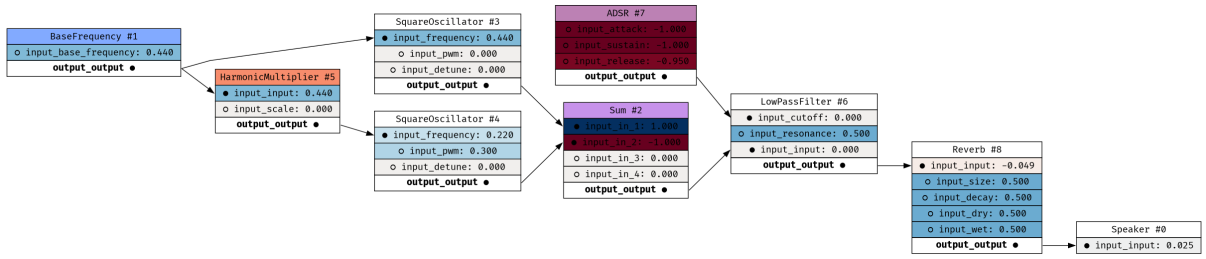
3.4.1. Synteza FM

Testowana struktura grafu wykonującego syntezę typu FM 3.2 wykorzystuje 2 operatory: jedną nośną i jeden modulator. Parametry grafu zostały ręcznie dostrojone aby wygenerować krótki dźwięk typu *pluck*, w którym modulator przekształca nośną sinusoidę w sygnał zbliżony do sygnału prostokątnego. Z perspektywy wynikowego spektrum sygnału, przedstawionego na rysunku 3.8 sygnał składa się z częstotliwości podstawowej i jednej składowej harmonicznej. Wizualizacja sygnału została przedstawiona na wykresie 3.4.



Rys. 3.4: Spektrogram, kształt fali oraz wizualizacja MFCC dla próbki dźwięku, którą ma imitować graf syntezy FM podczas testów różnych funkcji celu.

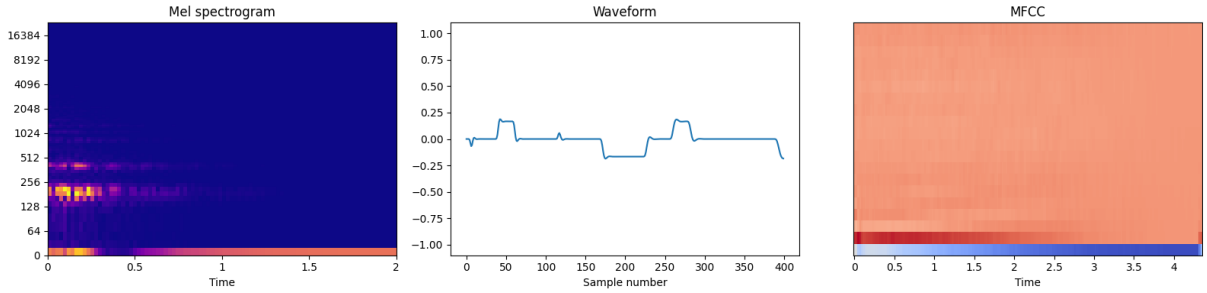
3.4.2. Synteza *analog modeling*



Rys. 3.5: Graf wykonujący syntezę typu *analog modeling*, wykorzystany do testów funkcji celu.

Synteza *analog modeling* zazwyczaj wykorzystuje więcej parametrów niż synteza FM (3.2), aby zwiększyć trudność problemu optymalizacyjnego graf został roszszerzony o węzeł dodający efekt pogłosu [15]. Struktura grafu (przedstawiona na rysunku 3.5) składa się z dwóch oscylatorów generujących sygnał prostokątny. Oscylator `SquareOscillator #4` generuje sygnał przesunięty o oktawę w dół w stosunku do częstotliwości podstawowej, jednocześnie jego parametr `input_pwm` skraca szerokość generowanego impulsu, aby wzbogacić barwę dźwięku o dodatkowe składowe harmoniczne. Barwa dźwięku zmienia się dynamicznie w czasie dzięki zastosowaniu filtra niskoprzepustowego (`LowPassFilter #6`), którego częstotliwość odcięcia

jest modulowana przez sygnał sterujący `ADSR #7`. Długość dźwięku generowanego przez oscylatory jest podobna jak w przypadku grafu FM (3.2), zastosowanie węzła `Reverb #8` przedłuża czas trwania dźwięku i dodatkowo „rozmywa go" w czasie, co pokazuje spektrum sygnału na wykresie 3.6.

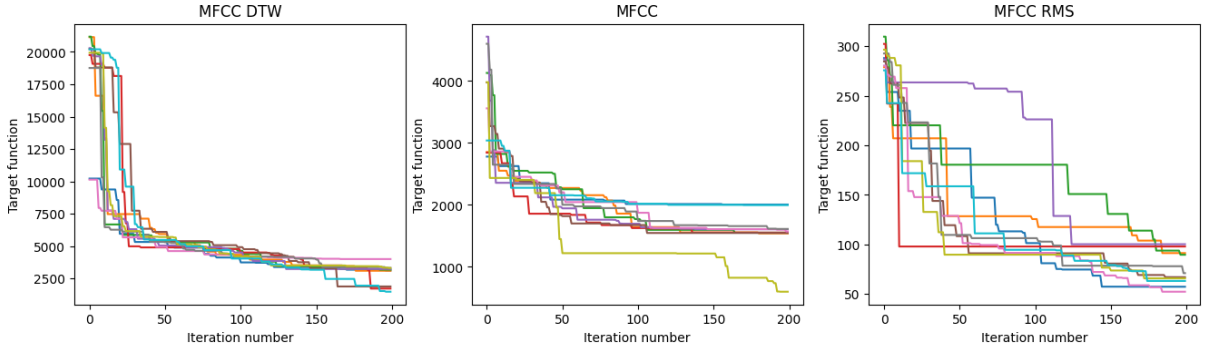


Rys. 3.6: Spektrogram, kształt fali oraz wizualizacja MFCC dla próbki dźwięku, którą ma imitować graf syntezy *analog_modeling* podczas testów różnych funkcji celu.

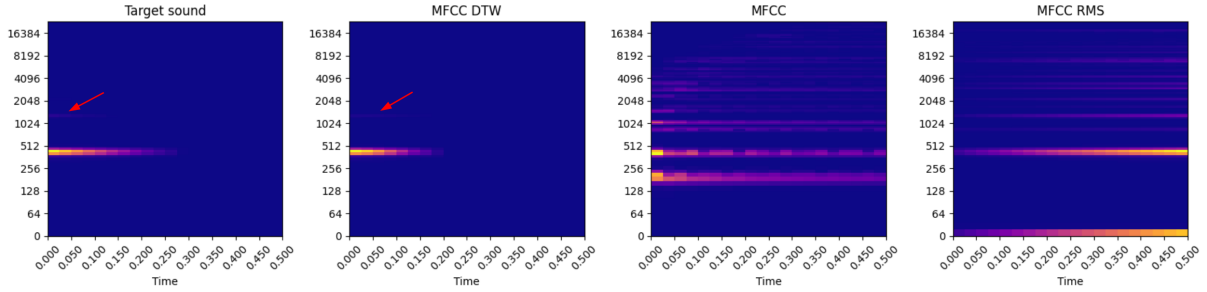
3.4.3. Plan testów

Dla obu grafów wykonano optymalizację parametrów wejściowych w celu imitacji danej próbki dźwięku. Optymalizację wykonano 10 razy dla każdej rozważanej (3.1.2) funkcji celu. Do optymalizacji parametrów wykorzystano algorytm evolucyjny *differential evolution* [39].

3.4.4. Wyniki testów



Rys. 3.7: Wykresy zmian funkcji celu podczas optymalizacji dla grafu syntezy FM.

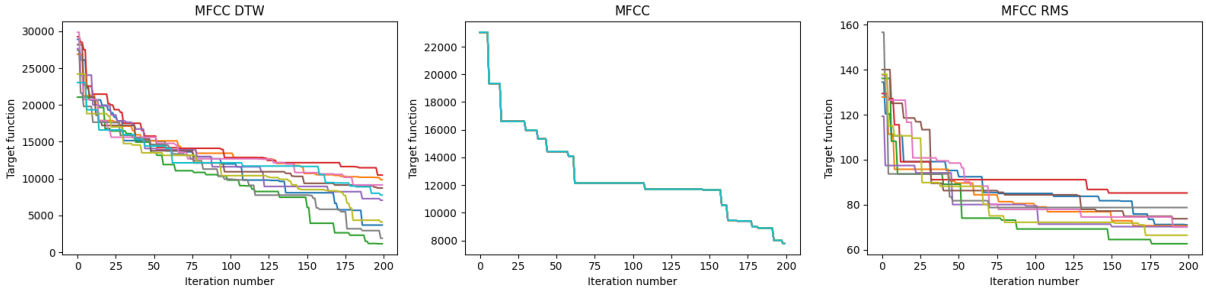


Rys. 3.8: Spektrogram dźwięku docelowego oraz dźwięków uzyskanych w procesie optymalizacji parametrów grafu FM. Czerwoną strzałką oznaczono składową harmoniczną (słabo widoczną na spektrogramie), która została poprawnie odtworzona przez algorytm optymalizacji.

Synteza FM

Algorytm optymalizacji jest w stanie poprawnie dostosować wartości parametrów grafu w przypadku zastosowania MFCC+DTW jako funkcji celu. Jak pokazuje spektrogram (3.8), poprawnie odtworzona jest zarówno dynamika dźwięku i składowa harmoniczna. Pozostałe funkcje celu nie pozwalają na odtworzenie sygnału, który w bliski sposób przypomina dźwięk docelowy, pomimo uzyskiwania wyników zbliżonych do MFCC+DTW we wcześniejszych testach (3.3).

Synteza *analog modeling*



Rys. 3.9: Wykresy zmian funkcji celu podczas optymalizacji dla grafu syntezy *analog modeling*. **TODO: pełny wykres**.

Rys. 3.10: Spektrogram dźwięku docelowego oraz dźwięków uzyskanych w procesie optymalizacji parametrów grafu *analog modeling*. **TODO: obrazek**.

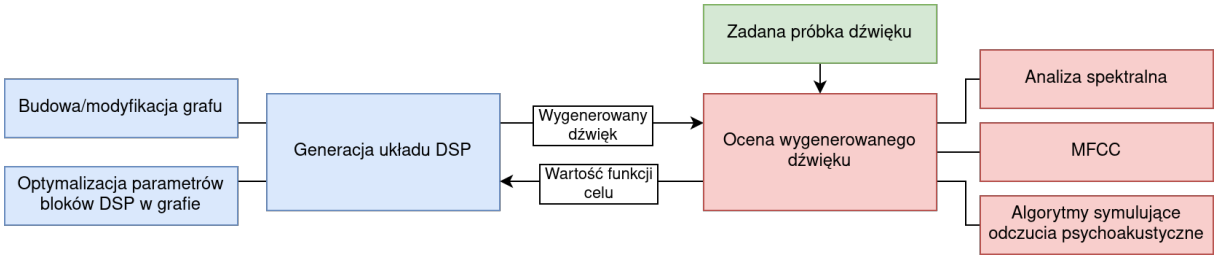
Podobnie jak podczas testów syntezy FM, funkcja celu MFCC+DTW pozwala na dokładne odwzorowanie barwy dźwięku. Na spektrogramie (3.10) widoczne są poprawnie odtworzone składowe harmoniczne, przebieg dynamiczny dźwięku oraz parametry efektu pogłosu. **Tutaj trzeba dodać więcej tekstu i spektrogramy jak się już przekręci cała optymalizacja.**

3.4.5. Wybór funkcji celu na podstawie wyników

Wyniki testów pozwalają jednoznacznie wybrać funkcję celu, która porównuje wartości MFCC sygnałów za pomocą algorytmu *dynamic time warping*. W zakresie pracy nie leży szczegółowe wytłumaczenie, czemu zastosowanie DFT usprawnia proces optymalizacji. Możliwym intuicyjnym wytłumaczeniem tego fenomenu jest fakt, że DFT pozwala na rozpoznanie poszczególnych fonemów w nagraniach mowy [38], niezależnie od prędkości wypowiadania słów. Analogicznie, w przypadku porównywania sygnałów dźwiękowych generowanych przez grafy przetwarzania sygnałów, wykorzystanie DFT może powodować „wygładzenie" niedokładności w zmianach tembru (transjentach [4]) i dynamiki dźwięku, które występują pomiędzy sygnałem docelowym i wygenerowanym.

Rozdział 4

Algorytm rozwiązania

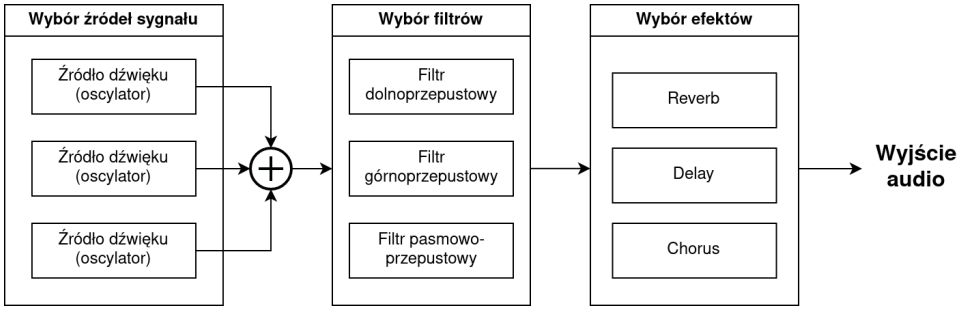


Rys. 4.1: Diagram algorytmu rozwiązania zaimplementowanego w ramach pracy. Algorytm oceny może wykorzystywać różne funkcje celu, finalnie zastosowano MFCC oraz *dynamic time wrapping*, proces wyboru funkcji celu opisuje rozdział 3.

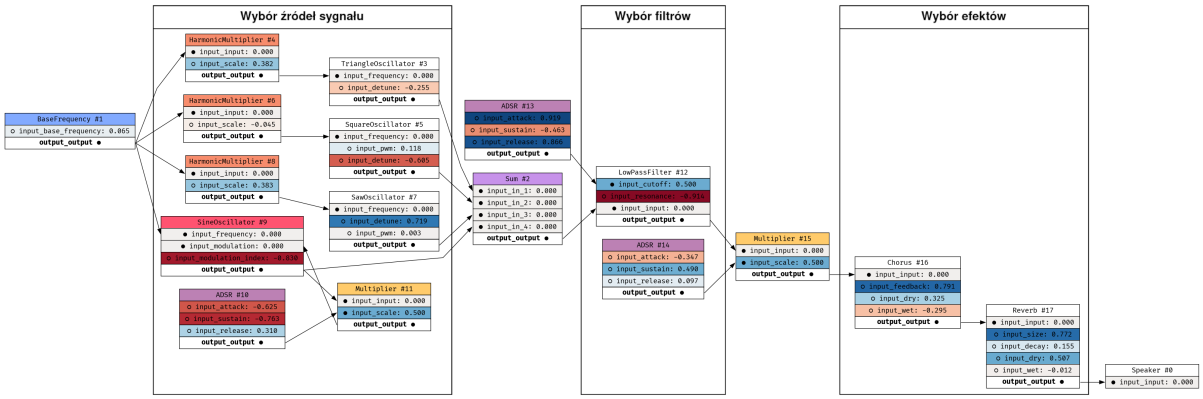
Jak opisano w rozdziale poświęconym definicji problemu (2), praca rozwiązuje problem budowy grafu DSP z wykorzystaniem dwóch algorytmów:

1. Algorytm generujący graf DSP, opisany w rozdziale 5,
2. Algorytm oceniający jak bardzo wygenerowany dźwięk jest bliski dźwiękowi docelowemu pod względem barwy, opisany w rozdziale 3.

Praca wykorzystuje algorytm genetyczny, w którym genotyp odpowiada za strukturę grafu 4.3 oraz wartości przypisane do parametrów grafu (2.1, 2.2). Algorytm generuje różne warianty grafów przetwarzania sygnałów powszechnie wykorzystywane w syntezatorach dźwięku [28] [16] [6]. Takie podejście pozwala na dostosowanie grafu do danego rodzaju syntezy, lub połączenie wielu typów syntezy w celu uzyskania bardziej złożonego brzmienia. Algorytm wybiera węzły dla każdej sekcji zilustrowanej na rysunku 4.2.



Rys. 4.2: Sekcje przetwarzania sygnałów oraz przykładowe węzły przetwarzania sygnałów, które są w nich powszechnie wykorzystywane.



Rys. 4.3: Przykład wygenerowanej struktury grafu, oznaczono segmenty z diagramu 4.2.

Przykładowo, w syntezatorach wykorzystujących syntezę typu FM [6], źródłem sygnału będą operatory wykorzystujące proste sygnały sinusoidalne, poddane modulacji częstotliwości. Dla syntezatorów analogowych (subtraktywnych), zamiast prostych sygnałów wykorzystywane są oscylatory generujące fale kwadratowe i piłokształtne, o dużej liczbie składowych harmonicznych. Uzasadnia to wykorzystanie filtrów dolnoprzepustowych, które z kolei nie występują w tradycyjnych syntezatorach FM (pojawiają się dopiero we współczesnych modelach [16]). Instrumenty eksperymentalne, wykorzystujące mniej popularne rodzaje syntezy, takie jak *physical modeling* [7], często opierają się jedynie na rozbudowanej sekcji oscylatorów, które posiadają wystarczająco dużo parametrów by wynagrodzić tym brak sekcji subtraktywnej. Po drugiej stronie spektrum znajdują się instrumenty wykorzystujące głównie sekcję efektów i syntezę granularną [3], aby w nieoczekiwany sposób modyfikować proste próbki dźwięku.

4.1. Wybór źródeł sygnału

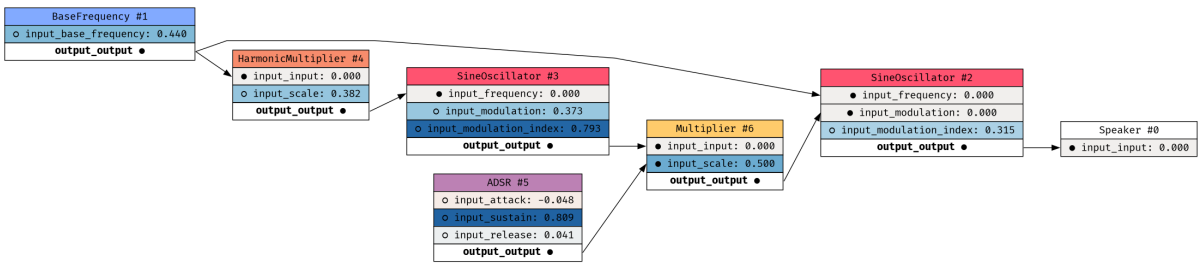
Graf posiada 3 sloty dla węzłów generujących sygnał. W pracy zaimplementowano różne typy syntezy, które mogą być wykorzystane przez algorytm do syntezy dźwięków o różnorodnych barwach.

4.1.1. Synteza FM

Zaimplementowane w pracy geny odpowiedzialne za syntezy FM odwzorowują uproszczone algorytmy syntezy wykorzystane w syntezatorze *Yamaha DX7* [6].

Gen `FM1`

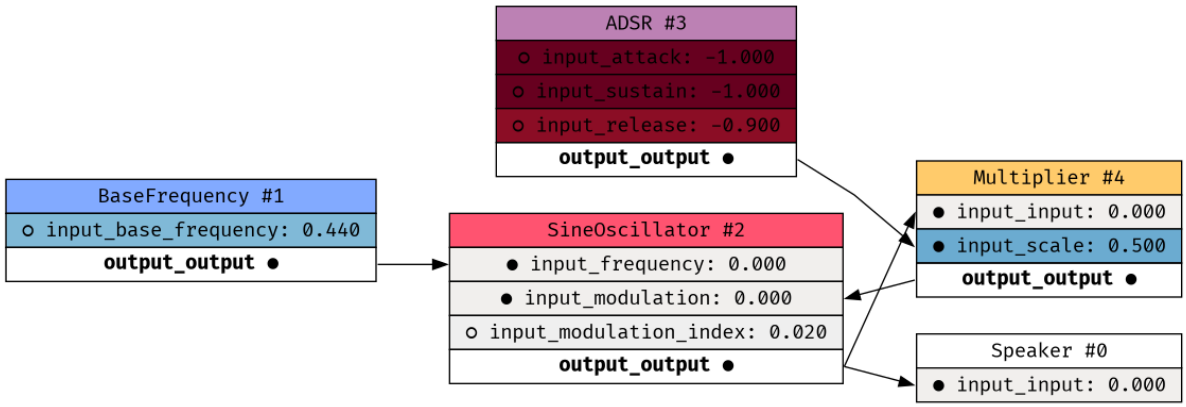
Gen `FM1`



Rys. 4.4: Graf wykorzystujący gen `FM1`.

Gen `FM1`, przedstawiony na diagramie 4.5, typową dla syntezy FM modulację fali sinusoidalnej za pomocą innej fali sinusoidalnej. Taki układ umożliwia uzyskanie dźwięków przypominających dźwięki fletu, trąbki lub dzwonków.

Gen `FM2`



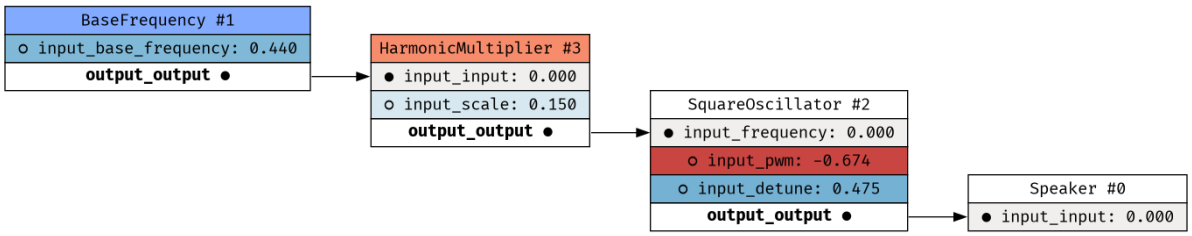
Rys. 4.5: Graf wykorzystujący gen `FM2`.

Gen `FM2`, zilustrowany na diagramie 4.5, wykorzystuje pojedynczy operator ze sprzężeniem zwrotnym. W zależności od ustawionych parametrów, taka struktura pozwala na uzyskanie dźwięków przypominających uderzenie w strunę lub służyć jako źródło szumu.

4.1.2. Synteza *analog modeling*

Zaimplementowane w pracy geny odpowiedzialne za syntezy FM odwzorowują uproszczone algorytmy syntezy wykorzystane w syntezatorze *Korg Minilogue xd* [6].

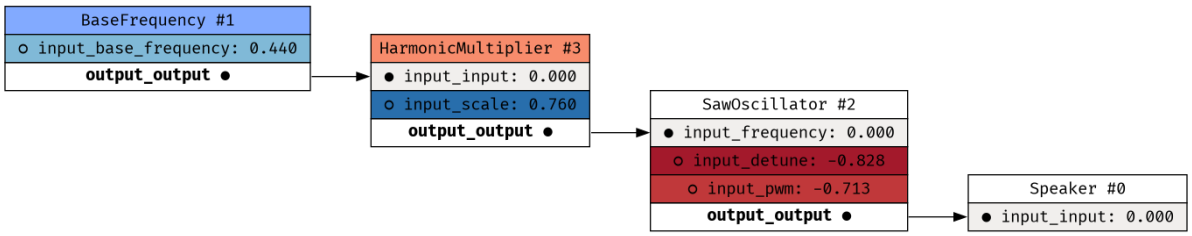
Gen `AN1`



Rys. 4.6: Graf wykorzystujący gen `AN1`.

Gen `AN1` (diagram 4.6) pozwala na uzyskanie fal prostokątnych o różnej szerokości impulsów.

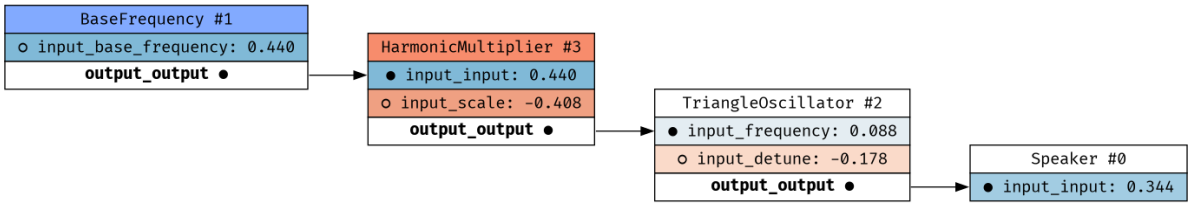
Gen AN2



Rys. 4.7: Graf wykorzystujący gen AN2.

Gen AN2, przedstawiony na diagramie 4.7, pozwala na uzyskanie sygnałów piłokształtnych.

Gen AN3



Rys. 4.8: Graf wykorzystujący gen AN3.

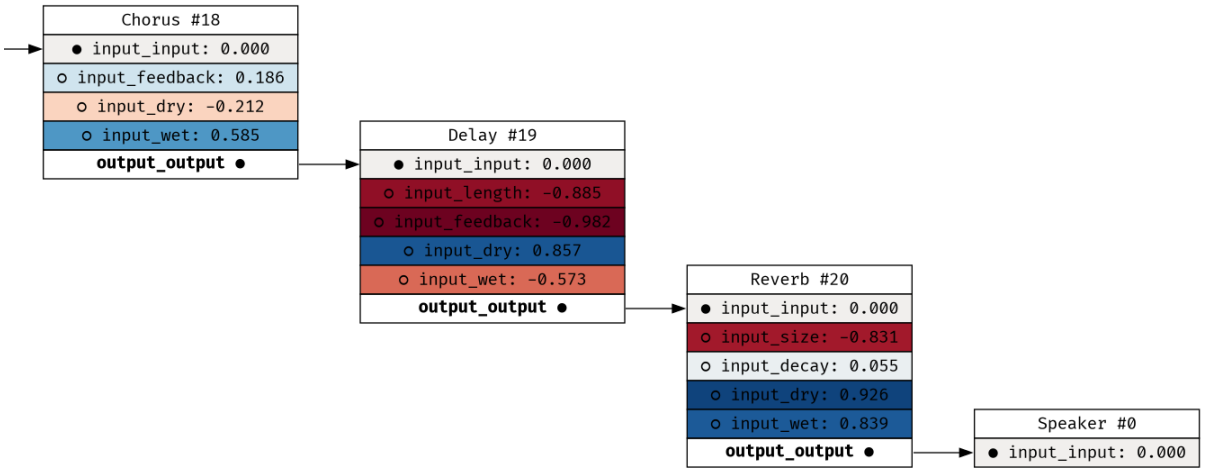
Gen AN3, (diagram 4.8), generuje sygnały trójkątne.

4.2. Wybór filtrów

W pracy wykorzystano gotową implementację filtru cyfrowego, emulującego rezonansowy filtr drabinkowy [2]. Implementacja została wykorzystana do wytworzenia filtrów dolno oraz górnoprzepustowego.

4.3. Wybór efektów

W pracy wykorzystano gotowy algorytm pogłosu (*reverb*) oparty na [15]. Algorytm generujący echo (*delay*) został zaimplementowany jako bufor kołowy. Algorym generujący efekt *chorus* został zaimplementowany jako wariant algorytmu *delay*, w którym długość bufora jest modulowana przez falę sinusoidalną. Efekty połączone sa w łańcuch $chorus \rightarrow delay \rightarrow reverb$. Algorytm generacji wybiera, które efekty będą obecne w grafie. Diagram 4.9 ilustruje fragment grafu, w którego strukturze znajdują się wszystkie 3 efekty.

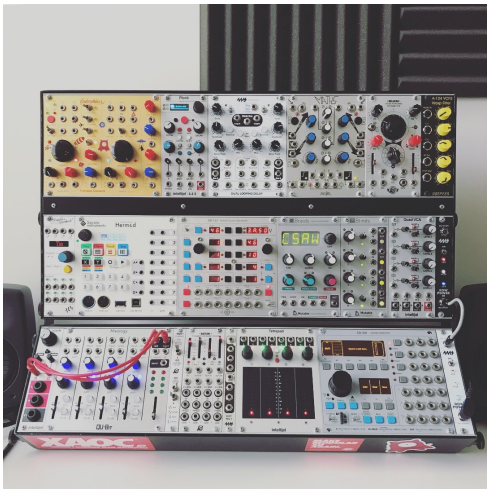


Rys. 4.9: Przykładowy łańcuch efektów w grafie.

Rozdział 5

Graf przetwarzania sygnałów

Na potrzeby badań zostało zaimplementowane środowisko, pozwalające na dynamiczne tworzenie grafów przetwarzania sygnałów (5.1). W projekcie nie zostało zastosowane gotowe rozwiązanie symulujące syntezator modułowy, takie jak Bespoke Synth [13], VCVRack [5] lub Pure Data [9], ponieważ nie udostępniały one gotowego interfejsu pozwalającego na łatwą integrację z językiem `Python`. Istniejące w internecie gotowe przykłady algorytmów syntezy audio pozwoliły na szybkie zaimplementowanie środowiska eksperymentowego, które posiada szeroki zbiór dostępnych algorytmów DSP oraz w przystępny sposób interfejsuje się z językiem `Python`, co umożliwia wykorzystanie gotowych pakietów obliczeniowych z dziedziny przetwarzania sygnałów.



Rys. 5.1: Przykładowy układ modułów w standardzie *Eurorack* [1]. W prawym dolnym rogu widoczne połączenia modulujące między modułami.

5.1. Podstawy syntezy dźwięku w syntezatorach modułowych

Proces syntezy dźwięku może zostać przedstawiony jako zbiór węzłów wykonujących syntezę lub przetwarzanie sygnału audio oraz połączeń między węzłami. Przykładowe elementy grafu:

1. Węzły:
    1. generujące sygnał:
        - synteza sygnałów (sinusoida, trójkąt, sygnał prostokątny),

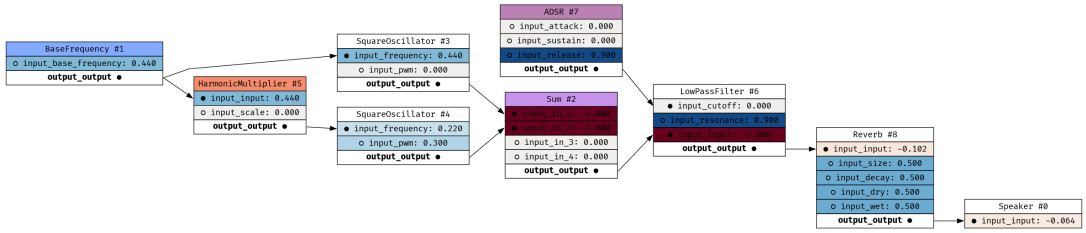
- sygnał modulujący (LFO, ADSR).

2. przetwarzające sygnał:
   - filtry (górnoprzepustowy, dolnoprzepustowy, pasmowo-przepustowy),
   - efekty (pogłos, echo).

2. Połączenia między węzłami:
   - modulowanie parametrów syntezy i przetwarzania sygnału.

Odpowiednikiem implementowanego środowiska w świecie rzeczywistym są syntezatory modułowe (przykładowo 5), które pozwalają na dowolne łączenie modułów wykonujących operacje DSP. Barwę dźwięku w syntezatorze modyfikuje się na dwa sposoby:

1. Ustawienie stałej wartości danego parametru w węźle DSP,
2. Modulacja wartości danego parametru w węźle DSP za pomocą wartości wyjściowej innego węzła.



Rys. 5.2: Przykładowy układ węzłów DSP w zaimplementowanym środowisku eksperymentowym. Układ wykonuje syntezę subtraktywną z modulowaną wartością częstotliwości granicznej filtru niskoprzepustowego oraz dodaje efekt pogłosu (*reverb*) [15].

Dla przykładowego układu DSP, przedstawionego na rysunku 5.1, skonfigurowane są między innymi parametry:

1. Częstotliwość podstawowa (węzeł `BaseFrequency #1`),
2. wartość, przez którą mnożona jest częstotliwość podstawowa w węźle `HarmonicMultiplier #5`,
3. Wartości `input_pwm` w węzłach `SquareOscillator #3` oraz `#4`,
4. Parametry algorytmu pogłosu w węźle `Reverb #8`.

Z kolei wartość parametru `input_cutoff` w węźle `LowPassFilter #6` **jest modulowana** przez sygnał wychodzący w węźle `ADSR #7`, co pozwala na dynamiczne zmiany częstotliwości odcięcia filtru w czasie, wzbogacając barwę generowanego dźwięku.

5.2. Wymagania

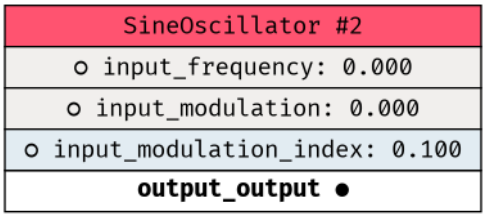
W ramach pracy zostały zdefiniowane wymagania dotyczące implementowanego później środowiska eksperymentowego, opisane w niniejszym rozdziale.

5.2.1. Węzły DSP

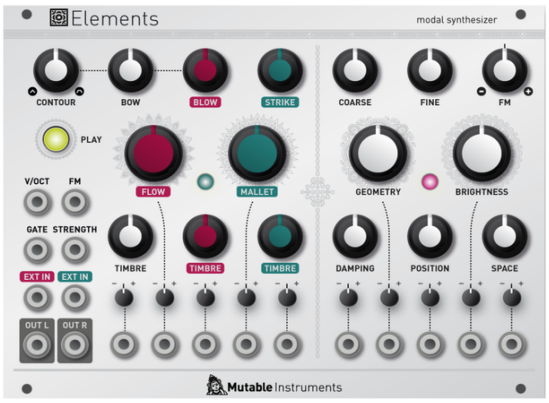
Pojedynczy węzeł DSP może zostać opisany za pomocą trzech cech:

1. Zbiór sygnałów wejściowych,
2. zbiór sygnałów wyjściowych,

3. wykonywana przez węzeł operacja.



Rys. 5.3: Węzeł DSP w zaimplementowanym środowisku eksperymentowym, generujący falę sinusoidalną z możliwością modulacji fazy.



Rys. 5.4: Moduł syntezy *Mutable Instruments Elements*, umożliwiający ręczne ustawianie **oraz** modulację parametrów. Moduł wykonuję syntezę typu *physical modeling* [22].

Przykładowo, przedstawiony na rysunku 5.2.1 węzeł posiada:

1. sygnały wejściowe:
   - `input_frequency` - częstotliwość generowanej sinusoidy,
   - `input_modulation` - wartość modulacji fazy, według równania 5.1,
   - `input_modulation_index`.
2. Sygnały wyjściowe:
   - `output_output` - wartość generowanego sygnału sinusoidalnego.

Węzeł generuje sygnał sinusoidalny o fazie modulowanej poprzez parametr `input_modulation` z siłą modulacji ustawianą przez parametr `input_modulation_index`, opisane za pomocą równania 5.1 (jest to uproszczenie równania syntezy FM przedstawionego w [37]) oraz listingu 5.1:

$$f(t) = \sin(t * f + m * m_i) \tag{5.1}$$

Listing 5.1: Implementacja węzła SineOscillator.

```
impl DspNode for SineOscillator {
    fn tick(&mut self) {
        let frequency = (self.input_frequency * 1000.0).abs();
        let phase_diff = (2.0 * std::f64::consts::PI * frequency) /
            ↪ SAMPLE_RATE;
        self.output_output =
```

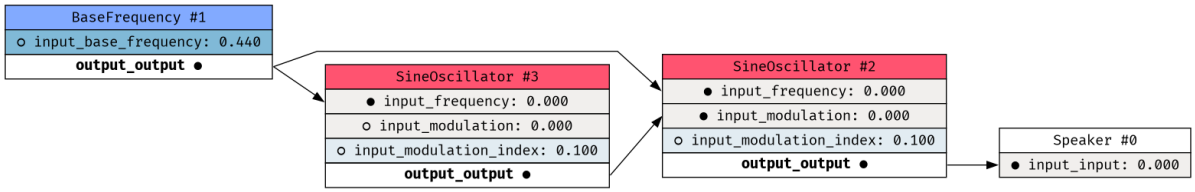
```
            (self.phase + self.input_modulation * self.
               ↪ input_modulation_index * 10.0).sin();
        self.phase += phase_diff;

        while self.phase > std::f64::consts::PI * 2.0 {
            self.phase -= std::f64::consts::PI * 2.0
        }
    }
}
```

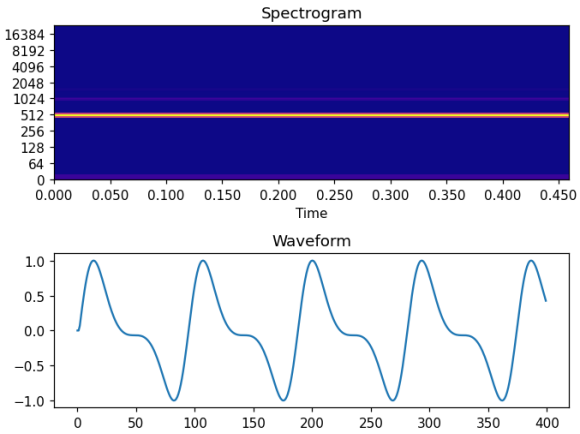
Wymaganie: zaimplementowane w ramach pracy środowisko eksperymentalne musi pozwalać na zdefiniowanie węzłów DSP, które generują lub przetwarzają sygnał. Węzły posiadają sloty wejściowe, z których czytają wartości parametrów sterujących wykonywanymi przez węzły operacjami.

5.2.2. Połączenia między węzłami – modulacja parametrów węzłów

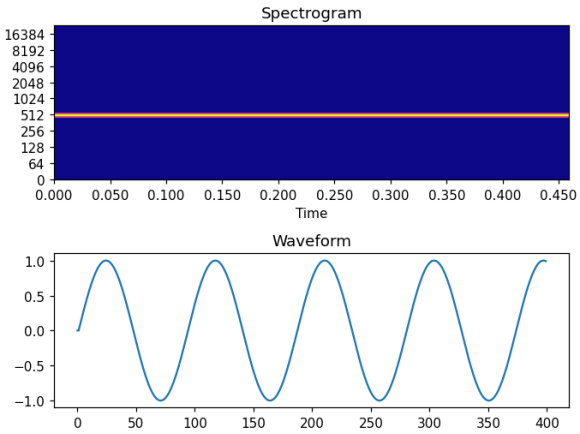
Każdy węzeł DSP w środowisku eksperymentowym posiada zbiór parametrów wejściowych. Poza możliwością ustawienia danego parametru wejściowego na konkretną wartość, możliwa jest też modulacja parametru wejściowego. Na rysunku 5.2.2 przedstawiony jest przykładowy układ węzłów i modulacji, które pozwalają na uzyskanie syntezy FM.



Rys. 5.5: Przykładowa modulacja parametru `input_modulation` za pomocą sygnału sinusoidalnego, charakterystyczna dla syntezy typu FM [29].



Rys. 5.6: Spektrogram oraz wykres sygnału wygenerowanego za pomocą układ z rysunku 5.2.2. Widoczne dodatkowe składowe harmoniczne wpływające na barwę dźwięku.



Rys. 5.7: Spektrogram oraz wykres sygnału wygenerowanego przez układ z rysunku 5.2.2 **po usunięciu** połączenia modulującego fazę oscylatora #2. Widoczna tylko jedna składowa harmoniczna: częstotliwość podstawowa.

Wymaganie: zaimplementowane środowisko pozwala na modulowanie dowolnego parametu wejściowego w węźle za pomocą wartości wyjściowej dowolnego węzła, **w tym modulowa-**

nie wejścia węzła wyjściem tego samego węzła (tzw. *circular patching*, popularny zarówno w syntezie FM jak i w układach analogowych).

5.2.3. Graf przetwarzania sygnałów

Węzły DSP oraz połączenia między nimi istnieją w ramach danego grafu przetwarzania sygnałów, który agreguje wiele węzłów i wiele połączeń. Instancja grafu DSP musi umożliwiać dynamiczną modyfikację grafu, na którą składają się następujące operacje:

1. Dodanie nowego węzła,
2. Dodanie nowego połączenia między węzłami,
3. Usunięcie węzła,
4. Usunięcie połączenia między węzłami,
5. Ustawienie i-tego parametru wejściowego danego węzła na określoną przez użytkownika wartość.

Po utworzeniu grafu, użytkownik musi mieć możliwość „uruchomienia" na grafie procesu syntezy dźwięku, który zwróci użytkownikowi strukturę danych zawierającą wygenerowany sygnał.

Wymaganie: zaimplementowane środowisko pozwala na dynamiczną modyfikację grafu przetwarzania sygnałów oraz na wygenerowanie sygnału z wytworzonego w środowisku grafu.

5.2.4. Automatyzacja pracy ze środowiskiem eksperymentowym za pośrednictwem języka `Python`

Wymaganie: ze względu na dużą dostępność gotowych algorytmów optymalizacyjnych oraz DSP w języku `Python` ([33], [39]), zaimplementowane środowisko musi udostępniać interfejs pozwalający na wykonywanie operacji zdefiniowanych w wymaganiach za pośrednictwem języka `Python`.

5.3. Opis zaimplementowanego środowiska eksperymentowego

W ramach pracy zaimplementowane zostało środowisko pozwalające na dynamiczne budowanie grafów DSP oraz na generowanie sygnałów dźwiękowych za pomocą wytworzonych grafów, według wymagań opisanych w sekcji 5.2. Środowisko zaimplementowano w języku `Rust`, dzięki czemu proces syntezy sygnałów jest szybszy niż w przypadku implementacji w języku interpretowanym. Zaimplementowana biblioteka udostępnia interfejs zgodny ze standardem *Python Extension Module* [21].

5.3.1. Przykłady użycia

Utworzenie grafu

Zaimplementowane środowisko pozwala na tworzenie grafów przetwarzania sygnałów za pomocą poleceń w języku `Python`. Listing 5.2 przedstawia proces tworzenia prostego grafu generującego sygnał sinusoidalny.

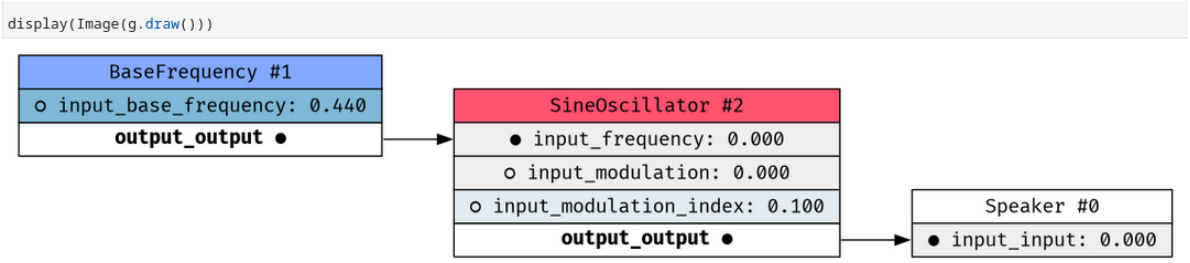
Listing 5.2: Utworzenie prostego grafu generującego sygnał sinusoidalny.

```
g = DspGraph()
```

```
carrier = g.add_sine(SineOscillator())
g.patch(
  g.base_frequency_node_id, "output_output",
  carrier, "input_frequency"
)

g.patch(
  carrier, "output_output",
  g.speaker_node_id, "input_input"
)

display(Image(g.draw()))
```



Rys. 5.8: Wynik wykonania kodu przedstawionego w listingu 5.2 w środowisku *Jupyter Notebook*, wizualizacja utworzonego grafu.

Uruchomienie procesu syntezy dźwięku

Jak pokazano na listingu 5.3, środowisko eksperymentalne zaimplementowane w ramach pracy w języku `Rust` zwraca obiekty typu `ndarray`, wykorzystywane w większości pakietów obliczeniowych wykorzystywanych w języku `Python`. Umożliwia to wykorzystanie gotowych bibliotek dostępnych w języku `Python`, aby przeanalizować sygnał lub zoptymalizować parametry syntezy [39] [33].

Listing 5.3: Typ danych zwracanych przez środowisko eksperymentalne.

```
>>> generated_signal = g.play(num_samples=100)
>>> type(generated_signal)
<class 'numpy.ndarray'>
```

5.3.2. Detale techniczne

Połączenia między węzłami w grafie

Ponieważ wymagania zdefiniowane w rozdziale 5.2 zawierają dynamiczne modyfikowanie grafu przetwarzania sygnałów, nie jest możliwe spredefiniowanie mechanizmu wymiany danych między połączonymi węzłami. Podczas implementacji rozważane były następujące architektury:

1. Przejście przez graf przed uruchomieniem syntezy i określenie kolejności wywołania węzłów,
2. Wykorzystanie struktury danych kolejki w każdym połączeniu,
3. Bezpośrednie przepisywanie wartości wyjść z węzłów do modulowanych przez nie wejść po każdej iteracji przetwarzania.

Ostatecznie wybrane zostało podejście 3, ze względu na konieczność spełnienia wymagania 5.2.2, konkretnie możliwości modulowania wejścia danego węzła przez wyjście tego samego węzła, co uniemożliwia wykorzystanie podejścia 1. Jednocześnie podejście 3 jest łatwiejsze do implementacji niż podejście 2. Implementację mechanizmu przesyłu danych między węzłami ułatwiło wykorzystanie makr proceduralnych, opisane w sekcji 5.3.2.

Zastosowanie *procedural macros* do automatycznej generacji akcesorów struktur węzłów

Podczas implementacji grafu DSP zostały wykorzystane makra proceduralne [10], które umożliwiają automatyczne zaimplementowanie metod odczytujących i-te wejście lub wyjście danego węzła. Alternatywnym podejściem byłoby wykorzystanie struktur takich jak słowniki lub mapy, które umożliwiają na inspekcję kluczy w strukturze danych podczas działania programu i dostęp do nich za pomocą indeksu, jednakże takie podejście zmniejsza wydajność programu i nie wykorzystuje wykorzystywanie systemu typów wbudowanego w język, co potencjalnie może być źródłem błędów podczas utrzymywania dużego zbioru węzłów i algorytmów, które wykonują. Wykorzystanie makr proceduralnych pozwala na ograniczenie powtarzalnych implementacji podobnych akcesorów i zachowanie zalet silnego systemu typów języka `Rust`.

Implementacja natywnego modułu dla języka `Python`

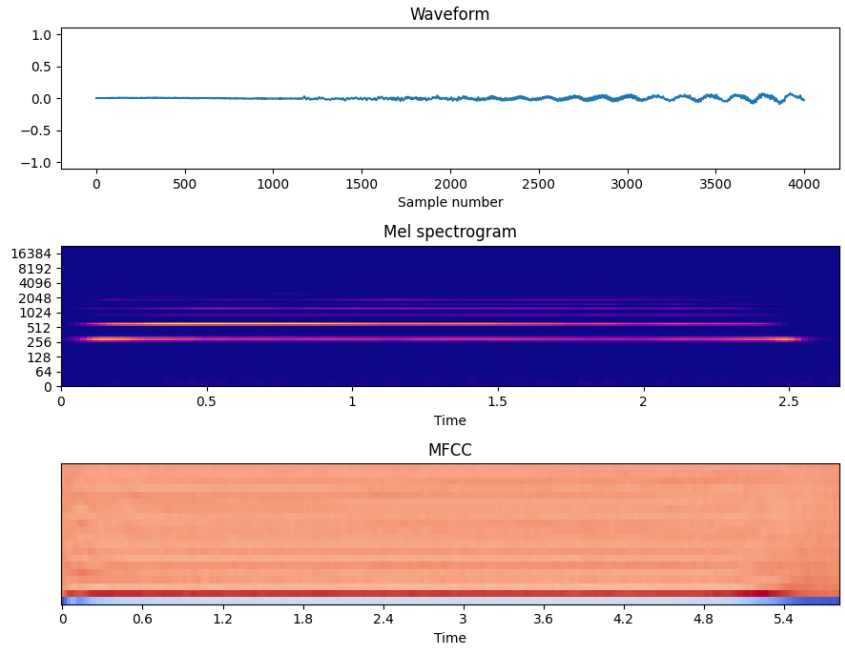
Aby umożliwić wykorzystanie grafu przetwarzania sygnałów z poziomu języka Python, wykorzystano narzędzie *Maturin* [8], służące do implementowania rozszerzeń zgodnych ze standardem *Python Extension Module* [21] w języku `Rust`.

Rozdział 6

Wyniki badań

Praca wykorzystuje próbki dźwięku z literatury [30] aby przetestować zaimplementowany algorytm i jednocześnie porównać jego wyniki z podobną pracą. Wykorzystano te próbki dźwięku z literatury, które **nie zostały wygenerowane** za pomocą *Pure Data* [9], ponieważ oryginalna praca wykorzystywała to oprogramowanie jako środowisko do syntezy dźwięku – takie porównanie nie byłoby miarodajne. Przed przeprowadzeniem optymalizacji ustalono wartość częstotliwości podstawowej f_0 dla każdego z testowanych dźwięków. Zastosowano implementację algorytmu estymującego częstotliwość podstawową *YIN* [32], dostępną w pakiecie obliczeniowym `librosa` [33].

6.0.1. Dźwięk fletu

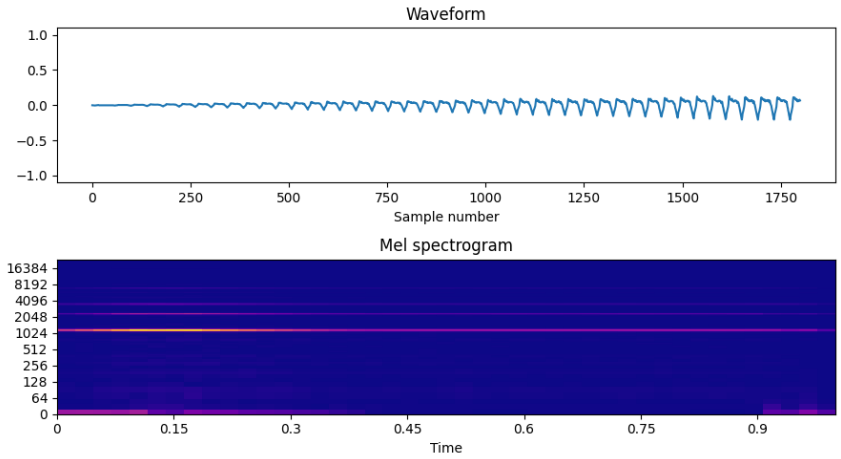


Rys. 6.1: Spektrum fouriera i współczynniki MFCC dla dźwięku `flute.wav` wykorzystywanego do eksperymentów w [30].

Dźwięk `flute_.wav` (6.1) jest nagraniem prawdziwego instrumentu dętego. Kształt fali jest nieregularny, charakterystyka spektralna zawiera dynamicznie pojawiające się i słabnące składowe częstotliwościowe.

6.0.2. Sampel z syntezatora *OP-1*

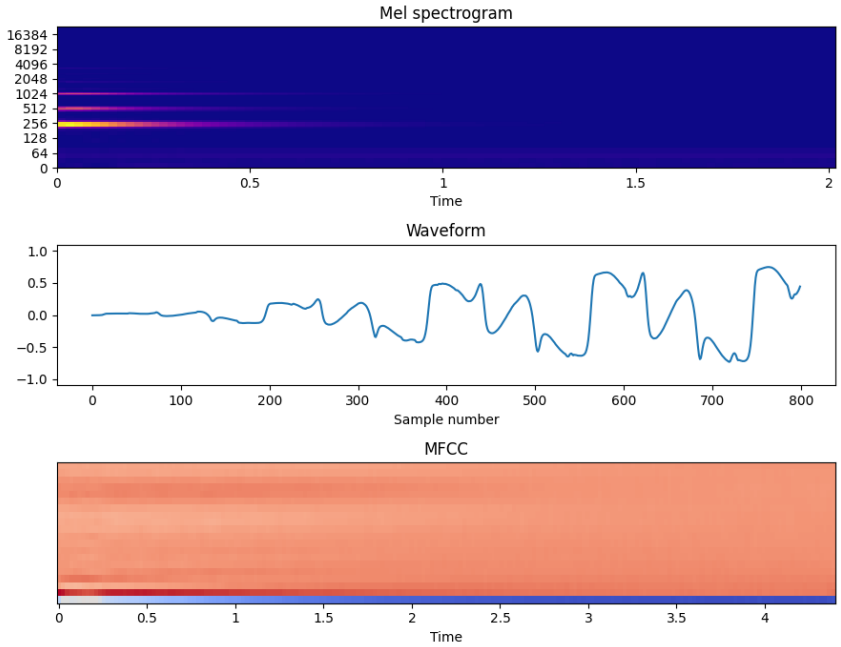
Dźwięk `op1_1.wav`, wygenerowany przy pomocy syntezatora *OP-1* jest próbą zasymulowania dźwięku instrumentu dętego.



Rys. 6.2: Spektrum fouriera i współczynniki MFCC dla dźwięku `op1_1.wav` wykorzystywanego do eksperymentów w [30].

6.0.3. Transjent

Dźwięk `transient.wav`, przedstawiony na rysunku 6.3, został wykorzystany w literaturze [31] do sprawdzenia jak dobrze algorytm generujący dźwięk potrafi przybliżyć dźwięki o dynamicznych zmianach w charakterystyce spektralnej. Tego typu dźwięki są typowe dla instrumentów takich jak fortepian lub klawesyn.



Rys. 6.3: Spektrum fouriera i współczynniki MFCC dla dźwięku `transient.wav` wykorzystywanego do eksperymentów w [30].

Rozdział 7

Analiza wyników, możliwe drogi dalszych badań

Na razie luźnym językiem

7.1. Analiza wyników

1. Fajnie że działa i jest w stanie wygenerować podobne dźwięki, dla prostych przypadków 1:1,
2. Nawet jak nie działa to idzie w kierunku celu, często generując interesujące brzmienia po drodze,
3. Problemy ze zmianami w dynamice, szczególnie pierwsze ułamki sekund,
4. Z jakiegoś powodu algorytm nie „dociąga" parametrów do końca skali.

7.2. Możliwe drogi dalszego rozwoju

1. Szybszy wariant DTF,
2. Różne wielkości okna DTF,
3. MFCC/Fourier na GPU.
4. Lepsze analizy funkcji celu - brakuje tutaj prac.
5. Integracja z softwami DAW, bo na MCU w środku zwykłego syntha raczej nie wejdzie.

Literatura

[1] Eurorack standard - wikipedia article. Dostępny na https://en.wikipedia.org/wiki/Eurorack.

[2] Ladder filter implementation. https://github.com/RustAudio/vst-rs/blob/master/examples/ladder_filter.rs.

[3] Microcosm hologram user manual. https://www.hologramelectronics.com/_files/ugd/74428b_c4e6e20555914198bdb59c12f9a9e4d4.pdf.

[4] Transient (acoustics). Dostępny na https://en.wikipedia.org/wiki/Transient_(acoustics).

[5] Vcv rack - virtual eurorack studio. https://vcvrack.com/.

[6] Yamaha dx7 user manual. https://usa.yamaha.com/files/download/other_assets/9/333979/DX7E1.pdf.

[7] Yamaha vl1 user manual. https://europe.yamaha.com/files/download/other_assets/9/321049/VL1E1.pdf.

[8] Maturin - build and publish crates with pyo3, rust-cpython, cffi and uniffi bindings as well as rust binaries as python packages., 2023. https://github.com/PyO3/maturin.

[9] Pure data - an open source visual programming language for multimedia, 2023. https://puredata.info/.

[10] The rust reference - procedural macros, 2023. https://doc.rust-lang.org/reference/procedural-macros.html.

[11] B. Bozkurt, K. A. Yüksel. Parallel evolutionary optimization of digital sound synthesis parameters. C. Di Chio, A. Brabazon, G. A. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. G. B. Tettamanzi, N. Urquhart, A. Ş. Uyar, redaktorzy, *Applications of Evolutionary Computation*, strony 194–203, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[12] F. Caspe, A. McPherson, M. Sandler. Ddx7: Differentiable fm synthesis of musical instrument sounds, 2022.

[13] R. Challinor. Bespoke synth - a modular daw for mac, windows, and linux., 2023. https://www.bespokesynth.com/.

[14] N. Collins. The analysis of generative music programs. *Organised Sound*, 13(3):237–248, 2008.

[15] J. Dattorro. Effect design 1: Reverberator and other filters. 1997.

[16] Elektron. Elektron digitone user manual, 2022. https://cdn.www.elektron.se/media/downloads/digitone/Digitone_User_Manual_ENG_OS1.40A_221123.pdf.

[17] J. Engel, L. H. Hantrakul, C. Gu, A. Roberts. Ddsp: Differentiable digital signal processing. *International Conference on Learning Representations*, 2020. https://openreview.net/forum?id=B1x1ma4tDr.

[18] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, M. Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[19] D. Faronbi, I. Roman, J. P. Bello. Exploring approaches to multi-task automatic synthesizer programming. *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, strony 1–5, 2023.

[20] S. Forsgren, H. Martiros. Riffusion - Stable diffusion for real-time music generation. 2022.

[21] P. S. Foundation. Python documentation - extending python with c or c++, 2019. https://www.korg.com/us/support/download/manual/0/811/4277/.

[22] N. Hind. *Common Lisp Music (CLM) Tutorials*. wydanie first, 2021. Available for free at https://ccrma.stanford.edu/software/clm/compmus/clm-tutorials/toc.html.

[23] E. Jacobsen, R. Lyons. The sliding dft. *IEEE Signal Processing Magazine*, 20(2):74–80, 2003.

[24] S. Kacprzak. Inteligentne metody rozpoznawania dźwięku.

[25] Y. Ke, D. Hoiem, R. Sukthankar. Computer vision for music identification. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, wolumen 1, strony 597–604 vol. 1, 2005.

[26] A. F. Khalifeh, A.-K. Al-Tamimi, K. A. Darabkh. Perceptual evaluation of audio quality under lossy networks. *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, strony 939–943, 2017.

[27] K. Kilgour, M. Zuluaga, D. Roblek, M. Sharifi. Fréchet audio distance: A metric for evaluating music enhancement algorithms, 2018. https://arxiv.org/abs/1812.08466.

[28] KORG. Korg minilogue xd user manual, 2017.

[29] S. Luke. *Computational Music Synthesis*. wydanie first, 2021. Available for free at http://cs.gmu.edu/~sean/book/synthesis/.

[30] M. Macret, P. Pasquier. Automatic design of sound synthesizers as pure data patches using coevolutionary mixed-typed cartesian genetic programming – trained sounds. https://metacreation.net/mmacret/GECCO2014.

[31] M. Macret, P. Pasquier. Automatic design of sound synthesizers as pure data patches using coevolutionary mixed-typed cartesian genetic programming. *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, strona 309–316, New York, NY, USA, 2014. Association for Computing Machinery.

[32] M. Mauch, S. Dixon. Pyin: A fundamental frequency estimator using probabilistic threshold distributions. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, strony 659–663, 2014.

[33] B. McFee, C. Raffel, D. Liang, D. Ellis, M. Mcvicar, E. Battenberg, O. Nieto. librosa: Audio and music signal analysis in python. strony 18–24, 01 2015.

[34] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[35] J. O. Smith. *Introduction to Digital Filters with Audio Applications*. W3K Publishing, http://www.w3k.org/books/, 2007.

[36] J. O. Smith. Physical signal audio processing. 2010.

[37] J. O. Smith. *Spectral Audio Signal Processing*. https://ccrma.stanford.edu/~jos/sasp/, accessed 20.04.2023. online book, 2011 edition.

[38] M. Sood, S. Jain. Speech recognition employing mfcc and dynamic time warping algorithm. P. K. Singh, Z. Polkowski, S. Tanwar, S. K. Pandey, G. Matei, D. Pirvu, redaktorzy, *Innovations in Information and Communication Technologies (IICT-2020)*, strony 235–242, Cham, 2021. Springer International Publishing.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[40] L. Zhang, C. Callison-Burch. Language models are drummers: Drum composition with natural language pre-training, 2023.

[41] F. Zheng, G. Zhang, Z. Song. Comparison of different implementations of mfcc. *Journal of Computer Science and Technology*, 16(6):582–589, Nov 2001. https://doi.org/10.1007/BF02943243.

Dodatek A

Instrukcja wdrożeniowa

Jeśli praca skończyła się wykonaniem jakiegoś oprogramowania, to w dodatku powinna pojawić się instrukcja wdrożeniowa (o tym jak skompilować/zainstalować to oprogramowanie). Przydałoby się również krótkie „*how to*” (jak uruchomić system i coś w nim zrobić – zademonstrowane na jakimś najprostszym przypadku użycia). Można z tego zrobić osobny dodatek.
`cargo run` i jazda

Dodatek B

Opis załączonej płyty CD/DVD