NI–VMM – podobnost MIDI souborů

Bc. Vojtěch Kopecký

30. listopadu 2022

Obsah

1 Úvod

Tato zpráva slouží jako dokumentace semestrálního projektu pro předmět NI–VMM. Zadáním projektu byla tvorba webové aplikace pro vyhledávání hudebních skladeb ve formátu MIDI na základě podobnosti melodií.

V této zprávě nejdříve blíže popíši vlastnosti projektu. Dále prozkoumám způsoby extrakce melodií a jejich porovnávání. V implementační části se zaměřím na zvolení vhodných technologií, práci s MIDI soubory (tím pádem také stručně popíši standard MIDI) a proces tvorby jak backendové, tak frontendové části aplikace. Předvedu příklady výstupů hotové aplikace a provedu základní měření rychlosti vyhledávání. Na závěr zhodnotím dosažené výsledky a navrhnu možná vylepšení aplikace.

2 Popis projektu

Zaměřil jsem se na klasické klavírní skladby – díky tomu, že se ve skladbě hraje jenom na jeden nástroj, je určení hlavní melodie značně jednodušší. Kromě toho jsou MIDI soubory pro klasickou hudbu na internetu velmi dostupné.

Vstupem vyhledávání je kratší soubor ve formátu MIDI (cca 4–40 po sobě jdoucích not), který obsahuje nějakou melodii. Aplikace se následně tuto melodii pokusí najít ve všech skladbách v databázi. Výstupem je seznam skladeb, které obsahují melodii nejvíce se podobající vstupní melodii. Tento seznam je setříděn od nejlepších (nejvíce podobných) výsledků po nejhorší.

Databáze obsahuje 295 klasických skladeb volně dostupných ze stránky `http://www.piano-midi.de/`.

Implementace projektu spočívala v několika krocích:

- Extrakce melodie z MIDI souborů a datová reprezentace melodie v databázi
- Porovnávání dvou melodií (určení jejich podobnosti)
- Vyhledávání nejpodobnější melodie v celé databázi
- Uživatelské rozhraní přizpůsobené pro práci s hudbou (např. umožňuje přehrávání, vizualizaci melodií…)

3 Způsoby řešení

3.1 Hledání hlavní melodie ve skladbě

Sekvence výšek

Za hlavní melodii můžeme považovat sekvenci tónů, které jsou v daný čas skladby nejvyšší (tzv. výšky). Výhodou je jak lehká implementace extrakce, tak dostatečná přesnost – nejvyšší tóny většinou představují hlavní melodii, zvlášť v klavírních skladbách. Jsou však případy, kdy se hlavní melodie skrývá v basech (nejnižších tónech) a výšky jsou pouze komplementární.

Sekvence nejhlasitějších tónů

Dále můžeme za hlavní melodii považovat sekvenci tónů, které jsou v daný čas skladby nejhlasitější. Pro toto hledání nám už ale nestačí pouze notový zápis, ale nejlépe nějaký audio soubor – např. MP3, ve kterém bychom mohli analyzovat spektogram, nebo již zmíněný formát MIDI, ve kterém je hlasitost určitého tónu určena proměnnou velocity. Lze však usoudit, že i touto metodou bychom nedosáhli vysoce přesných výsledků – např. již kvůli tomu, že některé tóny mohou mít ve stejnou dobu stejnou hlasitost – poté bychom se museli dále rozhodovat podle výšky tónu či jiných vlastností.

Problémem je také fakt, že objektivně určit hlavní melodii nelze – záleží na subjektivní interpretaci dané skladby, což také vyžaduje určitou hudební znalost. I samotný uživatel, který melodii bude vyhledávat, si hlavní melodii může vyložit

jinak, než je obecně známo. U výše zmíněných metod proto nelze jednoduše ověřit, jak moc úspěšné jsou. Budeme tedy pracovat s domněnkou, že úspěšnost obou metod je dostatečná pro potřeby naší aplikace.

3.2 Podobnostní algoritmy

V obou metodách zmíněných v předchozí kapitole byly melodie reprezentovány sekvencí. To nám značně zjednodušuje situaci, jelikož vyhledávání podobných skladeb bude spočívat v porovnávání dvou sekvencí. Na tento problém existuje mnoho algoritmů, které jsou výpočetně rychlé a také jednoduché na implementaci.

Longest common subsequence (LCS)

LCS je problém nálezu nejdelší společné podposloupnosti dvou sekvencí. Příbuzným problémem je *Longest common substring* – neboli nález nejdelšího společného podslova. Slovo se od posloupnosti liší tím, že v obou sekvencích nesmí být přerušeno, zatímco společná posloupnost může přeskakovat prvky.

Při hledání podobných melodií je LCS užitečná – jedná se o jeden ze základních způsobů, jak určit podobnost sekvencí. LCS je navíc odolná proti malým změnám v melodii (např. jedna nota vložená navíc, či chybějící nota).

Dynamic time warping (DTW)

DTW je problém měření podobnosti dvou sekvencí, které se mohou lišit v rychlosti. Na první pohled tato metrika nemusí být pro hledání podobných melodií příliš užitečná, jelikož v extrahované melodii neuvádíme délku jednotlivých not, nýbrž pouze jejich tón. I přes to však DTW představuje sofistikovanější přístup k podobnosti melodií, než LCS, jelikož pracuje s odchylkou jednotlivých členů sekvencí a nezabývá se pouze jejich rovností – DTW tak může objevit podobnost dvou melodií, kde druhá melodie představuje první melodii posunutou např. o jeden půltón výše.

Jelikož mi jak LCS, tak DTW připadaly jako užitečné algoritmy k hledání podobnosti melodií, implementoval jsem je oba v mé aplikaci. Aplikace tedy bude sloužit také jako ukázka, v jakých případech si který algoritmus vede lépe. Efektivní algoritmy LCS a DTW využívají dynamického programování se složitostí $O(mn)$, kde m a n značí délky porovnávaných sekvencí.

4 Implementace

4.1 Volba technologií a struktura projektu

Aplikaci jsem rozdělil na backend a frontend. Backend bude zajišťovat veškerou „vědeckou“ práci, tedy extrakci melodie, správu databáze skladeb a samotné vyhledávání podobnosti. Backend tyto služby poskytne frontendu pomocí REST API. Na frontendu bude poté implementováno uživatelské rozhraní pro vyhledávání.

Pro backend se přirozeně nabízel jazyk Python, který disponuje mnoha knihovnami pro vědeckou práci, zároveň se dalo jednoduše experimentovat pomocí Jupyter notebooku, než všechno implementovat na ostro. Nejdříve jsem se však snažil najít

Python knihovnu pro práci s MIDI soubory, abych extrakci nemusel implementovat ve frontendu (zde se nabízela Javascriptová knihovna *Tone.js*, která je schopna MIDI soubory konvertovat na JSON). Pro Python jsem našel knihovnu *Mido*, která k MIDI souborům přistupuje více low-level způsobem. I přes tento fakt jsem si vybral Python s tím, že knihovnu *Mido* vyzkouším. Jako web framework jsem zvolil *Django*, který se postará o samotné API a jakoukoliv práci s databází.

Pro frontend jsem zvolil Javascriptový framework *Vue 3* s grafickou knihovnou *Vuetify*.

4.2 Extrakce melodie

Nejdříve bylo potřeba extrahovat melodii z MIDI souborů. MIDI soubory se mohou skládat z několika stop (tracks). Každá stopa se pak skládá z tzv. zpráv (messages). Existují různé druhy zpráv s různými parametry (standard je dobře popsaný např. na stránce [1]) S knihovnou *Mido* [2] můžeme MIDI soubor jednoduše načíst a vypsat všechny zprávy z nějaké stopy:

```python
from mido import MidiFile

# Load midi file
midi = MidiFile("../midi/piano_midi_de/bach/bach_846.mid")

# Print all messages in the second track
for message in midi.tracks[1]:
    print(message)
```

```
MetaMessage('midi_port', port=0, time=0)
MetaMessage('track_name', name='Piano right', time=0)
program_change channel=0 program=0 time=0
control_change channel=0 control=7 value=100 time=0
control_change channel=0 control=10 value=64 time=0
control_change channel=0 control=91 value=127 time=0
MetaMessage('text', text='bdca426d104a26ac9dcb070447587523', time=0)
note_on channel=0 note=67 velocity=56 time=241
note_on channel=0 note=67 velocity=0 time=120
note_on channel=0 note=72 velocity=60 time=0
note_on channel=0 note=72 velocity=0 time=120
note_on channel=0 note=76 velocity=63 time=0
note_on channel=0 note=76 velocity=0 time=108
...
note_on channel=0 note=72 velocity=0 time=0
MetaMessage('end_of_track', time=0)
```

Na výstupu kódu lze vidět, že některé zprávy nastavují parametry skladby (MetaMessage, program_change, control_change) a některé zprávy určují noty, které se mají přehrávat (note_on).

Každá zpráva má dále atribut time, který určuje odchylku této zprávy od předchozí v tickích. Pokud má zpráva hodnotu time nastavenou na 0, zpráva se spustí ve stejnou dobu, jako ta předchozí. Seznam zpráv je tedy vypsán sekvenčně s tím, že každá zpráva nenese absolutní časovou hodnotu, ale pouze odchylku od té předchozí zprávy.

Ve zprávě note_on se výška noty určuje atributem note. Na stránkách standardu MIDI [1] lze nalézt tabulku pro zjištění, kterou notu dané číslo reprezentuje. Atribut velocity určuje hlasitost dané noty.

Ve výpisu si můžete všimnout, že nota se na nějakou dobu zapne nastavením velocity na nějakou kladnou hodnotu, a posléze se nota vypne opět zprávou note_on s velocity nastavenou na 0. Tento fakt je matoucí, jelikož existuje i zpráva note_off, kterou se noty také dají vypínat. Existují tedy dva různé způsoby vypínání not, na což je potřeba brát ohled při zpracovávání MIDI souborů z různých zdrojů. Častěji se na vypínání not používá note_on, avšak jsem zjistil, že např. program pro tvorbu hudby *FL Studio* používá note_off.

Dalším problémem, se kterým jsem se při extrakci potýkal, byl ten, že každá zpráva (i MetaMessages, které nemají mít s melodií nic společného) mohou mít nastavenou časovou odchylku time na větší než 0. Při extrakci více stop najednou pak nastaly problémy, jelikož jsem hodnotu time kontroloval pouze u zpráv note_on.

Práce s touto knihovnou tedy byla zapeklitější, avšak mi umožnila lépe poznat formát MIDI zevnitř a dala mi větší kontrolu při extrakci melodie. Na samotnou extrakci jsem použil metodu výšek.

4.3 Vyhledávání podobných melodií

V backendu jsem implementoval automatickou extrakci melodií ze všech skladeb ze stránky piano-midi.de do databáze. LCS a DTW jsem implementoval dynamickým programováním. Při vyhledávání se dotaz porovná se všemi skladbami v databázi, přičemž se každá skladba rozdělí na segmenty stejné délky, jakou má dotaz.

Segmentaci skladeb jsem nejdříve zkoušel implementovat metodou „sliding window“ – segmenty mají opět stejné délky, jako dotaz, akorát např. druhý segment nezačíná na konci prvního segmentu, nýbrž druhou notou celé skladby. Tímto způsobem lze najít potencionálně lepší výsledky, avšak je výpočetně složitější, než jednoduše rozdělit skladbu na na sebe navazující segmenty – při jednoduchém postupu je složitost porovnání dotazu (m) se všemi segmenty skladby ($n_1, n_2, \ldots$) $O(mn_1 + mn_2 + \ldots) = O(m(n_1 + n_2 + \ldots)) = O(mn)$, kdežto metoda sliding window má složitost $O(m^2(n-m))$. Kvůli větší složitosti jsem od této metody musel upustit, jelikož vyhledávání trvalo příliš dlouho.

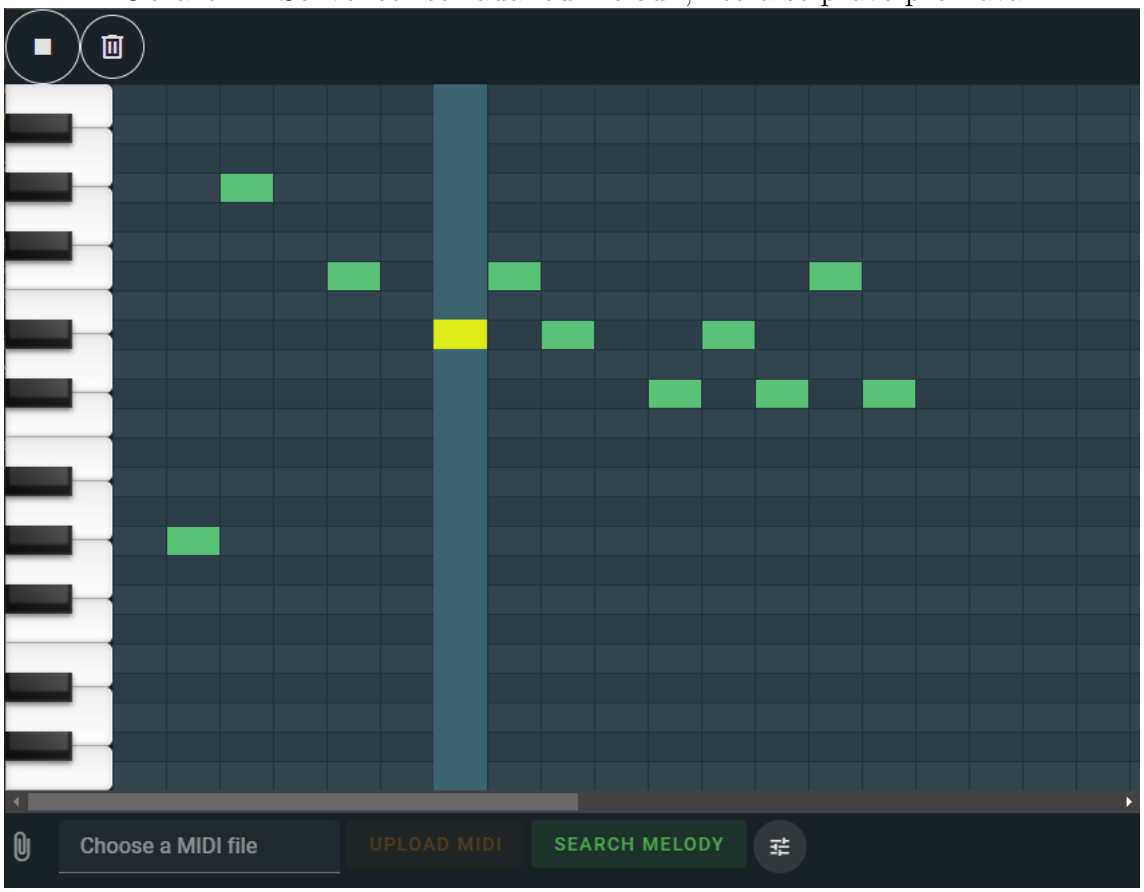
Každá skladba se tedy rozdělí na na sebe navazující segmenty stejné délky, jako dotaz. V každé skladbě algoritmus vyhledá segment s nejlepší hodnotou LCS/DTW. Algoritmus pak vrátí seznam skladeb seřazený podle nejpodobnějšího segmentu.

4.4 Uživatelské rozhraní

Pro frontend jsem si vybral Javascriptový framework Vue, který si zakládá na rozložení UI do menších celků (tzv. komponent) a reaktivitě – prvky na stránce automaticky reagují na změnu vnitřních dat. Díky těmto vlastnostem jsem mohl jednoduše sestavit tzv. sekvencer pro zadávání vstupní melodie pro vyhledávání. Sekvencer sestává z ovládacích prvků (pro přehrání či vyčištění zadané melodie) a z mřížky pro zadávání melodie. Díky Vue dokážu reprezentovat zadanou melodii polem, na jehož změnu celý sekvencer reaguje. To také zjednodušuje implementaci několika operací – např. vyčištění zadané melodie = nastavení všech prvků pole na počáteční hodnotu.

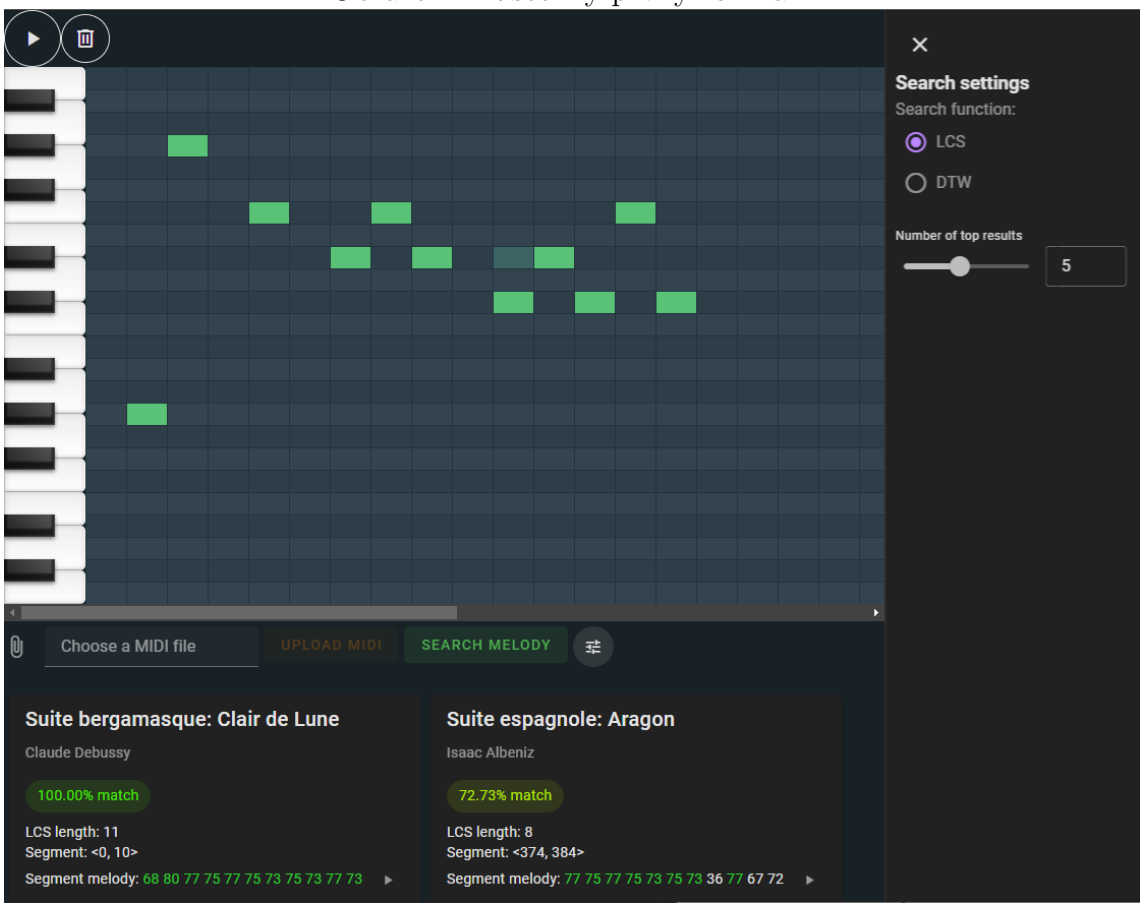
Díky sekvenceru si uživatel svůj dotaz může přehrát a dále ladit už před tím, než jej vyhledá. Další výhoda spočívá v tom, pokud uživatel chce nahrát jako dotaz MIDI soubor. Místo toho, aby se po nahrání souboru hned spustilo vyhledávání, se z něj nejdříve vyextrahuje melodie, která se následně zobrazí v sekvenceru. Uživatel tak může zkontrolovat, jestli se MIDI soubor zpracovává správně a případně může melodii opět ladit bez potřeby nějakého externího programu.

Obrázek 1: Sekvencer se zadanou melodií, která se právě přehrává



V pravé části rozhraní se nachází panel s nastaveními vyhledávání – zvolení funkce LCS/DTW a počet vrácených nejlepších výsledků. V dolní části rozhraní se po dokončení vyhledávání zobrazí seřazené výsledky.

Obrázek 2: Všechny prvky rozhraní



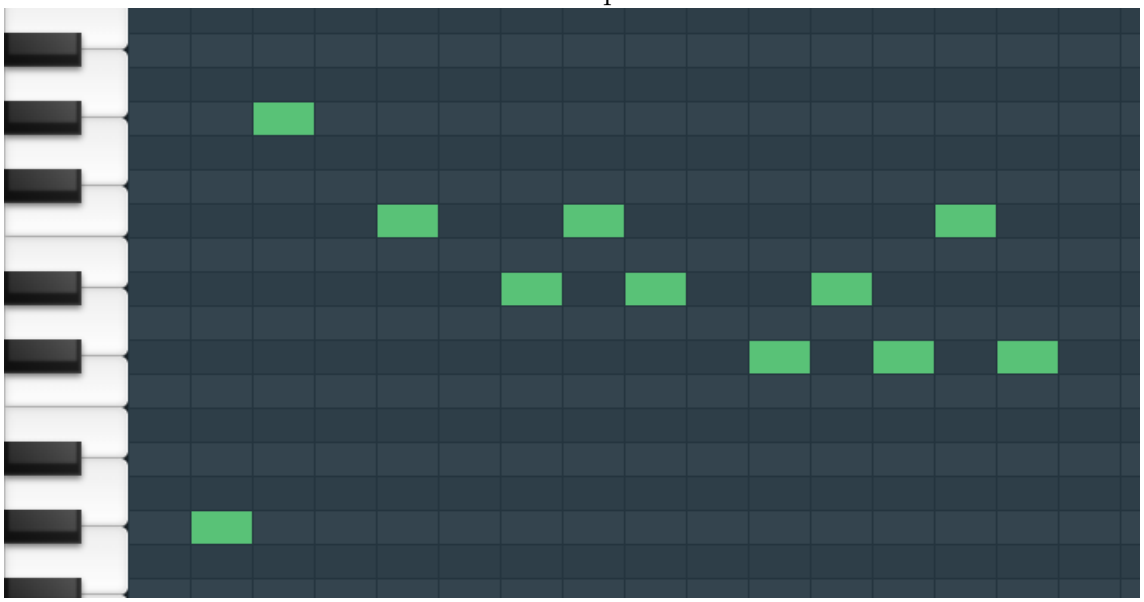
5 Příklady výstupu

Vzhled výsledku se liší podle zvoleného vyhledávacího algoritmu. Pro oba dva algoritmy se zobrazí název skladby, autor, metrika algoritmu (délka nejdelší podsekvence pro LCS, vzdálenost pro DTW), pozice segmentu v skladbě a výpis melodie segmentu. U výpisu melodie je tlačítko pro její přehrání.

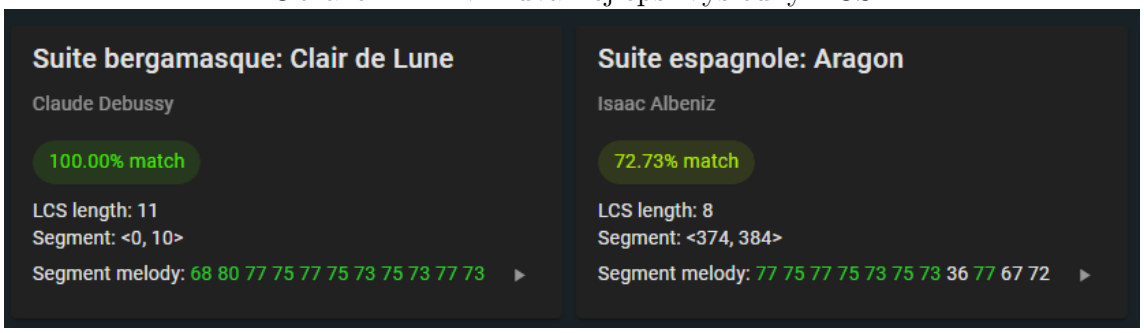
Pro LCS se navíc ve výpisu melodie segmentu zvýrazní zeleně noty, které patří do nalezené největší společné podsekvence.

Pro DTW se zobrazí graf porovnávající dotaz a segment. Zobrazuje se v něm navíc mapování členů sekvencí. Grafy se generují v backendu při zpracování dotazu pomocí knihovny *Matplotlib* a po jejich vygenerování jsou podávány frontendu jako obrázky.

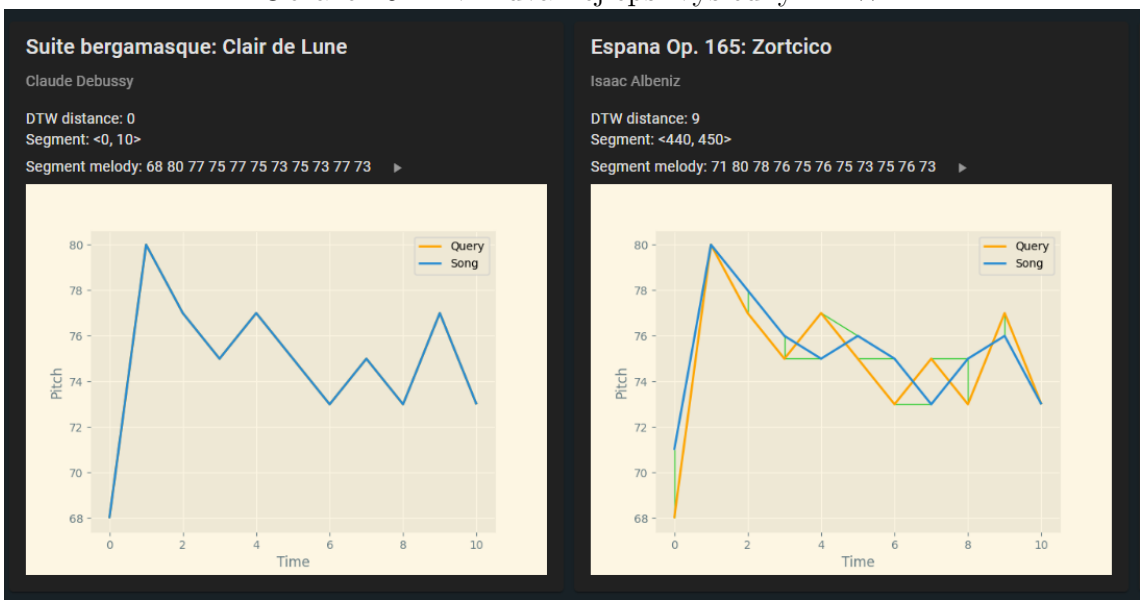
Obrázek 3: Vstupní melodie



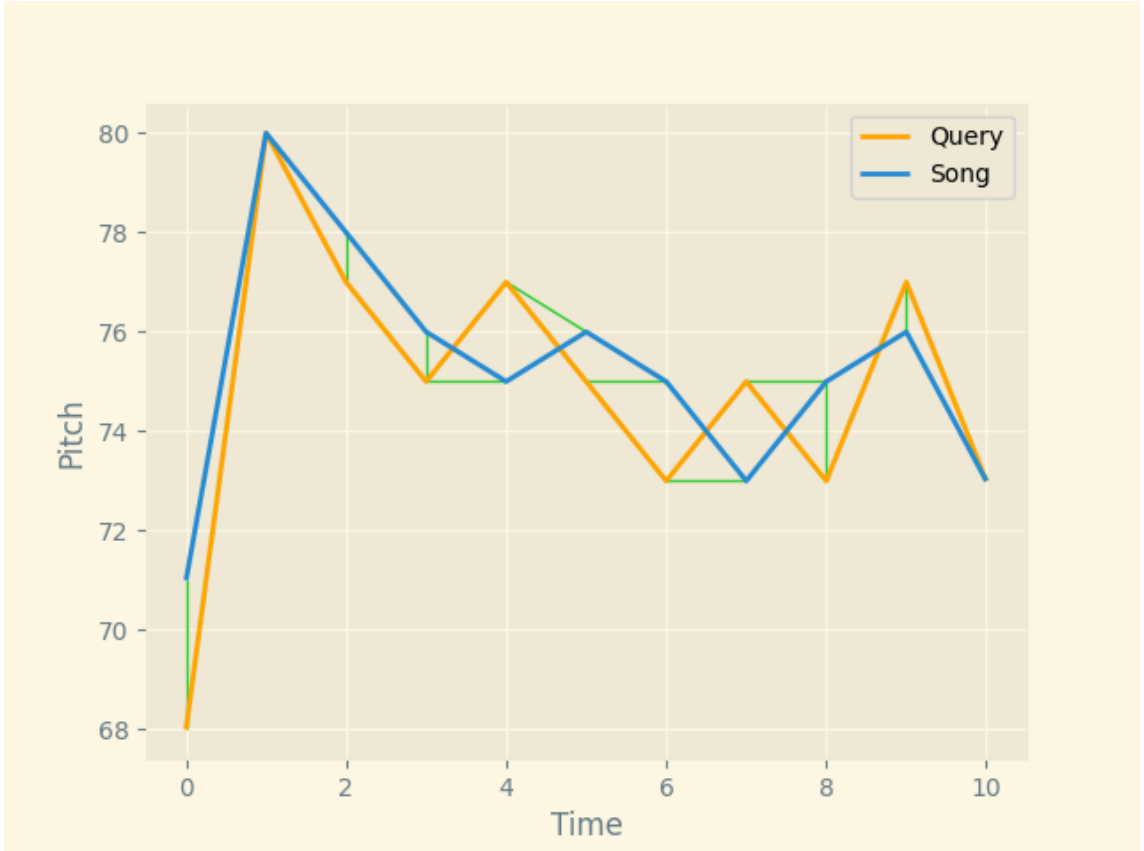
Obrázek 4: První dva nejlepší výsledky LCS



Obrázek 5: První dva nejlepší výsledky DTW



Obrázek 6: Detail grafu DTW pro druhý výsledek



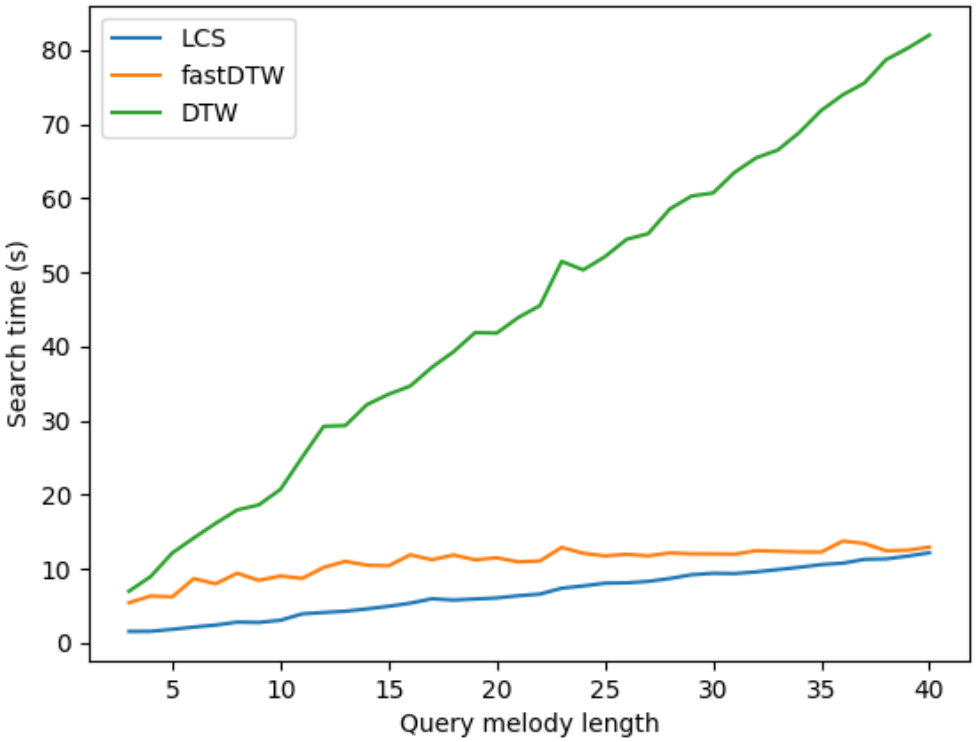
6 Měření

Prvním zajímavým experimentem bylo měření rychlosti vyhledávání při plné databázi (295 skladeb) v závislosti na délce dotazu. Vstupní melodii jsem postupně zvyšoval od 3 not až na 40 not. Testoval jsem jak oba dva mnou implementované algoritmy (LCS a DTW), tak také pro zajímavost algoritmus fastDTW [3], dostupný jako Python package, který by měl DTW implementovat v lineárním čase a paměti.

Z grafu je patrné, že LCS a DTW jsou ovlivňovány délkou dotazu lineárně, což částečně potvrzuje, že oba mají složitost $O(mn)$. DTW je však pomalejší o nějakou konstantu – průměr této konstanty byl $6,61$.

Naopak fastDTW byl ovlivněn délkou dotazu minimálně, což odpovídalo předpokladu, že algoritmus skutečně pracuje v lineárním čase.

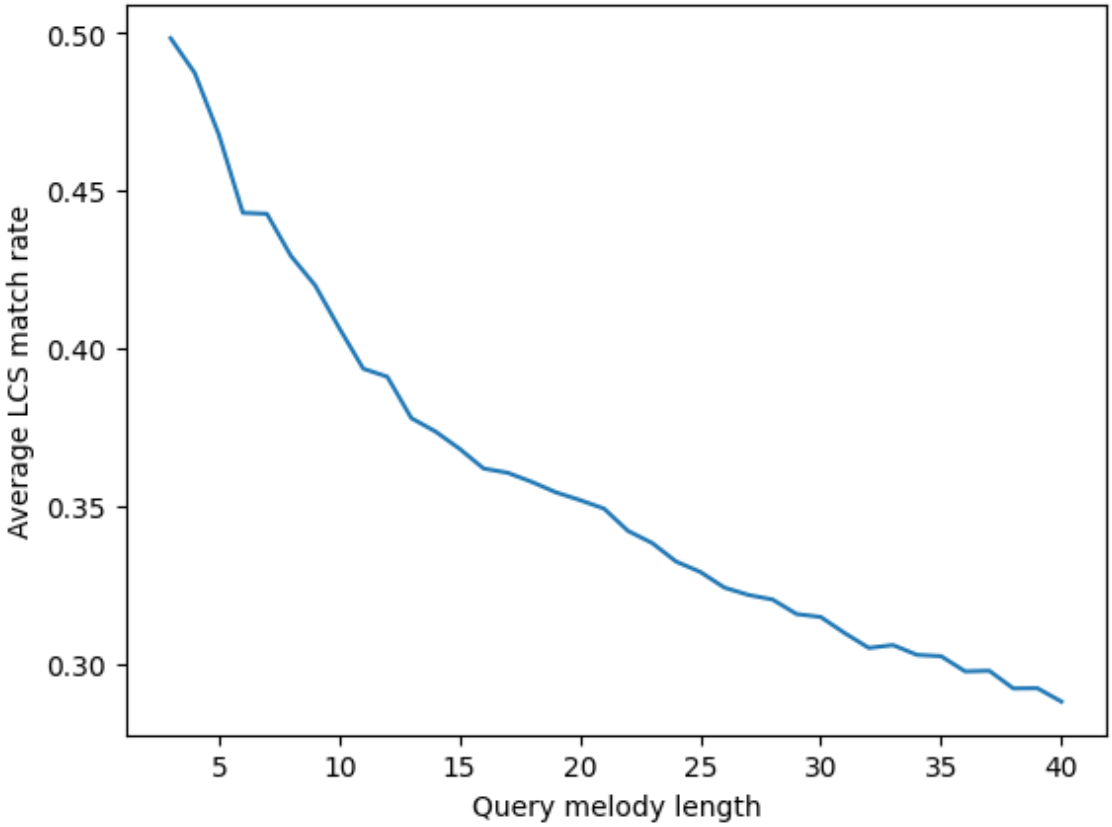
Obrázek 7: Graf závislosti délky vstupní melodie a rychlosti vyhledávání algoritmů



V dalším experimentu jsem zkoumal opět na odlišné délce vstupní melodie, jak se vyvíjí průměrná délka největší společné podsekvence ku maximální možné délce společné podsekvence (tedy plná délka vstupní melodie). V podstatě tato hodnota udává, jak moc podobné jsou všechny nejlepší segmenty každé skladby (prozkoumáno vždy všech 295 skladeb v databázi) vstupní melodii.

V grafu je vidět, že při velmi malém vstupu dokážeme najít opravdu hodně velmi podobných segmentů – průměrná podobnost se pohybuje mezi $45 - 50\%$. Při středně velkém vstupu (10–15) však průměrná podobnost rychle klesá, a při velkých vstupech (25–40) klesá pomaleji.

Obrázek 8: Graf závislosti délky vstupní melodie a průměrné podobnosti



Vývoj této metriky můžeme interpretovat tak, že velmi malé vstupy nejsou příliš užitečné pro to, abychom v seznamu výsledků našli to, co hledáme – v seznamu totiž bude mnoho skladeb s vysoce podobnými segmenty. Stačí však zadat o trochu delší vstup a průměrná hodnota podobnosti rychle klesá, a tím pádem seznam výsledků bude trochu čitelnější.

7 Další vývoj

Nejprve lze vylepšit samotný sekvencer v uživatelském rozhraní – lze přidat mnoho ovládacích prvků, např. dosah dostupné klaviatury, aby se mohly používat vyšší/-nižší klávesy. Nastavitelná by také mohla být rychlost přehrávání.

Samotné vyhledávání je velice pomalé – nejen, že běží v Pythonu, ale také nelze použít multithreadingu, což se navíc obecně nepraktikuje na web serverech – výpočetně náročné úkony jsou většinou předány další aplikaci, napsané např. v jazyce C++, Rust… Zrychlení vyhledávání by tak spočívalo nejen v překopání lineárního prohledávání (namísto toho použít indexaci), ale také v přesměrování výpočetně náročného úkonu jiné službě napsané v jiném jazyce.

8 Závěr

Vytvořil jsem aplikaci včetně backendu a frontendu, ve které jsem implementoval všechny kroky k provedení vyhledávání, od extrakce melodie až k zobrazování výsledků v uživatelském rozhraní. Výsledná aplikace kromě samotného vyhledávání může sloužit jako dobrá ukázka výhod či nedostatků obou implementovaných algoritmů – LCS a DTW. Při implementaci jsem měl možnost naučit se práci s novými technologiemi (např. Django) či osvěžit si znalosti u jiných. Kvůli použití knihovny *Mido* jsem musel hlouběji pochopit standard MIDI, což považuji za spíše pozitivní zkušenost (i když má standard MIDI své mouchy). Obecně bych jak zkušenost při práci na projektu, tak konečný výsledek hodnotil kladně.

Odkazy

1. *Standard MIDI-File Format Spec. 1.1, updated* [online]. [cit. 2022-11-27]. Dostupné z: http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html.
2. *Mido - MIDI Objects for Python* [online]. [cit. 2022-11-27]. Dostupné z: https://mido.readthedocs.io/en/latest/.
3. *fastdtw 0.3.4* [online]. [cit. 2022-11-29]. Dostupné z: https://pypi.org/project/fastdtw/.