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, 2022

Tekst zawarty w niniejszym szablonie jest udostępniany na licencji Creative Commons: *Uznanie autorstwa – Użycie niekomercyjne – Na tych samych warunkach, 3.0 Polska*, Wrocław 2023.

Oznacza to, że wszystkie przekazane treści można kopiować i wykorzystywać do celów niekomercyjnych, a także tworzyć na ich podstawie utwory zależne pod warunkiem podania autora i nazwy licencjodawcy oraz udzielania na utwory zależne takiej samej licencji. Tekst licencji jest dostępny pod adresem: `http://creativecommons.org/licenses/by-nc-sa/3.0/pl/`.

Licencja nie dotyczy latexowego kodu szablonu. Sam szablon (tj. zbiór przygotowanych komend formatujących dokument) można wykorzystywać bez wzmiankowania o jego autorze. Dlatego podczas redakcji pracy dyplomowej niniejszą stronę można usunąć.

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

TODO!

Keywords: synthesis, sound, audio, graph, optimisation

Spis treści

Spis rysunków

Spis tabel

Spis listingów

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

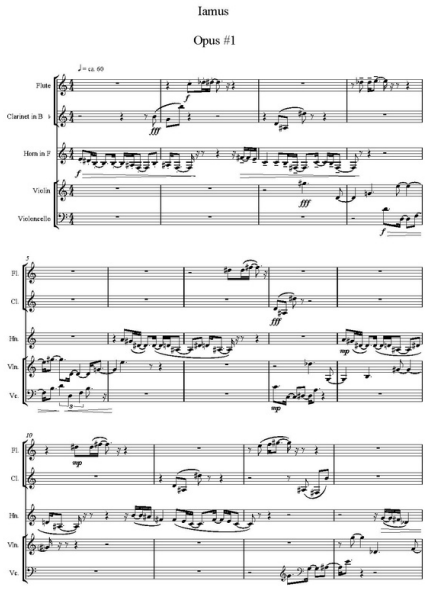
Wstęp

1.1. Wprowadzenie

Rozpowszechnione algorytmy sztucznej inteligencji wspomagające pracę inżynierów dźwięku można podzielić na trzy główne kategorie [**?**] :

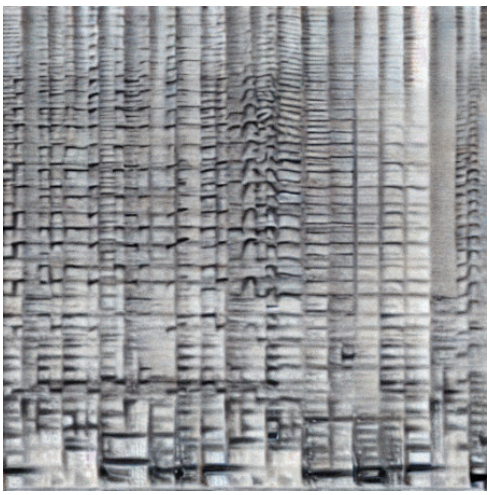
1. algorytmy generujące symboliczny zapis muzyki (nuty lub dane MIDI) (1.1), [**?**],
2. algorytmy generujące gotowy plik audio (1.2).
3. algorytmy symulujące brzemienie instrumentów muzycznych za pomocą sieci neuronowych [**?**].

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, rozwija się na bazie nowych możliwości, które zapewniają algorytmy wywodzące się ze *Stable Diffusion* [**?**]. Najnowsze 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. Jednakże zamiast na obrazach artystów, modele takie jak *Stable Riffusion* [**?**] uczą się na spektrogramach, które następnie są w stanie wygenerować (1.2). Po wygenerowaniu spektrogramu przez model, jest on konwertowany do pliku audio 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`.

Obie metody opisane w rozdziale 1.1 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 symbo-

liczny 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 dwóch wyżej wymienionych (1.1) głównych 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 grafu 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 *patch bay*, pozwalający na modyfikowanie grafu przepływu sygnałów wewnątrz syntezatora, bądź połączenie go z zewnętrznym sprzętem muzycznym bądź elektronicznym (1.3).

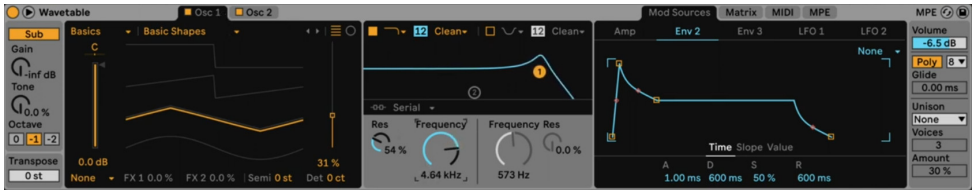
Rys. 1.3: Syntezator *Mother 32* firmy *Moog*, po prawej stronie widoczny jest *patch bay* z podłączonymi przewodami, które zmieniają konfigurację połączeń między układami generującymi i przetwarzającymi sygnał dźwiękowy.

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



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

1.2.1. Generowanie grafu przetwarzania sygnałów

Pierwsze zagadnienie sprowadza się do przetestowania szeregu algorytmów pozwalających na wygenerowanie grafu przetwarzania sygnałów DSP oraz ich modyfikację. Przykładem modyfikacji grafu może być wprowadzanie w nim losowych zmian lub krzyżowanie dwóch grafów DSP w przypadku wykorzystania algorytmu genetycznego. Graf przetwarzania sygnałów można opisać jako zbiór połączonych węzłów generujących i przetwarzających sygnał dźwiękowy. Każdy węzeł można opisać poprzez:

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

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

N - liczba węzłów,

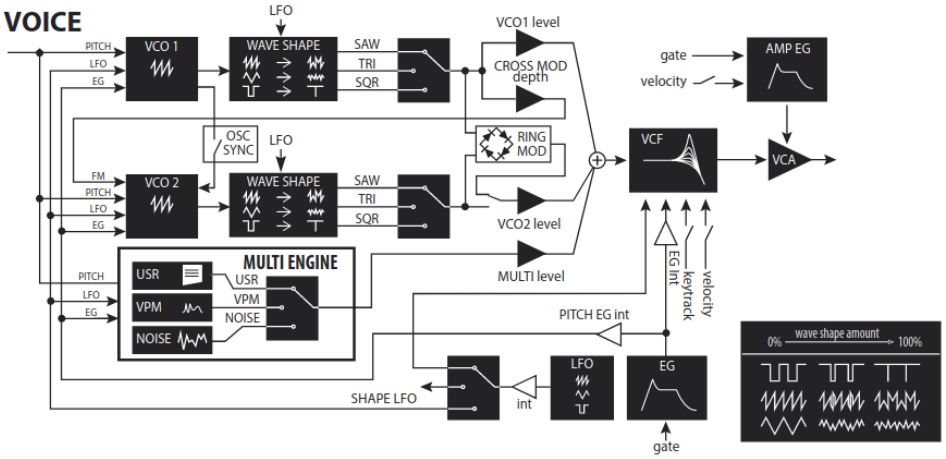
$i_j = [p_1, p_2, \ldots, p_n]$ – Zbiór wejść (*inputs*) j-go węzła,

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

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

$C = [\{o_{(j,k)}, i_{(l,m)}\}, \ldots]$ – zbiór połączeń między węzłami, opisujący, które k-te wyjście j-go węzła podłączone jest do którego m-go wejścia l-go węzła.

Nie wszystkie wejścia w grafie muszą być podłączone do któregoś z wyjść. Wejście, które nie zostało nigdzie podłączone przyjmuje jako wartość parametr liczbowy optymalizowany później w funkcji celu 1.1. W przypadku schematu 1.5 takimi „wolnymi” wejściami są przykładowo sygnał określający częstotliwość odcięcia filtru sygnału lub parametry określające parametry generatora obwiedni (*EG*).



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

Dla powszechnie wykorzystywanego w analogowych syntezatorach subtraktywnych schematu przetwarzania sygnałów (1.5) można wyróżnić przykładowe węzły:

Oscylator (*VCO*):

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

Filtr (*VCF*):

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

1.2.2. Funkcja celu oceniająca podobieństwo barwy dźwieku

Drugie zagadnienie obejmuje przetestowanie szeregu algorytmów, które można wykorzystać jako funkcję celu, która będzie optymalizowana poprzez „dostrajanie" grafu przetwarzania sygnałów dźwiękowych.

Funkcja celu, oceniająca, jak sygnał wygenerowany ($\bar{x}$) przez algorytm jest bliski sygnałowi docelowemu (x) może zostać przedstawiona w następujący sposób:

$$F(x, \bar{x}) = q \tag{1.1}$$

Gdzie q oznacza liczbę rzeczywistą, która jest tym mniejsza, im bardziej zbliżone do siebie są barwy dźwięku sygnałów x i $\bar{x}$, przy założeniu że oba sygnały przedstawiają dźwięk o tej samej częstotliwości podstawowej.

1.2.3. Problem optymalizacyjny

Następnie, dla danego układu N węzłów przetwarzania oraz dla macierzy połączeń C, należy rozwiązać następujący problem optymalizacji, należy rozwiązać problem maksymalizacji funkcji opisanej równaniem 1.1 dla parametrów wszystkich wejść i_j oraz o_j, które nie są połączone bezpośrednio pomiędzy węzłami.

1.3. Zakres pracy, plan badań

1.3.1. 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 (lub nie zawiera ich wcale, co można założyć dla uproszczenia problemu). 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 [**?**] 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 [**?**] [**?**],
2. synteza subtraktywna [**?**] [**?**],
3. algorytmy *physical modeling* [**?**] [**?**],
4. symulacja efektu pogłosu/echa [**?**] [**?**].

1.3.2. 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 [**?**] [**?**],
2. techniki wykorzystywane do generowania „cyfrowych podpisów" sygnałów dźwiękowych (*sound fingerprinting*) [**?**],
3. algorytmy wykrywające spadek jakości dźwięku z perspektywy psychoakustycznej [**?**] [**?**].

1.4. Struktura i zawartość pracy

Praca podzielona jest na następujące części:

Środowisko eksperymentalne - graf przetwarzania sygnałów (2)

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

Analiza możliwych funkcji celu (??)

Porównuje funkcje z dziedziny przetwarzania sygnałów, które pozwalają określić jak podobna jest barwa dźwięku dwóch sygnałów dźwiękowych.

Optymalizacja struktury grafu DSP oraz jego parametrów (??)

Opisuje proces badawczy, w którym narzędzia wytworzone w rozdziałach 2 oraz **??** zostały wykorzystane do automatycznego wytworzenia grafu DSP, który naśladuje barwę zadanej próbki dźwięku.

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

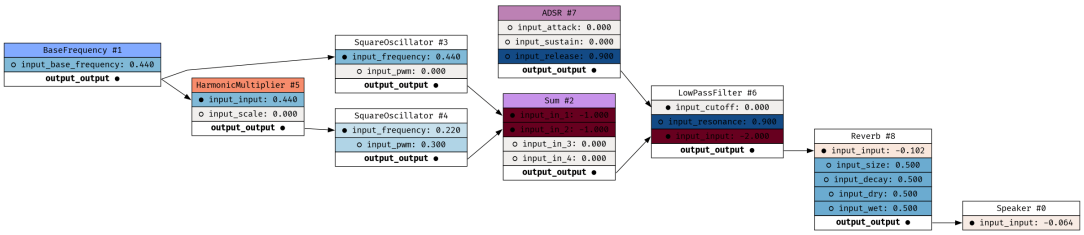
Rozdział podsumowuje uzyskane wyniki badań, podejmuje dyskusję nad ogólną skutecznością i przydatnością zaimplementowanego rozwiązania oraz kreśli potencjalne drogi dalszego rozwoju prac badawczych w podobnej tematyce. W czasie, gdy niniejsza praca była tworzona, zostały opublikowane badania dotyczące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

Rozdział 2

Graf przetwarzania sygnałów

Na potrzeby badań zostało zaimplementowane środowisko, pozwalające na dynamiczne tworzenie grafów przetwarzania sygnałów (2.2). W projekcie nie zostało zastosowane gotowe rozwiązanie symulujące syntezator modułowy, takie jak Bespoke Synth [**?**], VCVRack [**?**] lub Pure Data [**?**], 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. 2.1: Przykładowy układ modułów w standardzie *Eurorack* [**?**]. W prawym dolnym rogu widoczne połączenia modulujące między modułami.



Rys. 2.2: Przykładowy układ węzłów DSP w zaimplementowanym środowisku eksperymentowym. Układ wykonuje syntezę subtraktywną z modulowaną wartością częstotliwości granicznej filtru niskoprzepustowego oraz dodaje efekt pogłosu (*reverb*) [**?**].

2.1. 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 2.1), 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.

Dla przykładowego układu DSP, przedstawionego na rysunku 2.2, 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` modulowana jest przez sygnał wychodzący w węźle `ADSR #7`, co pozwala na dynamiczne zmiany częstotliwości odcięcia filtru w czasie, wzbogacając barwę generowanego dźwięku.

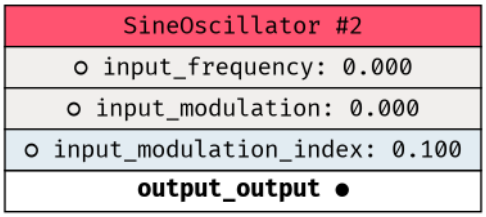
2.2. Wymagania

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

2.2.1. 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. 2.3: Węzeł DSP w zaimplementowanym środowisku eksperymentowym, generujący falę sinusoidalną z możliwością modulacji fazy.

Rys. 2.4: Moduł syntezy *Mutable Instruments Elements* umożliwiający modulację parametrów syntezy typu *physical modeling* [**?**].

Przykładowo, przedstawiony na rysunku **??** węzeł posiada:

1. sygnały wejściowe:
   - `input_frequency` - częstotliwość generowanej sinusoidy,
   - `input_modulation` - wartość modulacji fazy, według równania **??**,
   - `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 **??** oraz listingu **??**.

$$f(t, f, m, m_i) = sin(t * f + m * m_i) \tag{2.1}$$

Listing 2.1: Implementacja węzła SineOscillator.

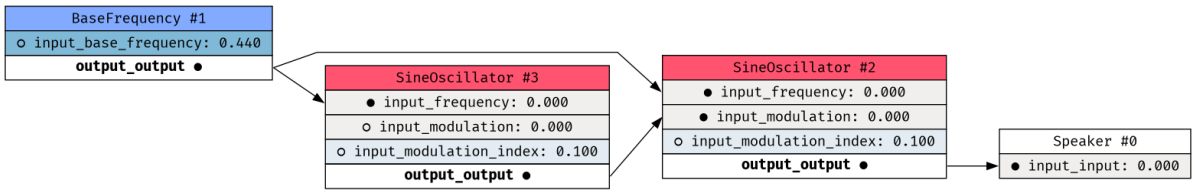
```
impl DspNode for SineOscillator {
    fn tick(&mut self) {
        let frequency = (self.input_frequency * 1000.0).abs();
        let phase_diff = (2.0 * std::f64::consts::PI * frequency) /
            ↪ SAMPLE_RATE;
        self.output_output =
            (self.phase + self.input_modulation * self.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

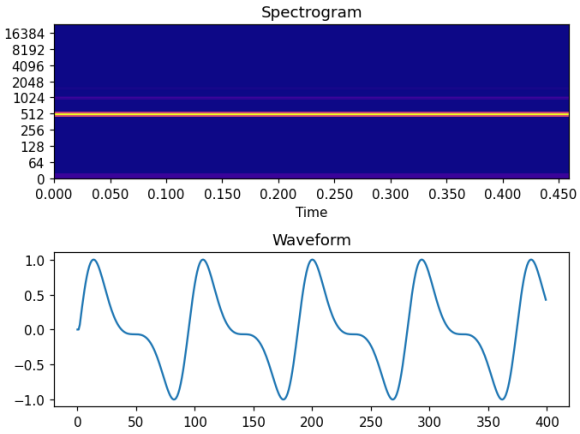
Wymaganie: zaimplementowane w ramach pracy środowisko eksperymentalne musi pozwalać na zdefiniowanie węzłów DSP, które generują lub przetwarzają sygnał. Węzły posiadają sloty wejściowe, z których czytają wartości parametrów sterujących wykonywanymi przez węzły operacjami.

2.2.2. Połączenia między węzłami – modulacja parametrów węzłów

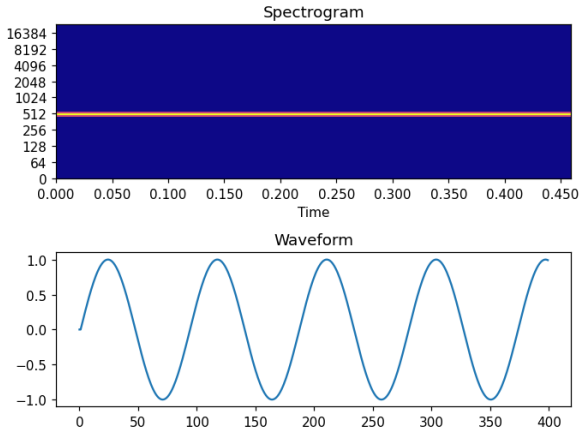
Każdy węzeł DSP w środowisku eksperymentowym posiada zbiór parametrów wejściowych. Poza możliwością ustawienia danego parametru wejściowego na konkretną wartość, możliwa jest też modulacja parametru wejściowego. Na rysunku **??** przedstawiony jest przykładowy układ węzłów i modulacji, które pozwalają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



Rys. 2.5: Przykładowa modulacja parametru `input_modulation` za pomocy sygnału sinusoidalnego, charakterystyczna dla syntezy typu FM [**?**].



Rys. 2.6: Spektrogram oraz wykres sygnału wygenerowanego za pomocą układ z rysunku **??**. Widoczne dodatkowe składowe harmoniczne wpływające na barwę dźwięku.



Rys. 2.7: Spektrogram oraz wykres sygnału wygenerowanego przez układ z rysunku **?? 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 modulowanie wejścia węzła wyjściem tego samego węzła** (tzw. *circular patching*, popularny zarówno w syntezie FM jak i w układach analogowych).

2.2.3. 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.

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

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

2.3. 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 2.2. 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* [**?**].

2.3.1. Przykłady użycia

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

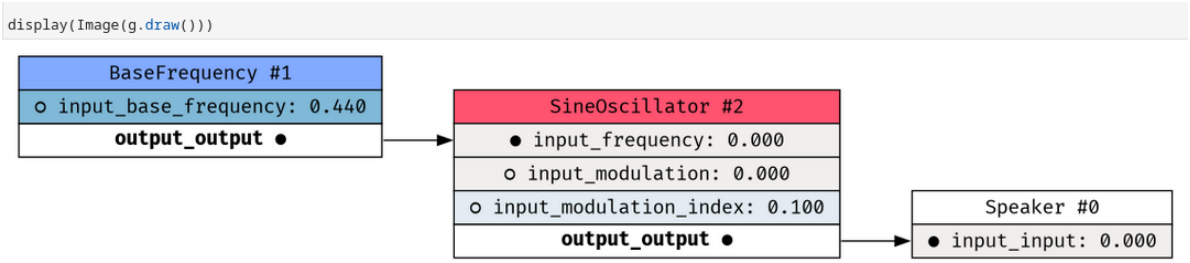
Listing 2.2: Utworzenie prostego grafu generującego sygnał sinusoidalny.

```
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. 2.8: Wynik wykonania kodu przedstawionego w listingu **??** w środowisku *Jupyter Notebook*, wizualizacja utworzonego grafu.

Jak pokazano na listingu **??**, 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 [**?**] [**?**].

Listing 2.3: Typ danych zwracanych przez środowisko eksperymentalne.

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

Rozdział 3

Funkcja celu – porównanie barwy dźwięku

Rozdział 4

Optymalizacja struktury grafu DSP oraz jego parametrów

Rozdział 5

Analiza wyników, możliwe drogi dalszego rozwoju

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

Tutaj jest miejsce na zamieszczenie opisu zawartości załączonej płyty. Opis ten jest redagowany przed załadowaniem pracy do systemy APD USOS, a więc w chwili, gdy nieznana jest jeszcze nazwa, jaką system ten wygeneruje dla załadowanego pliku. Dlatego też redagując treść tego dodatku dobrze jest stosować ogólniki typu: „Na płycie zamieszczono dokument `pdf` z niniejszej tekstem pracy" – bez wskazywania nazwy tego pliku.

Dawniej obowiązywała reguła, by nazywać dokumenty według wzorca `W04_[nr albumu]_[rok kalendarzowy]_[rodzaj pracy]`, gdzie `rok kalendarzowy` odnosił się do roku realizacji kursu „Praca dyplomowa", a nie roku obrony. Przykładowo wzorzec nazwy dla pracy dyplomowej inżynierskiej w konkretnym przypadku wyglądał tak: `W04_123456_2015_praca inżynierska.pdf`, Takie nazwy utrwalane były w systemie składania prac dyplomowych. Obecnie działa to już inaczej.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.