PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Comparação de padrões melódicos para recuperação musical

Theo Necyk Agner Caldas

RELATÓRIO DE PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Ciência da Computação

Rio de Janeiro, novembro de 2023

PUC
RIO

Theo Necyk Agner Caldas

Comparação de padrões melódicos para recuperação musical

Relatório de Projeto Final, apresentado ao programa Ciência da Computação da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Augusto César Espíndola Baffa

Rio de Janeiro

novembro de 2023

Resumo

Caldas, Theo. Baffa, Augusto. Comparação de padrões melódicos para recuperação musical. Rio de Janeiro, 2023. Pontifícia Universidade Católica do Rio de Janeiro.

Este projeto se propõe a explorar métodos de comparação melódica monofônica para construção de ferramenta de auxílio à identificação de padrões musicais, com foco no processo de composição. Discute-se técnicas para resolver o problema de recuperação baseada em conteúdo melódico, no qual o usuário recupera informações sobre determinada obra musical através de uma interpretação, cantada ou tocada, de seus fragmentos melódicos. Para tal, levanta-se ferramentas já existentes para viabilizar a construção do protótipo que implementa os métodos discutidos.

Palavras-chave
reconhecimento de melodias monofônicas, extração de padrões melódicos, recuperação baseada em conteúdo, música.

Abstract

Caldas, Theo. Baffa, Augusto. Melodic pattern comparison for music retrieval. Rio de Janeiro, 2023. Pontifícia Universidade Católica do Rio de Janeiro.

The project explores monophonic melodic comparison methods in order to build a tool that identifies and visualizes musical patterns, with the goal of helping musicians surpass creative barriers in the composing process. It also aims the problem of melodic content-based retrieval, in which the user may recover song information given a played or singed interpretation of its melodic segments. Therefore, we explore external tools that enable the development of a prototype that implements the studied methods.

Key words
monophonic melody recognition, melodic pattern extraction, content-based retrieval, music.

Agradecimentos

Em homenagem a minha mãe por me apresentar a música, antes mesmo de eu nascer. Em homenagem ao meu pai por me apresentar os joguinhos de computador, antes mesmo de eu saber ler. Dedico esse trabalho à minha mãe e ao meu pai, que sempre apoiaram os meus sonhos e caminhos.

Agradeço a minha irmã, Julia Niemeyer, por ser minha maior inspiração na área acadêmica. Agradeço ao meu primo, Júlio Necyk, por me apresentar a programação e o curso de Ciência da Computação na PUC-Rio. Agradeço a meu amigo, Matheus Kulick, sem o qual eu não teria chegado ao final desse curso. Agradeço aos demais colegas que me motivaram ao longo desses cinco anos. Agradeço ao corpo docente do Departamento de Informática da PUC-Rio e aos demais funcionários.

Por fim, agradeço aos amigos Lucas Lima, Lara Miranda, João Terra, Matheus Kulick, Fernando Neto, Alice Guimarães e Leandro Marinho por gravar arquivos de áudio e autorizar sua utilização durante a pesquisa realizada.

The battle outside ragin'
Will soon shake your windows
And rattle your walls
For the times they are a-changin'

Bob Dylan

Sumário

1. Introdução 1
2. Metodologia 3
2.1. Terminologia 3
2.2. Formalização do Problema 6
2.3. Pesquisa Inicial 8
2.3.1. Análise de Similares 8
2.3.1.1. Análise de Acurácia 10
2.3.1.2. Análise de Performance 11
2.3.1.3. Análise de Abrangência 11
2.3.1.4. Análise de Diversidade de Entrada 13
2.3.1.5. Análise de Robustez de Entrada 14
2.3.1.6. Análise de Flexibilidade 15
2.3.1.7. Análise de Transparência 16
2.3.1.8. Conclusão da Análise de Similares 16
2.3.2. Principais Abordagens 18
2.3.2.1. Rede Neural 19
2.3.2.2. Comparação de Sequências e Programação Dinâmica 20
2.3.2.3. Histograma 22
3. Objetivos 24
4. Fundamentos 25
4.1. Estudos Preliminares 25
4.2. Implementação das abordagens 30
4.2.1. Estudo sobre Mongeau e Sankoff 30
4.2.1.1. Versão Inicial 30
4.2.1.2. Adaptação para sequência melódica 33
4.2.1.3. Normalização 34
4.2.1.4. Conclusão do Estudo sobre Mongeau e Sankoff 36
4.2.2. Estudo sobre Liu, Wu e Chen 36
4.2.2.1. Distância de Histogramas 37
4.2.2.2. Espaço de Histogramas e Árvore R* 38

4.2.2.3. Conclusão do Estudo sobre Liu, Wu, Chen 39
4.3. Estudos em Leitura e Tratamento de Áudio 40
4.3.1. Componente de Mapeamento Monofônico 40
4.3.2. Componente de Conversão WAVE 42
4.3.3. Componente de Mapeamento de Volume 42
4.3.4. Componente de Isolamento de Instrumento 43
4.4. Estudos em Leitura de MIDI 44
4.4.1. Componente de Leitura de Arquivo MIDI 44
5. Projeto e Especificação do Sistema 46
5.1. Fluxogramas 46
5.2. Arquitetura de Código 49
5.3. Diagramas de Classe 49
6. Solução do Protótipo 52
6.1. Consolidação do Dataset 52
6.1.1. MIDI x Áudio 52
6.1.2. Acesso e Notoriedade 52
6.1.3. Discografia The Beatles em MIDI 53
6.2. Leitura e Codificação 53
6.3. Extração de Padrões Melódicos 53
6.4. Recuperação 55
7. Testes e Resultados 57
7.1. Planejamento e execução de testes funcionais 57
7.2. Transparência 59
8. Considerações Finais 60
9. Referências Bibliográficas 61
10. Anexos 64
10.1. Lista de Músicas do Protótipo 64

Lista de Figuras

Figura 1 - Recuperação baseada em conteúdo melódico. 7
Figura 2 - Extração e comparação de padrões melódicos. 7
Figura 3 - Ferramenta de auxílio ao processo de composição musical. 17
Figura 4 - Visualização de espectograma de par de entradas. 20
Figura 5 - Transformações de S1 = (a, a, b, c) para S2 = (a, b, d). 21
Figura 6 - Histogramas de S1 = (a, a, b, c) e S2 = (a, b, d). 23
Figura 7 - Visualização (tempo x frequência) da melodia m1. 26
Figura 8 - Visualização dos contornos melódicos m1 (azul), m2 (vermelho) e m3 (verde). 27
Figura 9 - Visualização dos contornos melódicos de m1, m2, m3, m4 e m5. 28
Figura 10 - Visualização dos contornos melódicos de m6 e m7. 28
Figura 11 - Pseudocódigo da recorrência da função de distância. 31
Figura 12 - Pseudocódigo dos casos base da recorrência da função de distância. 32
Figura 13 - Valor de distância e caminho ótimo de transformações na chamada de T8. 33
Figura 14 - Valor de distância e caminho ótimo de transformações na chamada de T14. 35
Figura 15 - Cálculo do valor de distância de histograma entre S1 e S2. 37
Figura 16 - Cálculo da quantidade de inserções de HV(S1) para HV(S2). 37
Figura 17 - Gráfico (volume x tempo) extraído de um arquivo de áudio. 42
Figura 18 - Separação de faixa não monofônica em duas faixas monofônicas. 45
Figura 19 - Leitura de arquivo áudio ou MIDI (processo 1). 47
Figura 20 - Extração de padrões melódicos (processo 2). 47
Figura 21 - Busca de instâncias musicais (processo 3). 48
Figura 22 - Treinamento (fluxo 1) e recuperação (fluxo 2). 49
Figura 23 - Classe Parser. 50
Figura 24 - Classe Encoder. 50
Figura 25 - Classe MelodyComparer. 51
Figura 26 - Classe MelodyStorage. 51
Figura 27 - Caso ideal. 54
Figura 28 - Caso intermediário. 54
Figura 29 - Caso desfavorável. 55
Figura 30 - Intervalos originais. 56
Figura 31 - Intervalos após transformações aplicadas. 56

Lista de Tabelas

Tabela 1 - Resultados da análise de acurácia.....10
Tabela 2 - Resultados da análise de performance. 11
Tabela 3 - Resultados da análise de abrangência. 12
Tabela 4 - Resultados da análise de diversidade de entrada.....13
Tabela 5 - Resultados da análise de robustez de entrada. 14
Tabela 6 - Resultados da análise de flexibilidade. 15
Tabela 7 - Resultado dos testes em MIDI. 59

Motivação

Este Projeto Final apresentado ao programa Ciência da Computação da PUC-Rio partiu da vontade pessoal de integrar o domínio técnico e teórico adquirido ao longo de cinco anos de dedicação ao estudo dos fundamentos computacionais com a paixão incansável pela arte musical, que insiste em acompanhar minha jornada pessoal e profissional. Pessoalmente, acredito que os produtos do desenvolvimento tecnológico têm caráter tanto construtivo quanto destrutivo, no que afeta a vida de pessoas em sociedade. Portanto, cabe às pessoas responsáveis por um projeto em computação, o exercício do pensamento crítico sobre como a tecnologia em potencial afeta e remodela as relações plurais de trabalho já estabelecidas. No contexto contemporâneo da indústria musical, o papel da tecnologia resulta tanto na criação de pontes quanto na consolidação de barreiras entre o artista e o público em potencial. Se torna indispensável a reflexão sobre o impacto que a ascensão de plataformas e serviços digitais, as quais se encontram em constante mudança, proporciona ao consumo e produção de música. Nesse cenário, é com entusiasmo que busco contribuir para o estudo acadêmico computacional, no que tange a exploração diligente de novas formas de pessoas se relacionarem com a música.

1. Introdução

Considere o seguinte cenário: uma pessoa está com uma música "presa na cabeça" e quer muito ouvi-lá, mas se sente frustrada por não conseguir encontrá-la através de mecanismos padrões de busca, já que não sabe seu nome, não lembra sua letra e desconhece seu autor. Quem já viveu uma situação parecida consegue entender a importância do componente melódico para o campo musical.

Não raro é o caso de uma pessoa ter vaga lembrança de uma obra musical apenas pelo seu conteúdo melódico. Apesar de não ser o único elemento musical capaz de evocar sua lembrança, a melodia é parte fundamental da identificação e diferenciação de determinada obra [1]. Sendo assim, embora seja um padrão consolidado nos mecanismos de busca atuais, o processo de recuperação de música por palavras chave em texto plano — como nome, álbum, artista e letra (se houver) — se torna ineficaz nas poucas ocasiões onde essas informações não são de conhecimento do usuário. Nesse caso, a busca de música por melodia se torna uma alternativa em potencial.

O processo de busca de músicas baseada em sequências melódicas, denominado de *recuperação baseada em conteúdo*, é uma das possíveis aplicações projetadas a partir de um algoritmo de *extração de padrões* sobre melodias monofônicas (somente uma voz musical). Um sistema que implementa tal processo pode, por exemplo, pedir ao usuário uma interpretação em áudio de determinada obra musical — seja de forma cantada, assobiada ou tocada por um instrumento — e, a partir dela, apontar para instâncias musicais previamente analisadas (no contexto desse projeto, o termo instância musical designa uma gravação particular de determinada obra musical) que contenham fragmentos melódicos similares aos da melodia fornecida. O grau de semelhança metrificada para ditar se tal melodia é similar a outra é arbitrário e está sujeito à discussão, mas variáveis como tom e andamento são comumente relativizadas nesses sistemas. Considerando a captura de áudio em ambientes não profissionais e usuários com pouca proficiência musical, nota-se a dificuldade em identificar e descartar ruídos externos ou desvios melódicos não intencionais.

Durante a fase inicial de pesquisa, constatou-se que projetos de algoritmo para classificar similaridades melódicas datam da década de 1990. Mongeau e Sankoff [2] descrevem a função de distância entre dois fragmentos melódicos com base na teoria de comparação de sequências, apresentando um algoritmo de programação dinâmica em tempo polinomial. O algoritmo é, então, aprimorado por Rolland [3], que introduzi mais controle sobre os parâmetros e formaliza conceitos como descrições melódicas, ATPS (conjunto de tipos de

pareamento permitidos) e categorização por grafo de similaridade. Trabalhos posteriores, como o de Li, Wu, Chen [4], buscam otimizar o algoritmo, simplificando a função de distância de forma a não considerar a ordenação das notas.

Três décadas após as primeiras publicações de artigos sobre o assunto, em outubro de 2020, a empresa Google lança o sistema "Hum To Search" [5], que soluciona o problema de *recuperação baseada em conteúdo melódico*. O sistema, integrado ao motor de busca da Google, pode ser considerado o principal serviço no mercado que disponibiliza eficiente e abrangentemente a busca de música por melodia, ultrapassando aplicativos *mobile* como o SoundHound [6]. Nota-se que existem sistemas similares que realizam o reconhecimento de músicas somente pela abordagem de *fingerprint*, como o aplicativo Shazam [7]. No entanto, esses apenas lidam com a comparação direta de instâncias musicais em bancos de músicas, não sendo possível a interpretação do usuário. A abordagem computacional proposta pela Google é essencialmente diferente das encontradas nos artigos da década de 1990, pois utiliza técnicas sofisticadas de aprendizado de máquina, como indica sua equipe de pesquisa [8].

Acredita-se que há potencialidade para melhorias na maioria desses sistemas, principalmente pela adição de funcionalidades. Nesse contexto, vejo a oportunidade de criação de uma ferramenta de software que não apenas liste músicas que contenham melodias similares, mas que também elucide a semelhança e a recorrência entre diferentes fragmentos melódicos. Essa aplicação pode ser interessante como ferramenta auxiliar ao artista durante a busca de referências e criação de ideias, seja no processo de composição musical ou em seus estudos musicais gerais. Em nenhum momento durante o projeto, discute-se a geração automática de linhas melódicas por parte do sistema.

2. Metodologia

Esta seção tem como objetivo garantir uma visão geral do que se trata o projeto, de onde partiu a pesquisa e qual é a sua proposta principal. Uma terminologia de conceitos musicais é fornecida, para maior entendimento por parte do leitor. Em seguida, há a formalização do problema estudado. Logo após, encontra-se uma síntese da análise de similares realizada, que culmina na identificação de oportunidades a serem exploradas dentro do contexto da indústria de aplicações musicais. Por último, são apresentadas as principais abordagens técnicas estudadas, que serão aprofundadas mais a frente.

2.1. Terminologia

Primeiramente, procura-se esclarecer alguns conceitos presentes neste relatório. O livro Teoria da Música, por Bohumil Med, [9] define algumas das noções musicais utilizadas. Como o projeto é de natureza computacional e não se direciona necessariamente para leitores com domínio em música, apresenta-se definições adaptadas para uma perspectiva inteligível por leigos.

- nota musical: menor elemento de um som. Definida por sua altura e duração.

- altura (de uma nota): característica sonora que determina a qualidade grave ou aguda de um som emitido. É definida pela frequência de onda (em Hertz) mas comumente é discretizada em tons ou graus — Dó, Ré, Mi, Fá, Sol, Lá, Si — que são tradicionalmente escritos pelas letras C, D, E, F, G, A, B, respectivamente.

- duração (de uma nota): quantidade de tempo em que o som é emitido e sua altura permanece constante. Pode ser definida em tempos relativos, como mínima, semínima, colcheia, semicolcheia, entre outros valores.

- pausa: quantidade de tempo entre o fim de uma nota e o início da próxima, isto é, a duração do silêncio. Também pode ser definida em tempos relativos. Para efeito prático, uma pausa pode ser interpretada como uma nota sem altura.

- intervalo: diferença discreta entre a altura de duas notas. Diz ser ascendente quando a nota antecessora é mais grave que a sucessora e descendente quando a nota antecessora é mais aguda que a sucessora.

- oitava: intervalo a partir do qual a percepção dos tons passa a se repetir. A título de exemplo, dada uma nota C, a nota a uma oitava ascendente, embora mais aguda, também é C. A informação sobre a qual oitava determinada nota pertence pode ser indicada por um número que acompanha a escrita da nota. Dessa forma, C4 é a nota C uma oitava mais aguda que C3, C5 é mais agudo que C4 e assim por diante. Com raras exceções, a massa de produção musical contemporânea segue a tradição musical da Europa-Ocidental, que por convenção adere ao Sistema de Afinação Temperado, o qual divide a oitava em doze semitons de frequências equidistantes.

- melodia: um dos principais componentes estruturais que definem uma obra musical. Trata-se de notas encadeadas em sequência. Pode ser escrita em uma pauta musical e tocada por algum instrumento melódico, incluindo o canto (com ou sem palavras). Uma obra musical pode conter muitas melodias que se complementam, mas é comum o caso de existir uma melodia principal que caracteriza a obra.

- melodia monofônica: quando não especificado, melodia se refere a melodia monofônica. Trata-se de uma melodia com somente uma voz musical, isto é, na qual ocorre apenas uma nota por vez.

- fragmento melódico, sequência melódica ou submelodia: sequência de notas contidas (parte) de uma melodia maior. Possivelmente, é antecedida e sucedida por longas pausas e ocorre em mais de um momento (se repete) ao longo da obra musical.

- padrão ou contorno melódico: sequência de intervalos mais expressivos em um fragmento melódico. Pode-se dizer que um fragmento melódico segue determinado padrão caso não sofra grandes variações de notas.

- interpretação melódica: performance particular da melodia de uma obra musical. Cada performance é única por conter variáveis que a distinguem como tom, andamento, entonação, ritmo, timbre, entre outras. A interpretação deve seguir os contornos melódicos da melodia original.

- ritmo: disposição de tempo entre notas e pausas, que se repete ao longo de uma obra musical. Pode conter acentuações, isto é, tempos em que a presença de notas é mais expressiva.

- tom (de uma melodia): pode ser entendido, vulgarmente, como a altura de uma melodia. Melodias costumam ser escritas sobre um tom, isto é, partindo de uma nota específica. Entretanto, o tom não é intrínseco à melodia, de forma que melodias podem ser transpostas livremente, respeitando seus intervalos.

- andamento: a velocidade (em tempos discretos por minuto) em que uma melodia é tocada. Toda interpretação melódica possui um andamento, mas uma mesma melodia pode ser performada em qualquer andamento, desde que preservados o tempo relativo de suas notas e pausas (ritmo).

- obra ou composição musical: uma música. Possui componentes estruturais como harmonia, arranjo, letra e, sobretudo, melodia. Geralmente, sua criação é atribuída a um compositor ou a um grupo de compositores.

No contexto desse projeto, o termo "instância musical" se refere a uma determinada performance oficializada — gravada, produzida, registrada e publicada — de uma obra em particular. Uma mesma obra pode conter uma ou milhares de instâncias, que não necessariamente são de autoria do compositor da obra.

Por último, o termo distância é aplicado para o grau de dissimilaridade entre duas melodias ou dois fragmentos melódicos. Logo, nesse estudo, melodias parecidas podem ser ditas próximas uma da outra, enquanto melodias distintas são distantes entre si.

A partir dessa terminologia é possível ter um maior entendimento da pesquisa realizada. A seguir, propõe-se uma formalização do problema ao qual o projeto se refere.

2.2. Formalização do Problema

O projeto propõe-se a estudar o problema computacional de *recuperação de música baseada em conteúdo melódico* (*melodic content-based retrieval*). Os primeiros artigos que abordam tal problema foram publicados na década de 1990. Rolland [3] define, de forma objetiva, o problema em questão. Em suma, trata-se de efetuar uma busca em um banco de instâncias musicais, cuja entrada é uma interpretação de fragmento melódico. O processo de busca, então, deve retornar instâncias musicais que contenham os fragmentos melódicos que mais se assemelham à entrada. A principal aplicação desse serviço é capacitar o usuário de recuperar uma instância musical desejada — e consigo os seus metadados, como nome, artista, duração, entre outros — a partir do conhecimento à priori da estrutura melódica da obra. Ou seja, procura-se obter o conteúdo todo através de sua parte. Ademais, através da comparação melódica, o usuário pode obter instâncias musicais não conhecidas à priori, adquirindo um maior conhecimento das múltiplas e complexas similaridades em obras musicais distintas. Para que qualquer *recuperação baseada em conteúdo* funcione, é preciso saber comparar de forma metrificada diferentes conteúdos. No caso do estudo melódico, esse subprocesso pode ser chamado de *extração de padrões melódicos*, como define Rolland [3]. Alguns autores usam o termo *comparação de padrões melódicos* para se referir ao subprocesso em questão. Com efeito, nesse projeto, os termos são considerados sinônimos.

Logo, o fluxo de resolução do problema deve incluir duas grandes etapas: (1) o treinamento do sistema; e (2) a execução da busca (ou recuperação). Na primeira, não há entrada do usuário, e a *extração de padrões melódicos* é utilizada para analisar todo o banco de instâncias musicais de forma a adquirir conhecimento. Na segunda, a recuperação é, de fato, realizada, visto que a *extração de padrões* é utilizada para comparar a entrada do usuário com as várias instâncias pré-analisadas pelo sistema.

A figura 1 ilustra o processo de *recuperação baseada em conteúdo melódico*, na qual ocorre a comparação de um (entrada do usuário) para muitos (instâncias previamente analisadas) padrões melódicos. A saída do sistema é um conjunto das instâncias que contém fragmentos melódicos semelhantes à entrada.

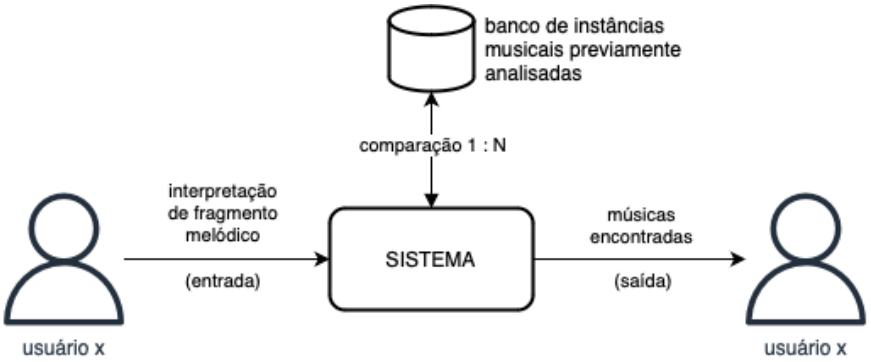

Figura 1 - Recuperação baseada em conteúdo melódico.
Fonte: elaborada pelo autor.

A *extração* (ou comparação) *de padrões melódicos* é a parte do processo que contém um algoritmo que dita o grau de dissimilaridade (ou de similaridade, a depender da implementação) entre dois fragmentos melódicos. O algoritmo deve conter alguma forma de obter a informação melódica — como altura e duração das notas — contida na melodia de entrada, seja através de arquivo de áudio (isto é, sua representação original) ou através de sua codificação para alguma representação intermediária. O algoritmo, então, deve se basear em alguma heurística pré-definida para ditar o quanto a representação melódica de um fragmento se assemelha à outra. Mais a frente, discute-se possíveis codificações e heurísticas encontradas durante a pesquisa do projeto.

A figura 2 ilustra o processo descrito, no qual os arquivos que contém duas melodias *A* e *B* são codificados e, em seguida, o valor de similaridade entre seus padrões melódicos é retornado pelo o algoritmo de comparação.

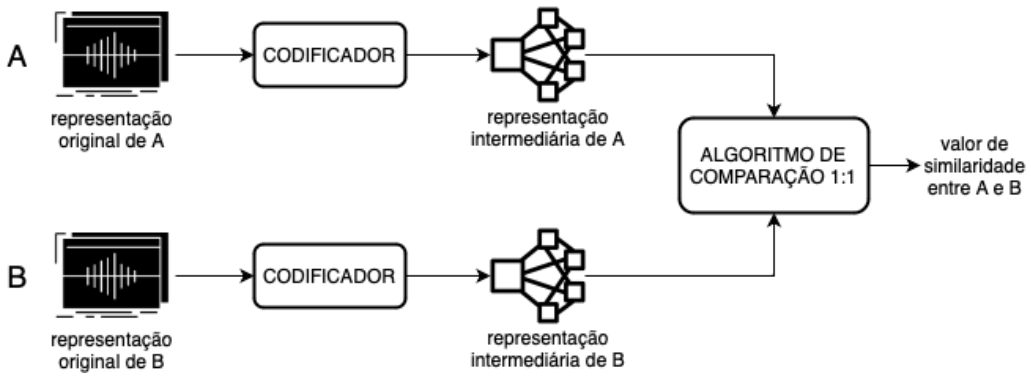

Figura 2 - Extração e comparação de padrões melódicos.
Fonte: elaborada pelo autor.

2.3. Pesquisa Inicial

A pesquisa realizada ao longo da fase inicial do projeto teve foco em: (1) destacar sistemas oferecidos comercialmente para o grande público que implementem a *recuperação baseada em conteúdo*, com intuito de elucidar oportunidades de aprimoramento do estado da arte; e (2) encontrar artigos publicados que discutem algoritmos de *extração e comparação de padrões melódicos*, para estudo e implementação de seus métodos.

2.3.1. Análise de Similares

Considerando o primeiro aspecto da pesquisa, decidiu-se fazer uma análise dos sistemas similares encontrados. Ao final dessa análise, concluiu-se que o sistema “Hum To Search” [5] da Google é atualmente o mais eficiente e eficaz serviço e, portanto, é a principal referência para este projeto. Ele pode ser acionado diretamente pelo popular motor de busca, selecionando a opção de captura de voz e, em seguida, a opção “Pesquise uma música”. Além desse similar, também destacou-se os aplicativos SoundHound [6] e o Shazam [7]. Para sedimentar a análise de seus pontos positivos e negativos, planejou-se uma série de conjuntos de testes que comparam os três sistemas similares.

A seguir, são definidas as qualidades almejadas na avaliação desses sistemas similares. A terminologia dessas qualidades se limita ao contexto deste projeto. O estudo proposto defende que um sistema que se propõe a resolver o problema de *recuperação baseada em conteúdo melódico* deve garantir excelência nos seguintes aspectos:

- Acurácia: o nível de precisão geral de resposta, isto é, o grau em que a saída do sistema é esperada pelo usuário, dada sua entrada.

- Performance: baixo tempo médio (em segundos) de resposta do sistema.

- Abrangência: o nível de amplidão de conhecimento do sistema, isto é, o quão largo é seu banco de músicas pré-analisadas, considerando instâncias musicais com diferentes níveis de popularidade, diferentes gêneros, e oriundas de diferentes regiões do mundo.

- Diversidade de entrada: o grau de capacidade do sistema de compreender diferentes fontes de áudio como entrada, entre elas, uma interpretação melódica cantada, assobiada, tocada por algum instrumento melódico ou a reprodução da própria instância musical.

- Robustez de entrada: o grau de capacidade do sistema de receber entradas de áudio com diferentes níveis de ruído, qualidade da captação de áudio e reverberação do ambiente.

- Flexibilidade: o grau de capacidade do sistema de ignorar imprecisões melódicas, isto é, o quanto ele é capaz de trabalhar com interpretações melódicas considerando diferentes níveis de proficiência musical do usuário.

- Transparência: o grau de clareza na forma em que o sistema apresenta o processo de comparação realizado, isto é, o quanto ele permite que o usuário entenda como determinada entrada foi mapeada para determinada saída. A transparência do sistema pode ser alcançada de forma gráfica ou sonora.

Para cada qualidade definida, com exceção da qualidade de transparência, foi gerado um conjunto de arquivos de áudio teste contendo interpretações melódicas de obras que buscam estressar tal qualidade em cada um dos sistemas. Os arquivos de áudio foram gerados de forma autoral, salvo o uso autorizado de arquivos de terceiros a ser especificado. As interpretações melódicas buscam representar um fragmento expressivo da obra em questão, e variam entre quinze e trinta segundos aproximadamente. Para a análise de cada qualidade, restrições foram definidas para a gravação dos áudios, a fim de não sobrepor a contribuição de mais de um fator analisado.

Os resultados foram sintetizados em tabelas. Com exceção da análise de performance, os resultados expressos nas tabelas possuem a seguinte legenda:

- ✓ : dada a reprodução do arquivo de áudio contendo uma interpretação melódica da obra esperada, o sistema apresentou uma lista contendo uma ou mais instâncias musicais da obra esperada.

- × : dada a reprodução do arquivo de áudio contendo uma interpretação melódica da obra esperada, o sistema não conseguiu apresentar alguma instância musical, isto é, não teve resposta.

- × (?) : dada a reprodução do arquivo de áudio contendo uma interpretação melódica da obra esperada, o sistema apresentou uma lista de instâncias musicais as quais categorizou como similar, mas a obra esperada não condisse com nenhuma delas.

Ademais, a coluna "Google" agrupa os resultados do sistema "Hum to Search" [5] da empresa Google, gerados através de seu aplicativo iOS. Da mesma forma, o sistema SoundHound [6] foi testado a partir de seu aplicativo iOS, como também foi o sistema Shazam [7], da empresa Apple.

2.3.1.1. Análise de Acurácia

O conjunto de arquivos de áudio para a análise de acurácia é composto por interpretações melódicas de obras populares norte-americanas, as quais imagina-se ser de conhecimento dos sistemas. As interpretações melódicas foram cantadas (com palavras) por uma mesma pessoa e capturadas por um microfone profissional em ambiente com ruído externo mínimo, buscando expressar fielmente o fragmento melódico desejado salvo alterações de tom e andamento.

Obra - Compositor / Sistema	**Google**	**SoundHound**	**Shazam**
Imagine - John Lennon	✓	✓	×
Just The Way You Are - Bruno Mars	✓	✓	×
As it Was - Harry Styles	✓	× (?)	×
Yellow - Coldplay	✓	✓	×
Happier Than Ever - Billie Ellish	✓	× (?)	×
Umbrella - Rihanna	✓	✓	×
Is This Love - Bob Marley	✓	✓	×
Fly Me to the Moon - Bart Howard	✓	✓	×

Tabela 1 - Resultados da análise de acurácia.
Fonte: elaborada pelo autor.

O primeiro ponto que deve-se observar a partir do resultado dessa análise (apresentado na tabela 1) está no fato de que o sistema Shazam não reconhece obras a partir de interpretações melódicas. Na realidade, o sistema se propõe a encontrar músicas apenas mediante à reprodução de suas instâncias musicais, isso é, não realiza nenhum tipo de extração melódica. Portanto, para as próximas análises, o Shazam foi desconsiderado, não servindo como sistema similar.

A partir dessa primeira análise, percebe-se que o sistema da Google apresenta uma maior acurácia geral, identificando todas as obras esperadas. Suspeita-se que, para atingir o alto nível de precisão, o sistema da Google realize, em paralelo à *recuperação baseada em conteúdo melódico*, uma busca

por letra de músicas através de um algoritmo de reconhecimento de fala humana. Em comparação, visto que músicas populares norte-americanas são amostras representativas e considerando a boa qualidade de captação de áudio, a pouca presença de ruído externo e o razoável nível de proficiência musical do usuário, esperava-se que o sistema SoundHound apresentasse mais acurácia. Entretanto, ele acertou apenas seis das oito músicas esperadas.

2.3.1.2. Análise de Performance

A análise de performance foi realiza em conjunto à análise de acurácia, cronometrando o tempo de resposta para cada música corretamente recuperada. A tabela 2 indica o tempo de resposta, em segundos, a partir do momento de início de reprodução do arquivo de áudio, já que em algumas ocasiões os sistemas apresentam o resultado antes do fim da reprodução do áudio.

Obra - Compositor / Sistema	**Google**	**SoundHound**
Imagine - John Lennon	8.31	18.63
Just The Way You Are - Bruno Mars	8.16	16.75
As it Was - Harry Styles	8.70	-
Yellow - Coldplay	8.36	17.48
Happier Than Ever - Billie Eilish	8.74	-
Umbrella - Rihanna	8.16	18.44
Is This Love - Bob Marley	8.59	17.99
Fly Me to the Moon - Bart Howard	8.05	18.39

Tabela 2 - Resultados da análise de performance.
Fonte: elaborada pelo autor.

O sistema da Google é muito eficiente, com tempo médio de resposta de 8.38 segundos. Em contraste, o SoundHound apresentou tempo médio de resposta de 17.94 segundos. Mais uma vez, suspeita-se que a Google realize busca de música por letra cantada, em paralelo à *recuperação baseada em conteúdo melódico.*

2.3.1.3. Análise de Abrangência

Após validado o fato de que ambos os similares conseguem reconhecer, com alguma precisão, obras musicais gerais a partir de interpretações cantadas (com palavras) de seus fragmentos melódicos, procurou-se pensar em obras que

imagina-se serem de menor conhecimento por parte do sistema. Assim, selecionou-se músicas regionais brasileiras, variando gênero — como forró, funk carioca, MPB, marchinha de carnaval, hinos oficiais, rock alternativo — e de artistas mais e menos conhecidos. Também entraram no conjunto de testes uma música em japonês e um jazz vocal do final da década de 1930.

As interpretações melódicas foram cantadas (com palavras) por uma mesma pessoa e capturadas por um microfone profissional em ambiente com ruído externo mínimo, buscando expressar fielmente o fragmento melódico desejado salvo alterações de tom e andamento. A única exceção foi a música tema do Metrô Rio, composta por Zanna, que foi tocada em um piano, instrumento que ambos os sistema são capazes de receber como entrada, como será demonstrado em seguida na análise de diversidade.

Obra - Compositor / Sistema	**Google**	**SoundHound**
Eu Só Quero um Xodó - Dominguinhos	✓	×
Haruka Kanata - Asian Kung-Fu Generation	✓	✓
Carry Me Back To Old Virginy - The Mill Brothers	× (?)	× (?)
Rap da Felicidade - Cidinho & Doca	✓	×
Acabou Chorare - Novos Baianos	✓	× (?)
Azul Da Cor Do Mar - Tim Maia	✓	×
Ô Abre Alas! - Chiquinha Gonzaga	✓	×
Hino do Brasil - Francisco Manuel da Silva	✓	✓
Clube de Regatas do Flamengo (Hino Popular) - Lamartine Babo	✓	×
Metrô Rio (Música Tema) - Zanna	✓	×
Solidão de Volta - Terno Rei	✓	× (?)
vcnvqnd - gorduratrans	✓	×
Farelos - Ana Frango Elétrico	× (?)	× (?)
Ruas - Aquino e a Orquestra Invisível	× (?)	× (?)
Flores - Sutil Modelo Novo	× (?)	×

Tabela 3 - Resultados da análise de abrangência.
Fonte: elaborada pelo autor.

Observando a tabela 3, o sistema da Google apresentou grande abrangência, recuperando onze das quinze obras selecionadas e falhando apenas para obras de artistas independentes cariocas pouco conhecidos por um

público amplo, assim como para o jazz antigo. O SoundHound apresentou baixíssima abrangência, recuperando apenas duas das quinze obras selecionadas. Curiosamente, as músicas identificadas foram "*Haruka Kanata"*, da banda japonesa Asian Kung-Fu Generation, e o hino nacional brasileiro, por Francisco Manuel da Silva.

2.3.1.4. Análise de Diversidade de Entrada

Dado que ambos os sistemas conhecem a obra *"Fly Me To The Moon"*, de Bart Howard, buscou-se testá-la para timbres de instrumentos diversos. Além da voz humana com palavras, o conjunto de áudio contém interpretações melódicas emitidas pela voz humana cantarolada (sem palavras), assobio, trompete e piano. Apesar de ser um instrumento harmônico (pode-se tocar mais de uma nota por vez), o piano foi tocado de forma monofônica, resultando em uma interpretação somente melódica. Além disso, foi testada a reprodução de uma instância musical famosa da obra, interpretada por Frank Sinatra.

As interpretações melódicas foram tocadas por uma mesma pessoa e capturadas por um microfone profissional em ambiente com ruído externo mínimo, buscando expressar fielmente o mesmo fragmento melódico desejado salvo alterações de tom e andamento.

Instrumento musical / Sistema	**Google**	**SoundHound**
Voz cantada (canto com palavras)	✓	✓
Voz cantarolada (canto sem palavras)	✓	×
Assobio	✓	×
Trompete	✓	✓
Piano (apenas melodia)	✓	✓
Reprodução de uma instância musical da obra	✓	✓

Tabela 4 - Resultados da análise de diversidade de entrada.
Fonte: elaborada pelo autor.

Ao observar a tabela 4, pode-se concluir que o sistema da Google expressa maior diversidade de entrada, retornando instâncias da obra esperada para todos os instrumentos planejados. Em paralelo, o sistema SoundHound falhou para a voz cantarolada e para o assobio.

É interessante notar que ambos funcionaram para o teste de reprodução da instância musical, considerando que essa possui vários instrumentos melódicos, harmônicos e percussivos tocados simultaneamente. Contudo, dado

que, para ambos, esse teste respondeu em tempo consideravelmente menor que os outros, acredita-se que os sistemas realizem, em paralelo à *recuperação baseada em conteúdo melódico*, uma busca baseada na abordagem de *fingerprint*, assim como o Shazam. Essa abordagem é mais eficiente em comparar instâncias musicais, mas, como não realiza a *extração de padrões melódicos*, não foi estudada durante o projeto.

2.3.1.5. Análise de Robustez de Entrada

Dado que ambos os sistemas conhecem a obra *"Imagine"*, de John Lennon, buscou-se testá-la para níveis de ruído diversos. Além de ruídos de som externos, considerou-se a qualidade de captação do áudio e a reverberação do ambiente de gravação. Foram testados áudios gravados por microfone profissional e por microfone de dispositivos celulares, ambos em ambiente livre de ruído externo. Com o microfone de celular, testou-se a gravação acompanhada de um instrumento harmônico (violão). Além disso, variou-se entre um ambiente fechado com ruído produzido por uma aparelho liquidificador, um ambiente de alta reverberação (banheiro domiciliar) com ruído de água corrente e um ambiente externo com ruído de multidão de pessoas.

As interpretações melódicas foram cantadas (com palavras) por uma mesma pessoa, buscando expressar fielmente o mesmo fragmento melódico desejado salvo alterações de tom e andamento.

Qualidade de captação + ruído / Sistema	**Google**	**SoundHound**
Microfone profissional + sem ruído	✓	✓
Microfone de celular + sem ruído	✓	×
Microfone de celular + violão	✓	×
Microfone de celular + liquidificador	✓	×
Microfone de celular + água do chuveiro	✓	✓
Microfone de celular + multidão	✓	×

Tabela 5 - Resultados da análise de robustez de entrada.
Fonte: elaborada pelo autor.

Considerando os resultado sintetizadas na tabela 5, tem-se que o sistema da Google possui uma grande robustez de entrada, satisfazendo todos os casos planejados. O sistema SoundHound, por outro lado, somente conseguiu gerar resposta para dois dos seis testes executados, incluindo o caso inicial ótimo.

2.3.1.6. Análise de Flexibilidade

Dado que ambos os sistemas conhecem a obra *“Just The Way You Are”*, de Bruno Mars, buscou-se testá-la para usuários de diversos níveis de proficiência musical. Com exceção do primeiro teste, todos os arquivos de áudio desse conjunto de teste foram gravados por terceiros. Para todos esses, após explicitado o propósito da utilização do áudio na pesquisa, foi solicitada a devida autorização de cada um dos contribuidores. Todos autorizaram. Para que o nome e identidade das pessoas que contribuíram não sejam expostos, os testes foram nomeados como “usuário 1”, “usuário 2”, e assim sucessivamente.

As interpretações melódicas desse conjunto de teste foram cantadas (com palavras) por seis pessoas diferentes com variadas proficiências musicais, buscando expressar fielmente o mesmo fragmento melódico desejado salvo alterações de tom e andamento. Além disso, as interpretações variam entre o timbre de voz feminino e masculino. Todos os arquivos de áudio desse conjunto de testes foram gravados por um microfone de dispositivo celular, em um ambiente livre de ruídos externos.

Autor da interpretação / Sistema	**Google**	**SoundHound**
Usuário 1	✓	✓
Usuário 2	✓	✓
Usuário 3	✓	×
Usuário 4	✓	✓
Usuário 5	✓	×
Usuário 6	✓	×

Tabela 6 - Resultados da análise de flexibilidade.
Fonte: elaborada pelo autor.

A partir de uma leitura da tabela 6, conclui-se que o sistema da Google possui uma excelente flexibilidade, recuperando instâncias musicais da obra para todas as interpretações melódicas testadas. Essa conclusão é realmente surpreendente visto que algumas interpretações possuem desvios melódicos muito acentuados do contorno melódico do fragmento cantado. O SoundHound, por outro lado, falhou para metade dos testes. Destaca-se a falha no teste “Usuário 5”, o qual possui uma interpretação melódica virtuosa, executada por uma pessoa com altíssima proficiência musical. Curiosamente, o SoundHound falhou para todas as pessoas no conjunto de testes com timbre de voz feminino.

2.3.1.7. Análise de Transparência

Para os casos em que os sistemas foram capazes de produzir uma lista de instâncias de resposta, seja a obra esperada pertencente a lista ou não, foi realizada uma breve análise de sua interface. Em ambos os sistemas, percebe-se que a qualidade de transparência não é atendida. Considerando a ocasião em que a obra esperada não está na lista de resposta, não é claro o porquê o sistema não identificou a similaridade entre a interpretação e as instâncias desejadas. Em determinadas vezes, após a reprodução de uma instância musical retornada não esperada pelo o usuário, e com os ouvidos atentos, é possível que o usuário identifique similaridades melódicas entre sua interpretação e a instância. Em outras vezes, mesmo ouvindo uma instância musical inteira, se torna difícil a compreensão por parte do usuário da discrepância entre a obra esperada e a instância respondida.

De qualquer forma, os sistemas não possuem nenhum apoio gráfico ou sonoro que explicite a similaridade melódica encontrada. A única informação de comparação apresentada pelo sistema da Google é uma taxa percentual de correspondência. Contudo, essa não releva nada sobre os fragmentos melódicos comparados. Mesmo considerando a ocasião em que uma instância da obra esperada foi retornada com sucesso, argumenta-se ser positivo um detalhamento sobre as comparações melódicas realizadas, ajudando o usuário a entender como foi executado o mapeamento de respostas. O aplicativo SoundHound possui uma transparência ainda pior que o aplicativo da Google, pois, em algumas ocasiões, não retorna resposta alguma, apresentando uma mensagem genérica de erro: “Aumente o volume da música e tente novamente.”

2.3.1.8. Conclusão da Análise de Similares

A partir da análise de sistemas similares realizada, tem-se o “Hum To Search” [5] da Google como principal referência de um serviço oferecido comercialmente para o grande público que implementa a *recuperação baseada em conteúdo melódico*, satisfazendo as qualidades de acurácia, performance, abrangência, diversidade de entrada, robustez de entrada e flexibilidade.

Com isso em mente, o projeto defende a oportunidade de construção de uma ferramenta de software que implementa a *recuperação baseada em conteúdo* através da *extração de padrões melódicos* e que tenha como diferencial a transparência dos fragmentos comparados.

Acredita-se que o desenvolvimento de funcionalidades projetadas para garantir boa transparência de comparação melódica seja relevante para pessoas relacionadas ao fazer musical, sejam elas compositoras, arranjadoras, produtoras, instrumentistas, professoras ou estudantes de música. Logo, todos

esses papéis compõem o público-alvo do projeto. Imagina-se o cenário, a ser validado, em que um compositor usa tais funcionalidades para buscar referências e ideias melódicas que complementem as suas próprias, impulsionando o estímulo criativo. A figura 3 ilustra o cenário descrito. Esse cenário foi brevemente simulado durante a pesquisa. A partir da entrada de uma obra composta para o estudo, o sistema da Google apresentou uma lista de instâncias musicais da obra *"Manhã de Carnaval"*, de Luiz Bonfá. Após uma pequena análise auditiva baseada em percepção musical, identificou-se similaridades entre os contornos melódicos de ambas as composições, o que motivou o compositor da obra de estudo a explorar novos caminhos melódicos.

Analisando o contexto contemporâneo, nos últimos meses, houve uma explosão de ferramentas que auxiliam a composição musical. Tais ferramentas, contrapondo o objetivo desse projeto, focam na geração de fragmentos melódicos inéditos por inteligência artificial. Por exemplo, Tchemeube *et al* [10] estuda a usabilidade de uma ferramenta do tipo embarcada na estação de trabalho de áudio digital (DAW) Cubase, no qual demonstra-se que tais ferramentas podem impulsionar o estímulo criativo do compositor e produtor musical.

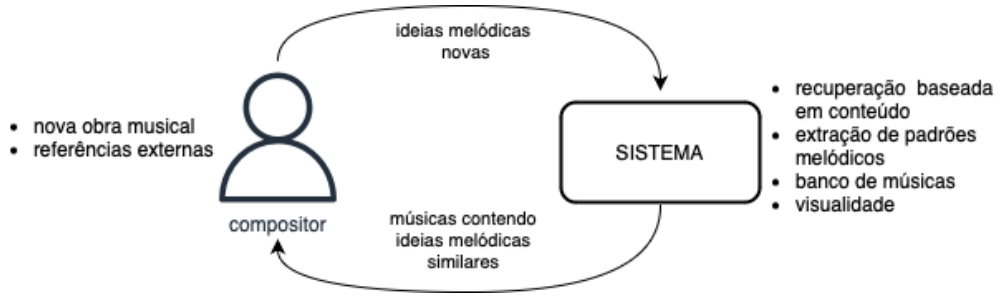

Figura 3 - Ferramenta de auxílio ao processo de composição musical.
Fonte: elaborada pelo autor.

Todo processo de expressão musical é complexo e distinto. Cada pessoa tem uma relação específica com sua própria musicalidade e não se deve tentar criar padrões. Entretanto, por experiência do autor, tem-se que durante o processo criativo musical é comum se encontrar preso e sem a sensação de progresso por falta de referências ou ideias novas. Nessa situação, ferramentas de auxílio criativo podem ser benéficas.

Dessa forma, a pesquisa encaminhada a seguir tem como objetivo encontrar a melhor abordagem computacional para a criação de um protótipo da

ferramenta de software anteriormente descrita. O protótipo deve procurar, no mínimo, garantir razoável performance, acurácia e transparência.

2.3.2. Principais Abordagens

Ao longo da pesquisa, foram encontrados artigos e referências que descrevem e propõem algoritmos para solucionar o problema de *extração de padrões melódicos* e, então, de *recuperação baseada em conteúdo*. Mediante suas particularidades, conseguiu-se agrupá-los em três principais abordagens.

Por alto, as três abordagens são baseadas no seguinte fluxo de desenvolvimento: recolher uma quantidade considerável de dados que representam instâncias musicais (arquivos de áudio ou outros) — considere-os o *dataset* do sistema; definir uma codificação intermediária dos dados que facilite seu processamento; definir uma função de distância que dita o quão similar uma instância é da outra; armazenar as instâncias do *dataset* processadas de forma a mapear suas distâncias; e adicionar nova entrada (gerada pelo usuário), retornando instâncias mais próximas.

Dentre esse fluxo, duas etapas são cruciais e vão caracterizar o tipo de abordagem usada: (1) a definição da função de distância*;* e (2) o mapeamento entre diferentes distâncias.

Sugere-se uma conotação ampla para a "função de distância", referindo-se, no caso, ao algoritmo utilizado para responder a pergunta: “dada as entradas melódicas *A* e *B*, quais são as características de *A* que a torna quantitativamente similar a *B* e vice-versa?”. Esse não é um problema trivial, visto que não existe um conceito musical formal para o grau metrificado de similaridade melódica. Mas, de forma geral, pode-se esperar que a função de distância proposta calcule a semelhança a partir dos contornos melódicos de cada fragmento. Ademais, é razoável considerar a normalização de variáveis como tom e andamento, visto que a transposição de tom e mudança de andamento são operações comuns em diferentes interpretações de uma mesma obra musical, preservando sua identidade original.

Ao passo que calcula-se os valores de distância de cada instância do *dataset*, é preciso armazená-las de forma a facilitar a posterior busca e comparação com uma nova entrada. A princípio, essa operação pode ser tão simples quanto percorrer todas as instâncias e comparar uma a uma. Porém isso, claramente, não é nada escalável, dado que o custo computacional seria linearmente proporcional ao tamanho do *dataset*, que precisa ser minimamente abrangente. A fundamentação teórica acessada propõe, então, estruturas de

dados que reduzem a complexidade de busca com base na definição da função de distância.

A seguir forneço uma síntese de cada uma das três abordagens, visto que os conceitos estudados são de vital importância para a construção do protótipo em questão.

2.3.2.1. Rede Neural

A primeira abordagem é, sem dúvida, a solução estado-da-arte do problema computacional discutido, pois se apropria de técnicas sofisticadas de aprendizado de máquina para a identificação de padrões melódicos. É provável que todos os sistemas atuais no mercado implementem variações dessa abordagem. Por ser minha principal referência, estudei a aplicação “Hum to Search” [5], da Google, que consegue reconhecer mais de 50 milhões de instâncias musicais. Apesar de não ser um projeto de software livre, a equipe de pesquisa da Google publicou oficialmente um pequeno artigo em seu blog que esboça, sem muitos detalhes, o funcionamento interno do sistema [8].

Em linhas gerais, utiliza-se uma grande rede neural que pode ser treinada para identificar os contornos melódicos sem que nenhuma heurística seja pré-definida. Ou seja, o próprio modelo consegue gerar uma função de distância ótima, após a etapa de treinamento. Para sua análise, o arquivo de áudio (representação original) de entrada é codificado em espectrograma, representação visual que indica relação de frequência por tempo. O modelo deve, então, aprender a destacar na imagem a melodia principal, não afetada por variáveis como timbre, ruídos externos, outros instrumentos e reverberação do ambiente de gravação. Para tal, a rede neural é treinada por pares de arquivos de entrada, nos quais há uma interpretação de determinada obra musical e uma instância musical da mesma. A rede neural, então, produz um *embedding* para o par, de forma a minimizar a distância de pares com melodia similar e maximizar a distância de pares com melodia diferente. Eventualmente, o modelo aprende a função de distância ótima. A figura 4 ilustra como o modelo enxerga as semelhanças no par de imagens de entrada.

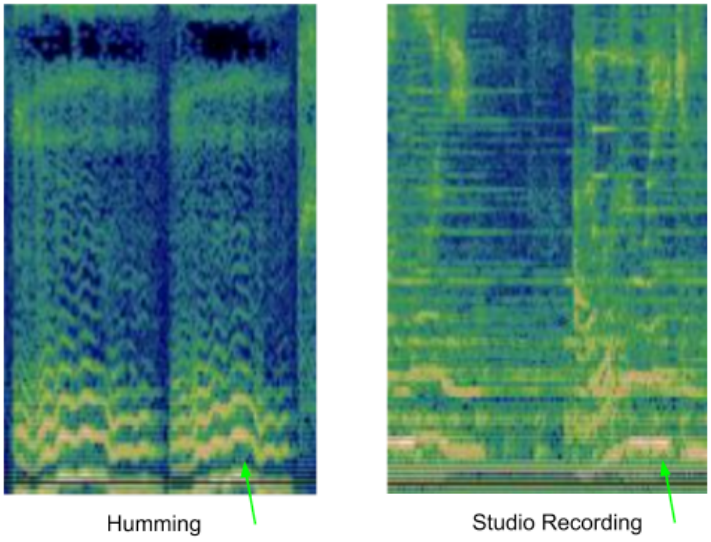

Figura 4 - Visualização de espectograma de par de entradas.
Fonte: Artigo da Google Research [8].

Essa abordagem contempla aspectos positivos como o fato de não precisar de grandes manipulações dos arquivos de áudio. Além disso, suspeita-se que o fato de o modelo não precisar de heurísticas pré-definidas ajuda a encontrar similaridades melódicas não previstas. A geração de *embeddings* também acelera a etapa de busca, podendo escalar, com a devida infraestrutura, para as dezenas de milhões de instâncias.

Contudo, considerando o escopo desse projeto, decidiu-se não seguir essa abordagem. O primeiro impeditivo seria conseguir um *dataset* de interpretações melódicas distintas suficientemente grande para a etapa de treinamento ser capaz de gerar a função de distância ótima. Além disso, acredito que a representação intermediária de espectograma não contempla uma boa transparência do algoritmo, pois não trata as notas como unidades discretas, dificultando a visualização dos intervalos presentes na melodia.

2.3.2.2. Comparação de Sequências e Programação Dinâmica

Em contraste ao aprendizado de máquina, a abordagem mais tradicional para o problema de extração de padrões melódicos é baseada na *teoria de comparação de sequências* [11]. Nesse tipo de abordagem, um fragmento melódico pode ser descrito por uma sequência (conjunto ordenado) de notas que possuem, no mínimo, dois atributos: altura e duração. É crucial, portanto, determinar uma interpretação intermediária que facilite a leitura nota por nota. Em algum ponto do processo de codificação da representação original para a

intermediária, esses dois atributos são normalizados com o efeito de ignorar alterações de tom e andamento.

Mongeau e Sankoff [2] propõem a seguinte heurística: dois fragmentos melódicos são considerados similares na medida em que suas notas são as mesmas — em altura e duração — e ocorrem na mesma ordem. Para efeito de exemplificação, pode-se considerar a seguinte simplificação do algoritmo. Sendo uma melodia *S1* = (*a, a, b, c*) e uma melodia *S2* = (*a, b, d*) — tal que cada letra representa uma nota específica, ou seja, uma tupla de valores de altura e duração — a distância, isto é, o grau de dissimilaridade entre as duas é proporcional à quantidade mínima de transformações em *S1* para se obter *S2*, ou vice-versa. Transformações básicas são do tipo: inserção, remoção e troca. No exemplo, de *S1* para *S2* pode-se realizar uma remoção do termo *a* e uma troca de *c* por *d*. A figura 5 ilustra o exemplo, na qual a linha vermelha representa a troca e o sublinhado vermelho representa a remoção. À cada tipo de transformação é associado um peso, de forma que a distância entre duas melodias é a soma ponderada das suas transformações mínimas. O peso de cada transformação também considera o intervalo e diferença de duração entre as notas. A distância mínima é computada através de um algoritmo de programação dinâmica, no qual para cada passo recursivo, a transformação de menor peso é escolhida, produzindo um caminho ótimo de escolhas de transformações.

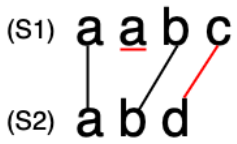

Figura 5 - Transformações de *S1* = (*a, a, b, c*) para *S2* = (*a, b, d*).
Fonte: elaborada pelo autor.

Rolland [3] se inspira bastante nos estudos de Mongeau e Sankoff. Ele também propõe um algoritmo de comparação de sequência e programação dinâmica, e ainda define alguns detalhes que ajudam a parte de engenharia de um sistema de *recuperação baseada em conteúdo*. O sistema, segundo ele, deve ser genérico o suficiente para aceitar diferentes representações intermediárias de uma melodia, acrescentando informações estruturais — chamadas de descrições — para além da altura e duração normalizadas de cada nota. Ele sugere experimentar com descrições globais (características relacionadas ao corpo melódico inteiro), descrições locais (relacionadas a uma passagem em particular) e descrições individuais (relacionadas a uma nota em particular). O uso composto de diferentes descrições resulta em mais ou menos

precisão de resposta, a depender da aplicação. A função de distância é similar à anterior, mas a ela são acrescentadas outros tipos de transformações. Rolland é mais detalhista na parte de mapeamento das distâncias, propondo um grafo de similaridade a partir do qual a busca *star center* é efetuada, isto é, dada uma instância analisada *pivot*, retorna-se instâncias similares, navegando pelas redondezas do grafo.

O estudo realizado conclui que, conceitualmente, essa abordagem é bem completa. A função de distância atua de forma a abraçar tanto variações nos intervalos quanto variações rítmicas, o que é bem interessante do ponto de vista musical. Além disso, pausas são bem definidas como notas sem altura, o que melhora ainda mais a precisão rítmica. Sobretudo, o tratamento de melodias como sequências de notas de altura e duração discretas possibilita uma possível funcionalidade de transparência das comparações.

Por outro lado, a abordagem requer que as sequências de entrada sejam representações fiéis da melodia em questão, isto é, não trata bem ruídos externos ou má interpretações pelo usuário (não garante robustez de entrada e flexibilidade). Para piorar, a acurácia depende de quanto se tem de informação sobre a melodia, sendo preciso um ajuste fino manual para ponderar os diferentes pesos da contribuição de cada intervalo e duração, além de outras variáveis adicionais como tom e andamento, informações essas cuja presença não pode ser assumida.

2.3.2.3. Histograma

A terceira e última abordagem pode ser entendida como uma simplificação da anterior. Liu, Wu e Chen [4] propõem melhorar a performance e diminuir a complexidade algorítmica da função de distância pela seguinte heurística: melodias são parecidas na medida em que a quantidade das ocorrências de suas alturas (ou intervalos) são iguais. Ou seja, não mais importa a ordenação e duração das notas. Dada uma sequência melódica de entrada, monta-se um histograma que dita a cardinalidade de cada nota em tal sequência. A distância entre duas sequências é, então, dada aproximadamente pela soma das diferenças absolutas das cardinalidades de cada nota. No exemplo anterior, o histograma de *S1* é {a: 2, b: 1, c: 1, d: 0} e de *S2* é {a: 1, b: 1, c: 0, d: 1}. Logo, a distância aproxima |2 - 1| + |1 - 1| + |1 - 0| + |0 - 1| = 3. A figura 6 ilustra os histogramas do exemplo. A grande vantagem em substituir a ordenação das notas pela construção do histograma é que esse possui um tamanho dimensional fixo, pois há no máximo 12 alturas discretas (ou intervalos ascendentes) que precisa-se representar, visto o efeito de percepção da oitava. Assim, os histogramas podem ser interpretados como pontos no espaço 12-

dimensional. Logo, o mapeamento proposto é uma *árvore R** modificada, que armazena e agrupa histogramas com alto grau de sobreposição.

À principio, descartar completamente a informação rítmica é contra-intuitivo, pois sabe-se que ela é de extrema importância para a compreensão melódica humana. Mas, ao comparar melodias bem distintas, essa simplificação parece ser precisa o suficiente, e Liu, Wu e Chen determinam uma série de parâmetros para fornecer mais controle sobre as comparações realizadas. Além disso, deve-se considerar que a simplicidade da função de distância torna sua implementação mais imediata, obtendo resultados mais imediatamente.

Por outro lado, uma implementação simplificada da abordagem pode não lidar tão bem com um par de sequências de diferentes tamanhos, visto que a diferença na ordem de grandeza de seus histogramas tornaria a comparação fora de escala. A abordagem também não menciona o tratamento de ruídos por imprecisão de áudio, e a acurácia de seus resultados foi testada somente a partir de arquivos MIDI.

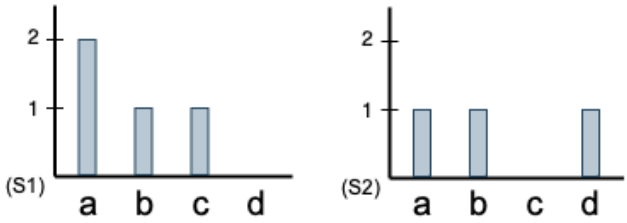

Figura 6 - Histogramas de *S1* = (*a, a, b, c*) e *S2* = (*a, b, d*).
Fonte: elaborada pelo autor.

3. Objetivos

A seguir, define-se os principais objetivos do projeto, que partem da pesquisa inicial na qual foram estudadas abordagens computacionais para a construção de um protótipo de ferramenta de software que explora a *recuperação de música baseada em conteúdo melódico* no contexto da oportunidade diferencial identificada, isto é, buscando prover aos músicos uma maior visualização de similaridades melódicas.

Para esse Projeto Final de Graduação, proponho (1) implementar e discutir a validade de diferentes abordagens computacionais para o problema de *extração de padrões melódicos* (2) levantar recursos externos disponíveis para apoiar o desenvolvimento do protótipo; (3) recolher uma quantidade suficiente de instâncias musicais de livre acesso para representar o *dataset* do protótipo; (4) produzir áudios locais com diferentes interpretações melódicas para validação interna do protótipo; (5) desenvolver protótipo de ferramenta de *recuperação baseada em conteúdo melódico monofônico* com foco na transparência das semelhanças entre fragmentos melódicos; e (6) conduzir testes externos para entender oportunidades de melhoria e receber feedbacks gerais.

Considerando o escopo do projeto, a avaliação final do protótipo será feita sobre apenas as qualidades de acurácia, performance e transparência. Ou seja, a proposta do protótipo não inclui maximizar a abrangência, flexibilidade, robustez de entrada ou diversidade de entrada do sistema.

No decorrer do projeto, arquivos de áudio e MIDI foram analisados. Grande parte dos arquivos para testes locais de interpretação melódica — cantados e tocados — foram produzidos pelo o autor, salvo o uso previamente autorizado de arquivos de terceiros, que serão referenciados explicitamente. Na produção autoral dos arquivos de teste, foi utilizada a estação de trabalho de áudio digital (DAW) Logic Pro [12]. Para testes de análise de arquivos de instâncias musicais, restringiu-se àquelas de livre acesso e disponíveis gratuitamente, respeitando seus direitos autorais.

A linguagem de programação escolhida para desenvolver o projeto foi Python, que possui suporte de uma infinidade de bibliotecas úteis de terceiros, das quais algumas, a serem especificadas, facilitaram substancialmente a implementação de código e construção do protótipo. Todo o código desenvolvido foi escrito e executado em máquina local, sem o uso de qualquer serviço nuvem.

O projeto não se propõe a ser um gerador musical e não tem a intenção de mimicar qualquer estilo artístico. O projeto tampouco procura se estender a aplicações de identificação de plágio em casos penais envolvendo direitos autorais.

4. Fundamentos

Esta seção fornece um detalhamento sobre os estudos preliminares realizados, os quais tiveram o foco na leitura, codificação e visualização de fragmentos melódicos. Além disso, procura-se aprofundar o estudo da fundamentação teórica acessada, apresentando implementações e discussões dos algoritmos de *extração de padrões melódicos* e de *recuperação baseada em conteúdo*. Por fim, são levantados recursos de software externos que serviram de apoio para a leitura de arquivos de áudio e MIDI.

4.1. Estudos Preliminares

Visando o desenvolvimento do protótipo e a compreensão dos obstáculos apresentados nos artigos estudados, realizou-se alguns testes na linguagem Python com a intenção de amadurecer a intuição primitiva sobre o problema.

Os estudos preliminares, então, têm o objetivo de produzir o código para ler um conjunto de arquivos de áudio com melodias simples e visualizar graficamente seus contornos melódicos. Para a leitura dos arquivos, utilizei o componente de mapeamento monofônico de frequências (a ser discutido posteriormente), obtendo uma lista de frequências calculadas a cada 0.05 segundos de áudio analisado. O modelo de visualização gráfica de melodias apresentado nessa seção não tem, à princípio, o propósito de ser acoplado ao protótipo final, servindo apenas como forma de estudo.

O primeiro arquivo de áudio analisado, *m1*, possui 6.5 segundos e contém a seguinte melodia: em durações constantes (de 0.5 segundos), com exceção da última (com 2 segundos), a sequência de notas C4, D4, E4, G4, E4, D4, C4, reproduzida por um teclado sintetizador. Nesse caso, não há interferência de ruídos externos, visto que o áudio foi convertido diretamente de um arquivo MIDI.

O desafio inicial se deu na codificação entre a representação original, uma grande lista de valores gerada pelo componente de mapeamento monofônico, e a representação intermediária, uma lista condensada de notas como tupla de altura e duração. Essa tarefa pertence ao codificador do sistema, que no momento possui uma implementação básica. No exemplo, *m1* gera 131 entradas de frequência e *timestamp*, que devem ser traduzidas para as 7 notas tocadas. A dificuldade está em determinar se variações na frequência (diferença absoluta entre frequências consecutivas) são oriundas da interpretação melódica ou são erro de precisão do mapeador monofônico. A primeira tentativa de contorno desse problema usa um valor limite para determinar se a variação de frequência foi intencional ou não. Determina-se que a variação foi intencional se passar do

valor limite. Entretanto, essa simples implementação ainda deixa passar imprecisões: particularmente, visto que o componente de mapeamento monofônico segue uma continuidade e sempre atribui frequências extremamente altas para as primeiras entradas, cria-se um problema de valor primário. Posteriormente, a solução para minimizar imprecisões é aprimorada, utilizando uma média de valores de frequência por intervalo de tempo.

Através da biblioteca Matplotlib [13], pode-se produzir o gráfico da lista condensada de notas, no qual o eixo x dita o tempo (em segundos), e o eixo y, a frequência (em Hertz). O gráfico descarta a informação de tempo antes do início da primeira nota e depois do início da última nota.

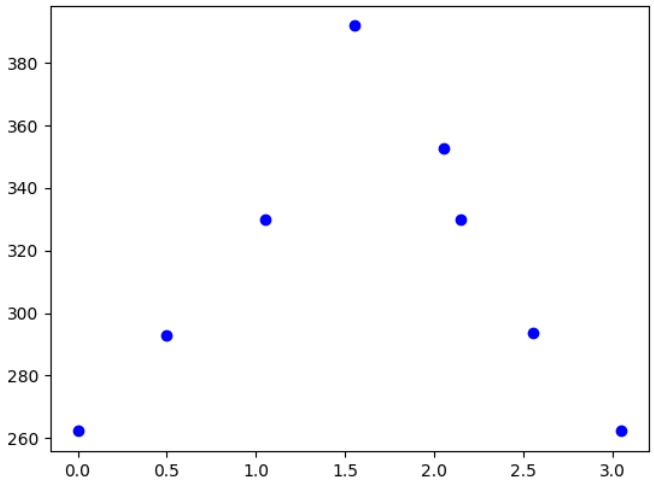

Figura 7 - Visualização (tempo x frequência) da melodia *m1*.
Fonte: elaborada pelo autor.

Na figura 7, pode-se perceber a imprecisão mencionada, pois o gráfico mostra que a lista condensada resultante contém uma nota intermediária (em torno de 350 Hertz, ou F4) entre G4 e E4, que não existe na interpretação melódica de *m1*.

Adiante, experimenta-se com a normalização de variáveis de tom e andamento, através de uma distruibuição linear — no intervalo [0.0, 1.0] — no domínio do tempo (soma total das durações) e da frequência (diferença entre a mais alta e a mais baixa). Assim, pode-se comparar visualmente apenas os contornos melódicos. Para efeito de teste, mais quatro áudios foram gerados de forma similar, ao passo que comparando-os a *m1*, tem-se:

- *m2*: tom diferente (mesmo instrumento, mesmo contorno melódico e mesmo andamento).

- *m3:* andamento diferente (mesmo instrumento, mesmo contorno melódico e mesmo tom).

- *m4:* pequena variação melódica (mesmo instrumento, mesmo andamento e mesmo tom).

- *m5:* grande variação melódica (mesmo instrumento, mesmo andamento e mesmo tom).

Para ajudar na visualização, constrói-se segmentos de reta ligando as alturas discretas consecutivas das notas de cada melodia.

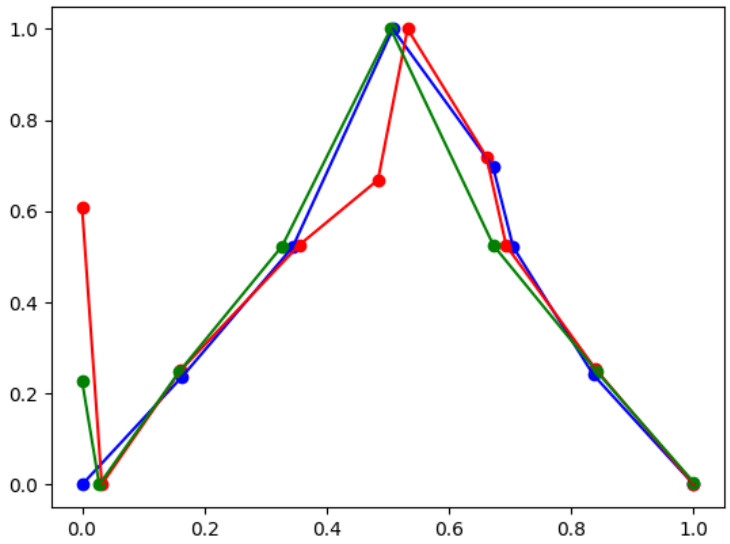

Figura 8 - Visualização dos contornos melódicos *m1* (azul)*, m2* (vermelho) *e m3* (verde).
Fonte: elaborada pelo autor.

Observando a figura 8, pode-se dizer de forma brusca que, apesar das imprecisões na leitura, as melodias *m1*, *m2* e *m3*, com andamento e tom diferentes, possuem contornos melódicos de mesmo "formato". Em contraste, ao adicionar à visualização as melodias *m4* e *m5*, que possuem intervalos e durações relativas diferentes, pode-se perceber a diferença em seus contornos (figura 9).

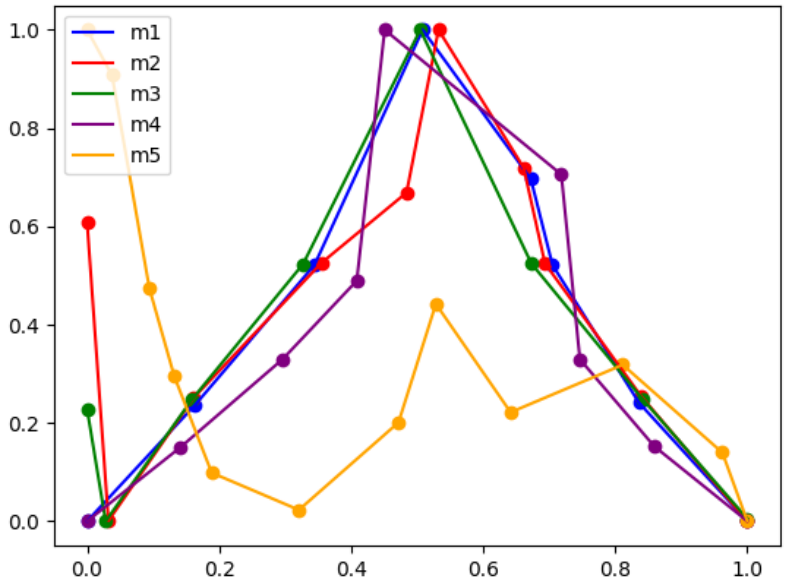

Figura 9 - Visualização dos contornos melódicos de *m1*, *m2*, *m3*, *m4* e *m5*.
Fonte: elaborada pelo autor.

Um resultado similar foi obtido na comparação de melodias interpretadas por voz cantada. Para efeito de teste, considerou-se a entrada de dois outros áudios, *m6* e *m7*, que contém interpretação melódica da melodia inicial de *"Asa Branca"*, por Luiz Gonzaga, cantados (com palavras) em tom e andamento diferentes. A figura 10 ilustra o resultado.

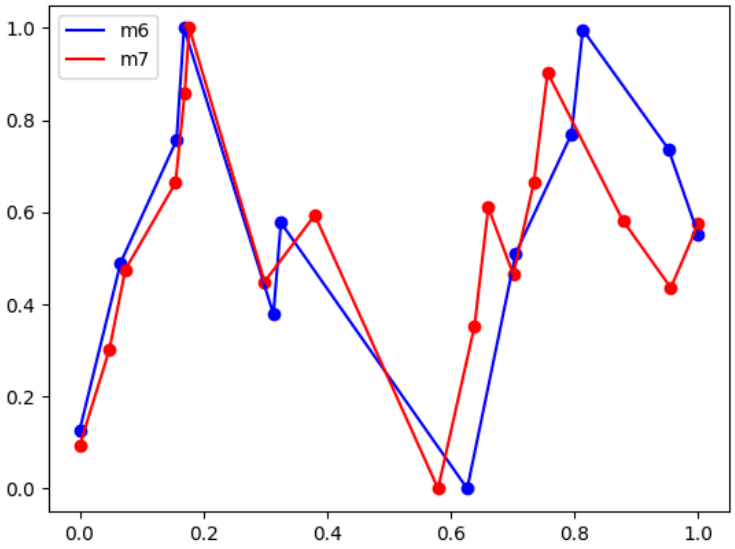

Figura 10 - Visualização dos contornos melódicos de *m6* e *m7*.
Fonte: elaborada pelo autor.

Essa exploração inicial da visualidade e transparência da *extração de padrões melódicos* ajuda a compreender como compará-los. Em particular, a ideia de observar o “formato" de cada melodia parte da noção intuitiva de que duas melodias *A* e *B* são dissimilares se, e somente se, *A* e *B* têm gráficos dissimilares, isto é, para todo instante *x* de tempo, o valor da altura *H(x)* de *A* tende a ser distante do valor *H(x)* de *B*. Em outras palavras, pode-se dizer que a função de distância entre duas melodias é proporcional à diferença da área embaixo de seus gráficos.

Uma função de distância baseada na heurística de diferença de área de gráfico, apesar de coerente, não é precisa o suficiente, se tornando ineficaz. Por exemplo, a inserção ou remoção de uma nota, intencionalmente ou por erro de leitura, afeta a comparação de altura de todas as notas seguintes, que estariam “desalinhadas”. Por outro lado, é desejável que a função de distância consiga observar semelhanças em fragmentos de diferentes partes do corpo melódico, de forma que uma simples variação de nota não seja relevante o suficiente para aumentar consideravelmente a distância total. O problema escala ao se considerar a entrada de fragmentos melódicos de diferentes tamanhos (em duração total ou quantidade de notas), já que a comparação entre um pequeno e um grande deve refletir o fato de que o primeiro pode estar contido no último.

Os artigos estudados apresentam estratégias mais maduras para comparar fragmentos melódicos de diferentes tamanhos. Portanto, o próximo passo foi tentar implementá-los.

4.2. Implementação das abordagens

Dentre as três abordagens descritas anteriormente — *rede neural*, *teoria de sequências* e *histograma* — somente as duas últimas foram, de fato, implementadas em código. Como discutido, a abordagem de rede neural seria muito fora do escopo do projeto, pois seu sucesso depende de um *dataset* muito grande. Além disso, o fato de que ela não possui uma heurística de comparação explícita é prejudicial para a transparência das comparações. Portanto, a abordagem não foi considerada para a construção do protótipo, apesar de ser a mais promissora em termos de performance, abrangência e robustez de entrada.

As duas outras abordagens, então, foram validadas através da implementação em código. Procurando descobrir qual delas teria mais potencial para garantir a transparência e a performance do protótipo, forma realizados uma série de testes ao longo do processo de desenvolvimento. Como discutido, a abordagem baseada na *teoria de sequências* é mais completa (no sentido de incluir mais detalhes), pois leva em consideração a ordenação entre as notas de uma sequência melódica e suas durações. Portanto, essa foi a primeira abordagem a ser implementada, seguindo a descrição inicial do algoritmo de Mongeau e Sankoff [2].

4.2.1. Estudo sobre Mongeau e Sankoff

Esta seção discutirá detalhes de implementação do algoritmo proposto por Mongeau e Sankoff [2]. Testes funcionais foram planejados e executados ao longo do estudo.

4.2.1.1. Versão Inicial

Os primeiros esboços de algoritmo de Mongeau e Sankoff para a *extração de padrões melódicos* seguem as linhas gerais da *teoria de comparação de sequência* [11], não se preocupando, nesse momento, com as particularidades do objeto de estudo melódico. Portanto, a versão inicial do algoritmo trata a sequência melódica simplesmente como uma lista de caracteres ou números inteiros, que em um segundo momento pode ser facilmente adaptada para uma lista de notas, dada uma codificação intermediária de melodias. Para esse nível de abstração, não são relevantes os detalhes de implementação dessas listas, que podem ser entendidas como listas dinâmicas ou simples arrays. A simplificação tem como objetivo a maior compreensão da recorrência e cuidado na implementação do algoritmo de programação dinâmica.

A chamada da função de distância, ou de dissimilaridade, $d(a, b)$, entre as sequências a e b retorna o valor associado à quantidade mínima de

transformações necessárias aplicadas a *a* para se obter *b*, que, por comutatividade, é igual ao valor retornado por *d*(*b*, *a*). As transformações permitidas nessa versão inicial são as de *inserção*, *remoção* e *troca*. Na implementação desenvolvida no projeto, os termos em uma sequência são inteiros e as transformações possuem pesos constantes, de forma que:

- *w*(ø, *x*): a inserção de um termo *x* qualquer tem peso 1.
- *w*(*x*, ø): a remoção de um termo *x* qualquer tem peso 1.
- *w*(*x*, *y*): a troca de um termo *x* qualquer por um termo *y*, tal que *y* é igual a *x* (mesmo inteiro), tem peso 0.
- *w*(*x*, *y*): a troca de um termo *x* qualquer por um termo *y*, tal que *y* é diferente de x (outro inteiro), tem peso 1.

Assim, a função de distância *d*(*a*, *b*, *i*, *j*) recebe quatro parâmetros: as duas listas, *a* e *b*, e seus dois respectivos índices, *i* e *j*. Como o algoritmo compara termo a termo, o índice de cada lista indica o último termo que foi comparado e a cada passo recursivo os índices decaem conforme a transformação selecionada. Com base no pseudocódigo apresentado nas figuras 9 e 10, $d_{i,\,j}$ denota o valor de distância calculado entre as sub-listas $a_{[0,\,i]}$ e $b_{[0,\,j]}$ (intervalos de início inclusivo e fim exclusivo). Para a chamada inicial, *i* e *j* devem expressar o tamanho das listas *a* e *b* respectivamente.

Mongeau e Sankoff descrevem o algoritmo através de uma recorrência, que contém três passos recursivos. No primeiro caso, a distância $d_{i,\,j}$ é igual à soma entre o peso da remoção do termo a_i e o valor de distância $d_{i\,-\,1,\,j}$ (chamada recursiva sem o termo a_i). No segundo caso, a distância $d_{i,\,j}$ é igual à soma entre o peso da troca do termo a_i pelo termo b_j e o valor de distância $d_{i\,-\,1,\,j\,-\,1}$ (chamada recursiva sem os termos a_i e b_j). No terceiro caso, a distância $d_{i,\,j}$ é igual à soma entre o peso da inserção do termo b_j e o valor de distância $d_{i,\,j\,-1}$ (chamada recursiva sem o termo b_j). Como procura-se achar o valor mínimo associado às possíveis transformações, após computar os valores para os três casos recursivos na chamada $d_{i,\,j}$, o menor dos três valores é escolhido. Na figura 11, o pseudocódigo sugerido pelos autores ilustra a recorrência proposta.

$$d_{ij} = \min \begin{cases} d_{i-1,j} + w(a_i, \text{ø}) \\ d_{i-1,j-1} + w(a_i, b_j) \\ d_{i,j-1} + w(\text{ø}, b_j) \end{cases}$$

Figura 11 - Pseudocódigo da recorrência da função de distância.
Fonte: Mongeau e Sankoff [2].

No artigo, também são especificados três casos base, que expressam as condições de parada recursiva. Na chamada $d_{i,0}$, a única transformação possível é a remoção de a_i. Na chamada $d_{0,j}$, a única transformação possível é a inserção de b_j. Por fim, na chamada $d_{0,0}$, não há transformações a serem aplicadas, retornando o valor 0. A figura 12 ilustra os casos base.

$$d_{i0} = d_{i-1,0} + w(a_i, \varnothing), i \geqslant 1;$$
$$d_{0j} = d_{0,j-1} + w(\varnothing, b_j), j \geqslant 1$$
$$\text{and}$$
$$d_{00} = 0,$$

Figura 12 - Pseudocódigo dos casos base da recorrência da função de distância.
Fonte: Mongeau e Sankoff [2].

Durante a implementação, deve-se perceber a importância de usar uma estrutura de dados matricial para armazenar os valores pré-calculados de $d_{i,j}$, para pares de *i* e *j* já visitados. Isso aumenta a performance consideravelmente, visto que os caminhos do grafo de chamadas recursivas muitas vezes se repetem. Uma estrutura similar pode armazenar, para cada chamada, a informação de qual transformação foi escolhida, o que possibilita retornar (e imprimir), ao final da execução, o caminho ótimo traçado pelo algoritmo.

Essa versão inicial do algoritmo de *extração de padrões* foi implementada na linguagem Python. Para efeito de validação, foi planejado um conjunto de testes com pares de sequências *a* e *b* acompanhados de valores de retorno *r* previstos para a função de distância. Após alguns ajustes na implementação, todos os testes foram bem sucedidos. O plano de testes contém chamadas para:

- T1: *a* = [1, 1, 2, 3]; *b* = [1, 1, 2, 3]; *r* = 0.
- T2: *a* = [1, 1, 2, 3]; *b* = [1, 4, 2, 3]; *r* = 1.
- T3: *a* = [1, 1, 2, 3]; *b* = [5, 1, 2, 3]; *r* = 1.
- T4: *a* = [5, 1, 2, 3]; *b* = [1, 1, 2, 3]; *r* = 1.
- T5: *a* = [1, 1, 2, 3]; *b* = [1, 1, 2]; *r* = 1.
- T6: *a* = [1, 2, 3]; *b* = [1, 1, 2, 3]; *r* = 1.
- T7: *a* = [1, 3]; *b* = [1, 1, 2, 3]; *r* = 2.
- T8: *a* = [1, 1, 2, 3]; *b* = [2, 1, 2]; *r* = 2.

O primeiro teste, T1, tem com objetivo validar a falta de transformação, ou em outras palavras, a presença da transformação identidade (troca de termos

iguais). T2 e T3 validam a transformação de troca. T4 valida a propriedade comutativa, pois retorna o mesmo resultado que T3. T5 valida a transformação de remoção. T6 valida a transformação de inserção. T7 valida a ocorrência de mais de uma transformação do mesmo tipo (inserção). T8 valida a ocorrência de mais de uma transformação de tipos diferentes (remoção e troca).

Para todos os testes, o caminho de execução também foi impresso. Por exemplo, a figura 13 demonstra o caminho tomada na execução da função de distância para o teste T8.

```
Dissimilarity between [1, 1, 2, 3] and [2, 1, 2]: 2
Debug Stack:
deletion of 3 -> +1
replacement of 2 with 2 -> +0
replacement of 1 with 1 -> +0
replacement of 1 with 2 -> +1
```

Figura 13 - Valor de distância e caminho ótimo de transformações na chamada de T8.
Fonte: elaborada pelo autor.

4.2.1.2. Adaptação para sequência melódica

Após o sucesso dos testes da versão inicial do algoritmo, o próximo passo foi substituir a lista de inteiros (simplificação) pela lista de notas, isto é, pela codificação intermediária dos arquivos musicais. Como discutido, uma codificação simples e eficaz é representar uma nota como uma tupla de altura e duração. A única mudança necessária para que a função de distância interprete o novo tipo de entrada se dá nos pesos de cada transformação. Em um primeiro momento, não considera-se a normalização de tom e andamento.

O artigo sugere que o peso entre as notas a_i e b_j seja calculado pela expressão

$$w(a_i, b_j) = w_{\text{altura}}(a_i, b_j) + k \,.\, w_{\text{duração}}(a_i, b_j)\text{ , na qual}$$

- $w_{\text{altura}}(a_i, b_j)$ é o peso associado a diferença de altura (intervalo) entre a_i e b_j;
- $w_{\text{duração}}(a_i, b_j)$ é o peso associado a diferença de duração entre a_i e b_j;
- k é uma constante que expressa o quão representativa é a contribuição de duração (ritmo) para a comparação geral melódica. Dependendo do estilo musical analisado, a comparação se torna mais ou menos eficaz dado um maior valor de k.

Mongeau e Sankoff fornecem um estudo aprofundado sobre os melhores pesos para cada intervalo e diferença de duração. Entretanto, para efeito de

simplificação, pode-se assumir que $w_{duração}(a_i, b_j)$ equivale a diferença absoluta entre ambas durações e, analogamente, $w_{altura}(a_i, b_j)$ é proporcional a diferença entre ambas alturas. Contudo, é interessante multiplicar $w_{altura}(a_i, b_j)$ pela menor duração entre a_i e b_j, para que a diferença de altura não seja tão significativa em notas ornamentais de pequena duração. Sabendo calcular o peso $w(a_i, b_j)$, já é possível realizar transformações de troca. Os pesos de inserção e remoção, em seguida, se tornam simplesmente *k* vezes o valor de duração da nota inserida ou removida.

Para testar a nova versão do algoritmo, reutilizou-se a leitura e codificação dos arquivos de áudio produzidos durante os estudos preliminares, dentre os quais, *m1* deve se mostrar semelhante a *m2* e a *m3*, e dessemelhante a *m4* e, sobretudo, a *m5*. O novo conjunto de testes proposto não mais pode prever o valor resultante de distância, pois o cálculo à mão se torna impraticável. Entretanto, pode-se validar a implementação se, para *a* = *m1*, os valores de distância para *b* = *m2* e para *b* = *m3* forem menores que para *b* = *m4* e para *b* = *m5*. O resultado da rodada de testes, para *k* = *0.1*, foi:

- T9 (*a* = *m1*, *b* = *m2*): 0.6
- T10 (*a* = *m1*, *b* = *m3*): 1.0
- T11 (*a* = *m1*, *b* = *m4*): 0.15
- T12 (*a* = *m1*, *b* = *m5*): 0.9

A princípio, os resultados não foram conclusivos. Esperava-se que o valor de retorno de T9 e T10 tendesse a zero, que T11 retornasse valor um pouco maior que os anteriores e que T12 retornasse o maior valor de todos. Entretanto, nesse momento ainda não fora aplicadas as normalizações de tom e andamento. Logo, é razoável a imprecisão da função de distância, que considera mudanças absolutas entre as notas.

4.2.1.3. Normalização

Dada a codificação intermediária de notas como tupla do tipo (*nota.altura*, *nota.duração*) já discretizadas, pode-se aplicar as normalização de tom e andamento comprimindo a sequência melódica, *a,* de *n* notas em uma sequência nova, *a'*, com *n - 1* notas, de forma que

$$(a'_i).altura = (a_i).altura - (a_{i+1}).altura, \text{ e}$$
$$(a'_i).duração = (a_i).duração / (a_{i+1}).duração$$

Assim, a lista resultante contém notas relativas, pois, semanticamente, a altura da nota expressa um intervalo (diferença entre alturas consecutivas) da lista original e, de forma similar, a duração expressa a razão entre durações consecutivas da lista original. Essa forma de codificação é uma das muitas formas de descrição melódica mencionadas por Rolland [3].

Após essa alteração na implementação, e executando novamente o conjunto de testes, para *k = 0.1,* tem-se:

- T13 (*a* = *m1*, *b* = *m2*): 0.0
- T14 (*a* = *m1*, *b* = *m3*): 2.35
- T15 (*a* = *m1*, *b* = *m4*): 6.15
- T16 (*a* = *m1*, *b* = *m5*): 9.9

Analisando comparativamente os resultados, tem-se que a normalização aumentou a acurácia da função de distância. T13 retornou zero, como esperado. T15 retornou valor maior e T16 foi o maior valor de todos. O único imprevisto foi em T14, que também deveria retornar um valor em torno de zero. Porém, recorrendo ao caminho executado, foi averiguado que o acréscimo no valor final se deu por conta da imprecisão na leitura do arquivo de áudio (pelo componente de mapeamento monofônico ou pelo codificador), que acabou atribuindo um intervalo maior que o esperado, modificando a sequência melódica. A figura 14 demonstra o ocorrido. Mesmo considerando o erro de imprecisão na leitura do arquivo, o valor final de T14 ainda foi menor que T15 e T16.

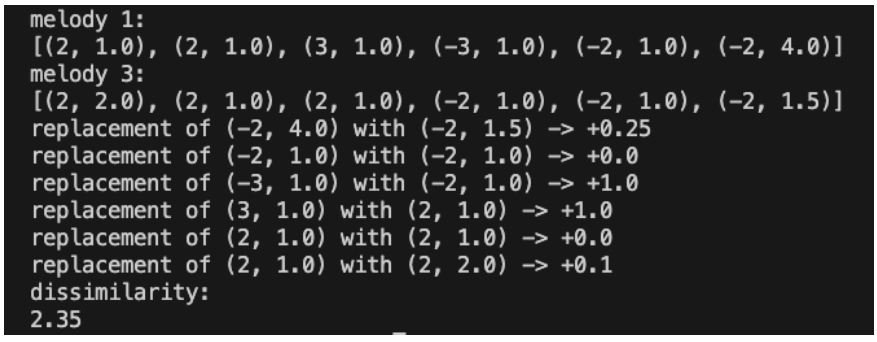

Figura 14 - Valor de distância e caminho ótimo de transformações na chamada de T14.
Fonte: elaborada pelo autor.

Outros valores de constante *k* foram experimentados, mas como, de forma geral, assume-se que variações nos intervalos são mais significativas que nas razões de duração, melhores resultados são obtidos com o valor baixo de 0.1.

A média de tempo de execução dos testes T9 a T16 (sem considerar o tempo de leitura dos áudios) foi de aproximadamente 0.015 segundos. Com isso, pode-se afirmar que para sequências com tamanho na ordem de dezenas de notas, o algoritmo garante boa performance.

4.2.1.4. Conclusão do Estudo sobre Mongeau e Sankoff

Para a extração de padrões melódicos, a abordagem, explorada através do artigo de Mongeau e Sankoff, parece bastante promissora. Além de ignorar mudanças de tom e andamento, o algoritmo de *extração de padrões melódicos* implementado consegue comparar, em tempo razoável, sequências melódicas de tamanhos diferentes, identificando pequenas e grandes mudanças de altura e duração no contorno melódico. O principal motivo que torna a abordagem uma boa escolha para a construção do protótipo está no fato de que a forma com a qual pode-se recuperar o caminho ótimo de transformações já é, por si só, uma boa transparência da funcionalidade das comparações, podendo ser facilmente apresentada de forma gráfica.

Por outro lado, o estudo proposto por Mongeau e Sankoff não tem foco no problema maior *de recuperação baseada em conteúdo*, faltando detalhes sobre a implementação da etapa de busca. De fato, esse problema é melhor explorado por Rolland [3], que descreve maiores aplicações para a *extração de padrões melódicos*, seguindo ainda o embasamento na *teoria de comparação de sequências*. Entretanto, em ambos os casos, pode-se observar como a etapa de busca não é trivial para a abordagem, visto que a função de distância compara as sequências mediante a entrada de uma sequência alvo (ou *pivot*), isto é, não há uma forma global de comparação que mapeie as diferentes distâncias.

Durante os estudos, descobriu-se que a abordagem de histograma possui uma vantagem sobre a de sequência, já que o próprio histograma já é uma representação global de distância. A seguir, os detalhes da implementação do algoritmo proposto por Liu, Wu, Chen [4] serão fornecidos.

4.2.2. Estudo sobre Liu, Wu e Chen

O algoritmo proposto por Liu, Wu, Chen [4] almeja otimizar o algoritmo descrito por Rolland (e, por consequência, o descrito por Mongeau e Sankoff). Além disso, afirma-se que o diferencial do algoritmo está em seu foco de mapear padrões melódicos com algum nível de dessemelhança, isto é, não apenas encontrar recorrências exatas de um fragmento melódico. Para tal, o conceito de *melodia prototípica* é introduzido, isto é, um padrão melódico com grande relevância dentro de uma obra musical, ao qual encontra-se uma alta quantidade

de fragmento melódicos similares. Assim, o algoritmo proposto é utilizado para extrair as *melodias prototípicas* de uma obra musical.

4.2.2.1. Distância de Histogramas

Como o artigo recomenda explicitamente o uso do formato MIDI para os dados de entrada, foram realizados testes através do componente de leitura de arquivo MIDI (a ser discutido posteriormente). A partir da leitura MIDI de um fragmento melódico, pode-se criar uma lista de valores de altura de notas em sequência, desprezando a informação de duração de cada nota.

A partir da lista de alturas *a* — ou de intervalos, considerando a normalização de tom (diferença de alturas consecutivas) — é possível construir o histograma *HV(a)*. Cada termo único na lista, isto é, um mesmo elemento com uma ou mais ocorrências, é uma dimensão do histograma. Em Python, o histograma *HV(a)* pode ser implementado pela estrutura de dicionário (similar a um *hash table*), cujas chaves são os termos únicos pertencentes a *a*, e os valores, indexados pelas chaves, são a cardinalidade (quantidade de ocorrência) de cada termo único.

Uma transformação de inserção na lista é representada pelo acréscimo de uma unidade no histograma (no termo correspondente). Portanto, pode-se medir a semelhança entre as listas apenas pelas transformações de inserção, quebrando a propriedade de comutatividade da função de distância. Logo, dado um par de sequências *S1* e *S2, e* seus histogramas *HV(S1)* e *HV(S2), a* distância (de histograma) entre ambas é dada pelo maior valor entre a quantidade de inserções necessárias de *HV(S1)* para *HV(S2)* e a quantidade de inserções necessárias de *HV(S2)* para *HV(S1)*. As figuras 15 e 16 ilustram o pseudocódigo para o cálculo proposto.

$$HD(S_1,S_2)=max(ins(HV(S_1), HV(S_2)), ins(HV(S_2),HV(S_1)))$$

Figura 15 - Cálculo do valor de distância de histograma entre *S1* e *S2*.
Fonte: Liu, Wu, Chen [4].

$$ins(HV(S_1), HV(S_2)) = \sum_{i=1}^{|\Sigma_D|} d_i, where \quad d_i = \begin{cases} h_i^{S_2} - h_i^{S_1} & if \quad h_i^{S_2} > h_i^{S_1} \\ o & otherwise \end{cases}$$

Figura 16 - Cálculo da quantidade de inserções de *HV(S1)* para *HV(S2)*.
Fonte: Liu, Wu, Chen [4].

Um pequeno conjunto de testes foi planejado e executado, validando a função de distância *HD(a, b)* implementada. Para cada par de sequências mocadas *a* e *b*, um valor de retorno *r* esperado foi retornado:

- T1: *a* = [1, 0, 0, 1, 1]; *b* = [1, 1, 0, 0, 1]; *r* = 0.
- T2: *a* =[1, 0, 0, 1]; *b* = [1, 0, 0, 1, 1]; *r* = 1.
- T3: *a* = [1]; *b* = [1, 1, 1, 2, 1]; *r* = 4.
- T4: *a* = [1, 0, 0, 1]; *b* = [-1, 2, 3, -1, -1]; *r* = 5.
- T5: *a* = []; *b* = [-1, 2, 3, -1, -1]; *r* = 5.

O primeiro teste, T1, tem com objetivo validar a indiferença na ordenação dos termos. T2 valida uma inserção. T3 e T4 validam mais de uma inserção. Finalmente, T5 valida o funcionamento para sequências vazias.

4.2.2.2. Espaço de Histogramas e Árvore R*

A grande vantagem dessa abordagem é que, apesar de possuir uma forma de comparar sequência uma a uma, a própria representação em histograma já é uma aproximação global da distância entre sequências, pois dado dois histogramas *HV(a)* e *HV(b)* com mesmas dimensões, a diferença *HV(a) - HV(b)* é aproximadamente o valor de *HD(a, b)*. Assim, Liu, Wu, Chen sugerem interpretar histogramas como pontos no espaço *x*-dimensional, onde *x* é a dimensionalidade (quantidade total de dimensões) dos histogramas, e armazená-los em uma estrutura semelhante a uma árvore R* [14]. Essa é uma estrutura de dados que realiza a clusterização de pontos espaciais em caixas contidas, podendo acessar pontos sem precisar percorrê-los todos.

Em Python, foram realizados alguns testes com a biblioteca rtree [15], que implementa a árvore R (estrutura similar à árvore R*). A biblioteca possibilita inserir objetos pontos (coordenadas) e objetos caixas (intervalo de coordenadas) de dimensão *x* especificada, e fazer requisições, de forma otimizada, como encontrar todos os objetos contidos em uma caixa especificada ou encontrar o objeto mais próximo de um ponto especificado. A biblioteca também permite salvar a árvore com os objetos inseridos em um arquivo local.

Para efeito de teste, considerou-se uma árvore 12-dimensional, tomando doze como a dimensionalidade máxima dos histogramas de sequências melódicas. Tal consideração é correta, pois, embora a altura discreta de notas MIDI variar do valor 21 ao 127, pelo efeito de percepção da oitava cada nota é identificada como um entre doze semitons únicos. Analogamente, considerando a normalização de tom, pode-se assumir que só existem doze intervalos ascendentes possíveis, pois intervalos maiores alcançam as mesmas notas que

os primeiros doze. Além disso, intervalos descendentes podem ser convertidos em intervalo ascendentes, considerando, novamente, que atingem as mesmas doze notas. Logo, uma segunda menor descendente equivale a uma nona maior ascendente, uma segunda maior descendente equivale a uma nona menor ascendente e assim por diante. Computacionalmente, dadas duas altura *x* e *y,* o intervalo resultante entre *x* e *y* é dado por *x* - *y mod* 12.

4.2.2.3. Conclusão do Estudo sobre Liu, Wu, Chen

O estudo de implementação do algoritmo de Liu, Wu, Chen não foi tão conclusivo quanto o esperado. A codificação em histograma e armazenamento em árvore R*, sem dúvidas, é uma boa maneira de garantir perfomance na etapa de *recuperação baseada em conteúdo*. Além disso, trabalhar com arquivos MIDI tornou o processo de testagem bem mais fácil, já que não precisa-se considerar imprecisões na leitura. Entretanto, o algoritmo começou a revelar problemas na comparação de sequências grandes com fragmentos menores (contidos), na qual os valores de distância são em ordens de grandeza diferentes. Honestamente, a implementação não considerou todos os detalhes discutidos no artigo e, nesse sentido, ficou incompleta. Entretanto, procurou-se resolver o problema dividindo as sequências melódicas de tamanho grande em partes menores antes de dar entrada no algoritmo. Isso foi feito analisando os valores de mensagem dos arquivos MIDI, sobre os quais é possível extrair a informação de tempo (relativo) entre as notas. Ou seja, buscou-se realizar cortes na sequência maior nos momentos de silêncio (pausa) de maior duração. Mesmo assim, os resultados não foram conclusivos.

Ademais, comparando-a com a implementação de Mongeau e Sankoff, a extração de padrões melódicos de Liu, Wu, Chen tem menos controle e não apresenta uma boa transparência das comparações, visto que só é retornado a diferença das cardinalidades entre notas.

A seguir, serão detalhados os estudos sobre tratamento de arquivos de áudio e MIDI, que foram cruciais para possibilitar a leitura de instâncias do *dataset* e de interpretações melódicas.

4.3. Estudos em Leitura e Tratamento de Áudio

Nesta seção são levantados recursos externos (tecnologias disponíveis gratuitamente) para apoio no processamento de informação melódica contida em arquivos de áudio. Os componentes a seguir podem ser entendidos como arquivos Python (.py) que acessam e encapsulam as bibliotecas discutidas.

4.3.1. Componente de Mapeamento Monofônico

No contexto da *recuperação baseada em conteúdo*, é interessante possibilitar a leitura de informação melódica (frequência por tempo) em arquivos de áudio, tanto para a análise de instâncias musicais quanto para a análise de interpretações melódicas do usuário. Infelizmente, diferentemente de um arquivo MIDI, um arquivo de áudio (.wav, .m4a, .mp3, etc) não contém descrição musical. Pode-se, entretanto, com alguma imprecisão, calcular a frequência fundamental presente em cada instante de tempo do áudio. Visando o escopo do projeto, em vez de desenvolver um componente de leitura de áudio do zero, foram pesquisados recursos externos disponíveis que implementam o mapeamento monofônico de frequência (*monophonic pitch tracker*).

A biblioteca CREPE [16], em Python, através de uma rede neural convolucional, disponibiliza uma chamada que recebe um arquivo de áudio no formato WAVE (.wav) — não aceitando outros formatos — e retorna uma tabela (.csv) contendo três colunas: tempo, frequência e confiança. Dentre seus argumentos, pode-se especificar o *step_size* (intervalo de tempo, em milissegundos) a ser utilizado. Assim, a cada *step_size* milissegundos, um valor de frequência fundamental (em Hertz) é estimado com algum grau de confiança, isto é, uma probabilidade no intervalo [0, 1), gerando uma entrada na tabela. Há também como especificar o tamanho do modelo pelo argumento *model_capacity*, isto é, o nível de balanço entre precisão e tempo de processamento. Para todos os testes, foi utilizada a capacidade “full”, ou seja, a maior precisão.

O modelo da CREPE não funciona em tempo real, isto é, enquanto o áudio está sendo capturado. Portanto, deve-se levar em consideração o tempo de processamento do arquivo de áudio, que varia bastante de acordo com o valor de *step_size,* além do valor de *model_capacity.* A escolha de um valor pequeno de *step_size* significa que muitas frequências serão estimadas ao longo da duração do áudio, aumentando a quantidade de informação extraída, mas também aumentando o tempo total de processamento. Por exemplo, escolhendo o valor de 50 milissegundos e analisando um arquivo de áudio de apenas 15 segundos, o tempo de processamento é de 6.2 segundos. Por outro lado, aumentar consideravelmente o valor de *step_size* afim de reduzir o tempo de

processamento é arriscado, já que variações melódicas (mudanças de altura) de pequena duração podem não ser capturadas. Os estudos preliminares, citados anteriormente, foram conclusivos para determinar um valor equilibrado de *step_size.* O valor final utilizado foi de 50 milissegundos.

Mesmo com a gravação de áudio mais limpa (livre de ruídos) e interpretação melódica melhor executada, não é razoável esperar que a extração de frequências seja uma representação fiel da melodia original. Quando se trabalha com áudio, sempre há imprecisão. Particularmente, a avaliação do modelo é fraca quando ocorre durante uma mudança de notas, muitas vezes estimando frequências intermediárias não intencionais. Para cada frequência estimada, o valor de confiança dita o grau de certeza do modelo sobre essa estimativa. Em alguns casos, esse mecanismo é eficaz, já que pode-se definir um valor limite — nos testes, foi utilizado o valor 0.9 — para filtragem, de forma a descartar as entradas com pouca confiança. Entretanto, nem sempre esse filtro é suficiente, deixando passar frequências indesejadas.

O filtro pela confiança também pode ser útil para descartar silêncios, já que para momentos nos quais não há notas sendo tocadas, o modelo tende a registrar uma confiança bem baixa. Contudo, silêncios com baixa duração, isto é, pequenas pausas entre as notas, não são bem representados. Logo, a precisão rítmica também é baixa, o que pode ser extremamente prejudicial, dependendo da abordagem de *extração de padrões melódicos* a ser utilizada. Para solucionar a falta de precisão rítmica, acredito ser aconselhável o uso de um recurso de mapeamento de volume. Esse componente será discutido mais a frente.

Por último, deve-se observar que o modelo de mapeamento de frequências da biblioteca CREPE é monofônico, o que significa que seu funcionamento é limitado a áudios com somente uma voz melódica. Isso não é um problema para a análise de interpretação melódica geradas pelo usuário. Porém, para analisar instâncias musicais, que pode conter vários instrumentos melódicos, harmônicos e de percussão, além de outros efeitos de produção, é necessário o uso de um recurso de isolamento de instrumento. Com esse, é possível escolher uma voz musical responsável pela melodia principal e gerar um novo áudio contendo apenas as frequências respectivas à voz. Esse componente será discutido mais a frente.

Uma das vantagens de usar um recurso de mapeamento monofônico é a possibilidade de extrair informação melódica independentemente do timbre da fonte sonora. Ou seja, funciona para canto, assobio ou qualquer outro instrumento melódico. A partir dos estudos preliminares, nos quais foram testados melodias em áudio geradas por teclado sintetizador e por canto, a

biblioteca CREPE foi validada como o componente de mapeamento monofônico do protótipo.

4.3.2. Componente de Conversão WAVE

Como o componente de mapeamento monofônico através da biblioteca CREPE apenas lê arquivos no formato WAVE (.wav), em alguns momentos durante o projeto foi necessário a conversão de formato de áudio. A biblioteca pydub [17] possui chamadas que realizam a conversão de arquivo MP3 (.mp3) para WAVE (.wav). Durante os estudos, foi verificado que possíveis perdas de qualidade entre os formatos devidas a mudanças de compressão de áudio não afetam a leitura de informação melódica.

4.3.3. Componente de Mapeamento de Volume

Embora não tenha sido utilizado durante os estudos preliminares e na implementação dos algoritmos, um componente de mapeamento de volume pode ser acoplado ao sistema para aprimorar a leitura de informação melódica em arquivo de áudio, adicionando precisão rítmica ao componente de mapeamento monofônico de frequências. Por exemplo, frequências estimadas em instantes de tempo de baixo volume (próximo ao silêncio) podem ser descartadas. Foram realizados alguns testes com o módulo wave da biblioteca padrão de Python, que possui chamadas para leitura de volume de um arquivo WAVE (.wav), nos quais se demonstrou possível mapear o início e fim de notas. A figura 17 demonstra a distribuição de volume (amplitude) pelo tempo para um arquivo de áudio teste, através do módulo e da biblioteca Matplotlib [13].

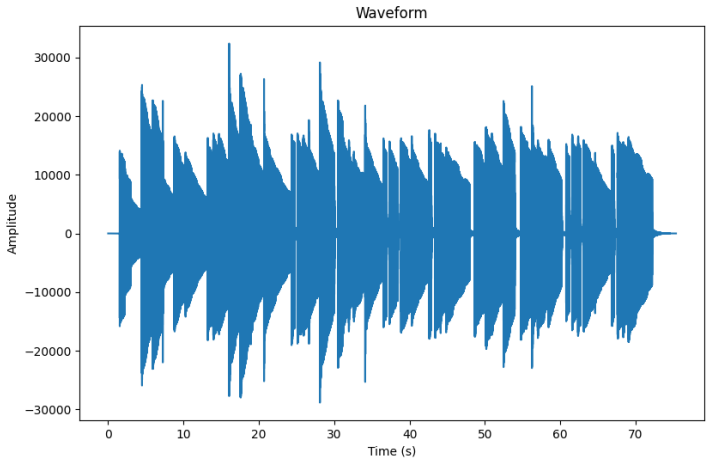

Figura 17 - Gráfico (volume x tempo) extraído de um arquivo de áudio.
Fonte: elaborada pelo autor.

4.3.4. Componente de Isolamento de Instrumento

A leitura de informação melódica de áudio de instâncias musicais com muitos instrumentos é uma tarefa extremamente difícil. Nem sempre a melodia principal de uma obra se encontra em uma única voz musical (instrumento melódico), a medida que várias vozes podem se misturar e complementar umas às outras em diferentes momentos. Entretanto, uma solução razoável é assumir que um instrumento único contém tal melodia e, então, tentar isolar seu áudio. Por exemplo, em grande parte da produção musical contemporânea, a melodia principal está contida no vocal. Portanto, é interessante a possibilidade de um componente que realiza esse isolamento, dado um tipo de instrumento especificado, como a voz humana. Felizmente, existem algumas opções de recursos externos que implementam a separação e isolamento de vozes. Porém, deve-se atentar ao fato de que esse tipo de procedimento trabalha com uma taxa de imprecisão muito alta, e muitas vezes o áudio isolado contém ruídos indesejados, ou não registra fragmentos melódicos importantes. Além disso, não pode-se garantir que os áudios isolados conterão melodias monofônicas.

O recurso levantado foi a biblioteca Demucs, disponibilizada por parte da equipe de pesquisa Meta AI [18]. Dentre outras opções levantadas, essa foi a de mais fácil uso. A biblioteca possui uma chamada para separar um arquivo de áudio especificado, podendo especificar seu formato e qual instrumento a ser isolado. Durante a pesquisa, foi testado o isolamento de vocal (voz humana).

4.4. Estudos em Leitura de MIDI

MIDI [19] é um padrão de comunicação que possibilita registrar o que deve ser tocado por um instrumento digital, cuja qualidade sonora de timbre não é determinante. Diferentemente de arquivo de áudio, que expressa ondas sonoras e pode captar uma instância musical, o formato de arquivo MIDI (.mid) é, essencialmente, uma descrição digital de um arranjo musical. Dentre outros parâmetros que podem ser especificados, o arquivo MIDI pode conter várias faixas (tracks) que representam diferentes instrumentos e, por sua vez, podem conter vários eventos de nota. Um evento de nota é uma abstração capaz de indicar o início ou fim de uma nota, com altura e tempos discretos, o que torna capaz a leitura melódica ao longo do tempo.

4.4.1. Componente de Leitura de Arquivo MIDI

Em Python, a biblioteca Mido [20] contém abstrações que possibilitam a leitura de arquivo MIDI. Especificando o nome ou índice de uma faixa que contenha a melodia principal, é possível ler seus eventos de nota e agrupá-los em uma lista.

A leitura de arquivos MIDI criados por terceiros acarreta o seguinte desafio: não existe um padrão para ditar qual faixa contém a melodia principal da obra representada no arquivo. Para piorar, a melodia pode estar distribuída em mais de uma faixa. Portanto, se faz necessário uma análise manual prévia, dado o conhecimento à priori sobre a obra musical, dos arquivos MIDI a serem usados como entrada no sistema. Foram experimentadas algumas formas de solução computacional para essa identificação — por exemplo, foi testada a verificação de palavras chaves, como “melody”, “lead”, “solo”, nos nomes de cada faixa — mas os resultados não foram conclusivos, visto que, mais uma vez, não pode-se garantir que o criador do arquivo siga qualquer padrão.

Também deve-se atentar ao fato de que as faixas podem ser não-monofônicas, isto é, podem conter mais de uma nota por tempo. Uma etapa de pré-processamento pode separar uma faixa não-monofônica em *n* faixas monofônicas, tal que n é o número máximo de notas simultâneas analisadas. Como é muito difícil ter noção sobre a intenção melódica presente na faixa não-monofônica, é razoável determinar que a separação seja feita por altura. Por exemplo, na separação de uma faixa *x* para duas faixas *x′* e *x″*, ao ler a faixa *x* original, sempre que se encontrar notas simultâneas, a mais grave é movida para *x’* e a mais aguda para *x”*. A mesma idéia pode ser extrapolada para a separação em *n* faixas. A partir das *n* faixas monofônicas separadas, pode-se

considerar que a melodia principal está contida em uma ou mais delas. A figura 18 ilustra o exemplo para *n* = 2.

A biblioteca Mido foi validada como componente de leitura de arquivos MIDI, pois suas chamadas são inteligíveis e possuem boa documentação.

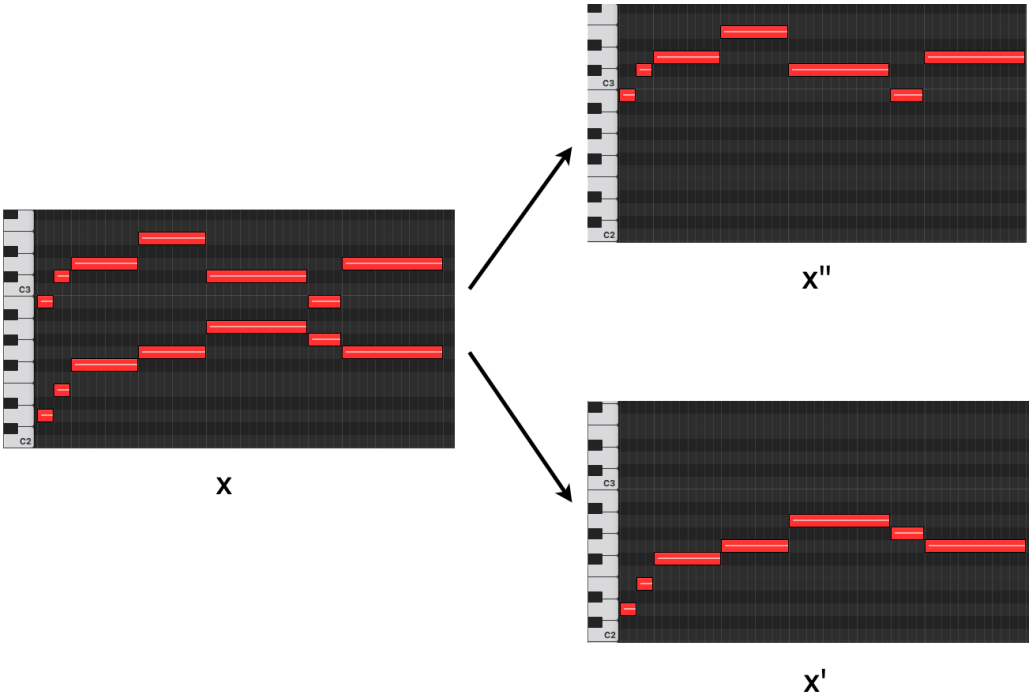

Figura 18 - Separação de faixa não monofônica em duas faixas monofônicas.
Uso da DAW Logic Pro [12] para ajudar na visualização MIDI.
Fonte: elaborada pelo autor.

5. Projeto e Especificação do Sistema

A partir dos estudos técnicos de implementação das abordagens de *extração de padrões melódicos* e *recuperação baseada em conteúdo*, assim como do levantamento de recursos auxiliares para o tratamento de arquivos de áudio e MIDI, pode-se finalmente definir o protótipo a ser desenvolvido.

5.1. Fluxogramas

O sistema protótipo contém dois fluxos de execução distintos: (1) o de treinamento; e (2) o de recuperação. O fluxo de treinamento ocorre sem a entrada de usuário. Basicamente, é o momento no qual ocorre a *extração de padrões melódicos* das instâncias musicais presentes no *dataset* do sistema. Sem o treinamento, é impossível executar a recuperação, visto que o sistema não teria nenhum conhecimento melódico com o qual comparar. Já o fluxo de recuperação é quando, de fato, ocorre a busca através da entrada de interpretação melódica pelo usuário, comparando-a com as instâncias previamente analisadas e retornando aquelas com menor distância de forma a garantir transparência.

O sistema protótipo contém três grandes componentes que funcionam em processos separados e são peças fundamentais em ambos os fluxos. O diagramas das figuras 15 a 17 fornecem uma visão ampla da abstração de funcionalidades do sistema protótipo.

Sugere-se a abstração dos três seguintes processos independentes: (1) leitura de arquivo áudio ou MIDI; (2) extração de padrões melódicos; e (3) busca de instâncias musicais. Cada processo possui componentes que realizam sub-tarefas (ilustrados nos diagramas por retângulos) e possui particularidades a depender do fluxo.

O primeiro processo é responsável por ler arquivos, tanto de áudio (.wav, .mp3, .m4a, etc) quanto MIDI (.mid), e gerar uma representação crua da melodia expressa em cada um deles. Esse processo é utilizado em ambos fluxos: durante a leitura das instâncias musicais que representam o *dataset* do sistema (treinamento) e durante a leitura de interpretações melódicas geradas pelo usuário (recuperação). Independentemente do fluxo, no caso da leitura de um arquivo MIDI, o mapeamento para notas é trivial (dado que sabe-se em qual faixa se encontra a melodia monofônica principal), enquanto no caso de leitura de arquivo de áudio, é preciso tratar o mapeamento de informação melódica. Primeiramente, assumindo que o áudio representa uma instância musical com múltiplas vozes, o componente de isolamento de instrumento deve separar a voz principal. Logo após, é necessário que os componentes de mapeamento de

frequências e de volume leiam as informações cruas de frequência e tempo. Como discutido, o resultado pode ser salvo em uma tabela (.csv). A figura 19 ilustra o fluxograma desse processo.

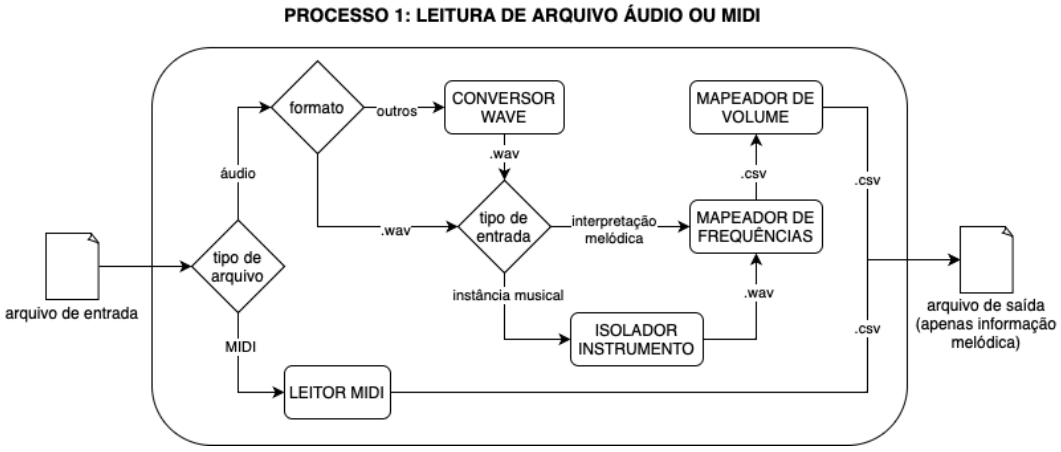

Figura 19 - Leitura de arquivo áudio ou MIDI (processo 1).
Fonte: elaborada pelo autor.

O segundo processo é responsável por receber a representação crua de uma ou mais melodias (tarefa delegada ao primeiro processo) e extrair seus padrões melódicos através da função de distância. Para tal, é preciso que a representação crua de cada melodia seja convertida em uma codificação intermediária, que pode expressar um ou mais fragmentos melódicos contidos na melodia original. O processo também é requisitado em ambos os fluxos. No fluxo de treinamento, o valor de distância retornado é armazenado numa estrutura de mapeamento de distâncias. No fluxo de recuperação, esse valor é simplesmente passado para a frente, sendo tratado pelo terceiro processo. A figura 20 ilustra o fluxograma desse processo.

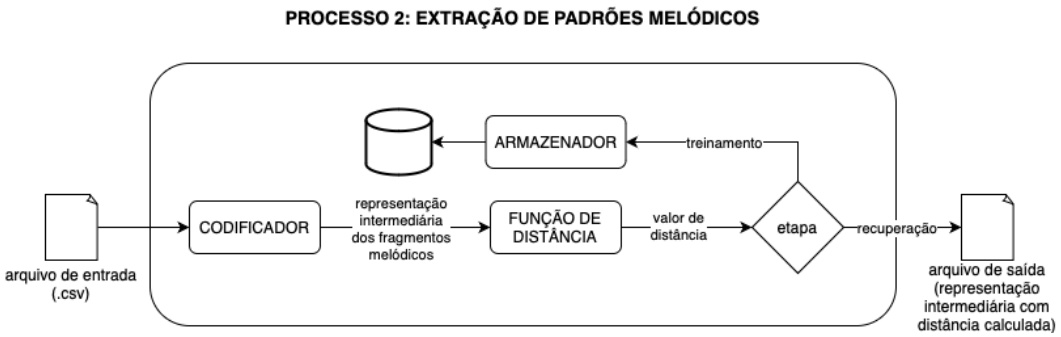

Figura 20 - Extração de padrões melódicos (processo 2).
Fonte: elaborada pelo autor.

O terceiro e último processo só é executado no fluxo de recuperação, dado que o fluxo de treinamento foi executado com sucesso. Ele é responsável por receber a leitura de entrada de interpretação melódica do usuário com valor de distância já calculado (tarefas delegadas ao primeiro e segundo processos), acessar a estrutura de mapeamento de distâncias, efetuar a busca e apresentar seu resultado, fornecendo transparência à comparação realizada. A figura 21 ilustra o fluxograma desse processo.

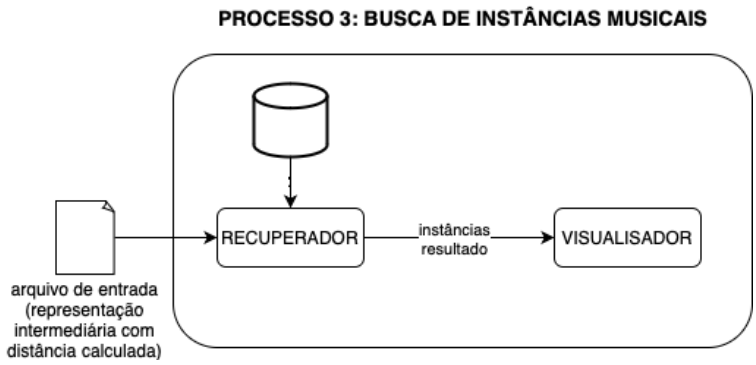

Figura 21 - Busca de instâncias musicais (processo 3).
Fonte: elaborada pelo autor.

A figura 22 ilustra os fluxogramas principais do sistema protótipo, correspondentes aos dois fluxos maiores. Para o fluxo de treinamento, o processo de leitura lê arquivos do *dataset* e o processo de extração compara os fragmentos melódicos, armazenando-os de forma a mapear as diferentes distâncias. Para o fluxo de treinamento, o processo de leitura lê interpretações melódicas pelo usuário, o processo de extração gera a distância para seus fragmentos melódicos e o processo de recuperação compara o valor de distância com o conhecimento disponível na estrutura de armazenamento, gerando uma visualidade final que transparece as semelhanças e comparações melódicas.

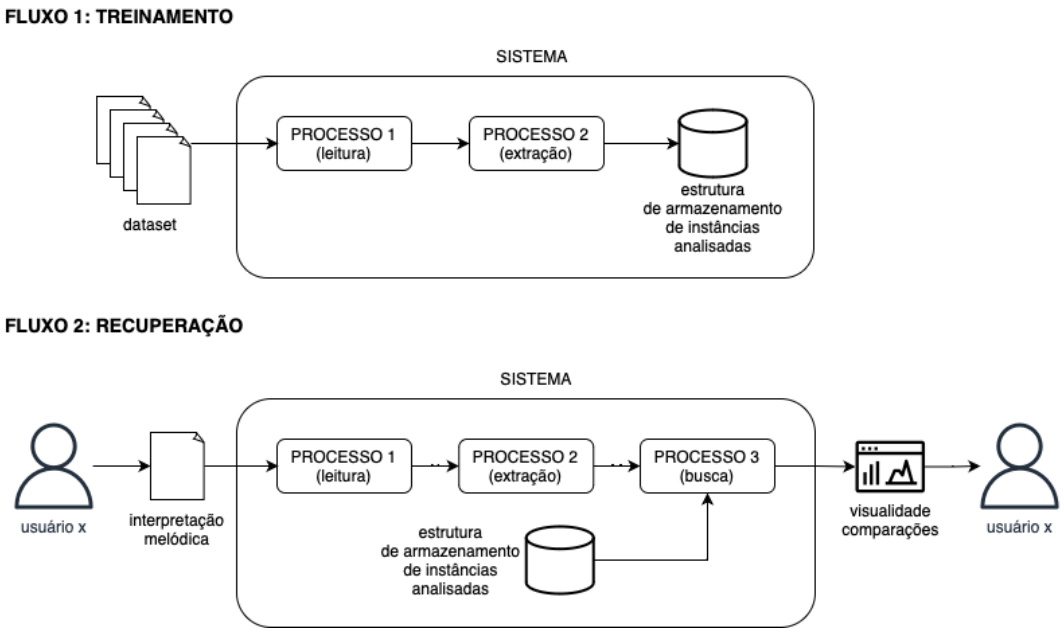

Figura 22 - Treinamento (fluxo 1) e recuperação (fluxo 2).
Fonte: elaborada pelo autor.

5.2. Arquitetura de Código

O código em Python foi estruturado através da arquitetura Model-View-Controller (MVC). De forma geral, as classes responsáveis por apresentar graficamente a comparação melódica compõem o módulo View, as classes que representam os dados das codificações intermediária dos padrões melódicos compõem o módulo Model, e as classes que responsáveis pela leitura e codificação da entrada do usuário compõem o módulo Controller.

5.3. Diagramas de Classe

Nesta seção, os diagramas das classes principais do protótipo serão apresentados. A figura 23 ilustra a classe *Parser*, que possui um método *readFile* para ler um arquivo musical, e *saveTable* para salvar a leitura em uma tabela. Para a leitura, é passado o nome do arquivo (e diretório). O método *saveTable* escreve os dados de leitura em um arquivo de tabela (.csv). A classe, entretanto, é apenas uma abstração, e seus métodos são implementados pelas subclasses *AudioParser* e *MIDIParser*. A subclasse *AudioParser* lê arquivo de áudio e possui algumas propriedades, como *pitchThreshold* que regula a variação de frequência, *volumeThrehold* que regula a variação de volume e *isMonophonic*

que dita se é ou não necessário realizar o isolamento de voz. A subclasse *MIDIParser* recebe o nome (ou índice) da faixa a ser lida, assumindo que ela contém a melodia principal.

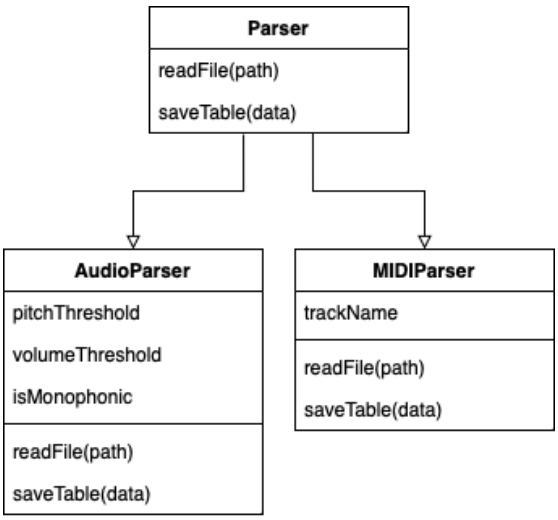

Figura 23 - Classe Parser.
Fonte: elaborada pelo autor.

A figura 24 ilustra a classe *Encoder*, que através de uma representação intermediária de melodias (a ser especificada), codifica a leitura inicial de arquivos áudio e MIDI.

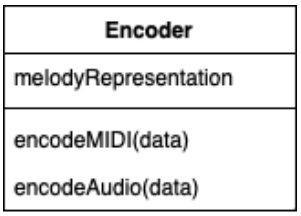

Figura 24 - Classe Encoder.
Fonte: elaborada pelo autor.

A figura 25 ilustra a classe *MelodyComparer*, responsável por implementar a função de distância para a *extração de padrões melódicos*, dados dois fragmentos melódicos codificados. É importante que seja possível, no caso da abordagem de programação dinâmica, especificar os valores de peso para cada transformação. Além disso, ela possui um método que permite retornar e imprimir o caminho ótimo traçado pela função de distância.

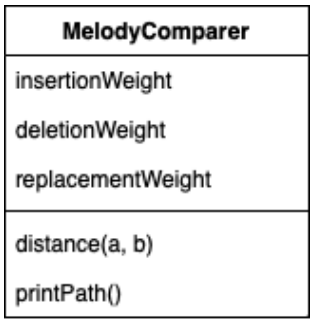

Figura 25 - Classe MelodyComparer.
Fonte: elaborada pelo autor.

A figura 26 ilustra a classe *MelodyStorage*, responsável por acessar a estrutra de dados de armazenamento e salvar (método *saveMelody*) e acessar (método *getMelodies*) os padrões melódicos, dados seus valores de distância.

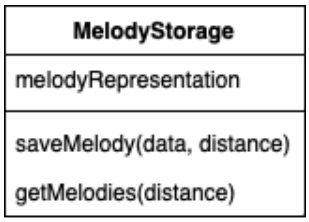

Figura 26 - Classe MelodyStorage.
Fonte: elaborada pelo autor.

6. Solução do Protótipo

Dada a especificação do sistema protótipo, algumas decisões foram feitas. Primeiramente, foi tomada a escolha do *dataset*. Depois, decidiu-se qual abordagem — *comparação de sequências* ou *histograma* — se basear para desenvolver os processos de extração de padrões e busca. Também definiu-se a forma de apresentação das comparações melódicas, buscando garantir boa transparência.

6.1. Consolidação do Dataset

Esta seção irá discutir quais pontos foram considerados na escolha de um *dataset* representativo para o sistema protótipo. Os resultados da *recuperação baseada em conteúdo* implementada são extremamente dependentes da escolha de quais instâncias musicais compõem o conhecimento do sistema.

6.1.1. MIDI x Áudio

A pesquisa por um *dataset* de instâncias musicais representativo para o sistema depende da escolha entre a leitura de arquivos de áudio e a leitura de arquivos MIDI. O objetivo inicial da *recuperação baseada em conteúdo* é justamente poder recuperar os arquivos de áudio de instâncias musicais e seus metadados (nome, artista, data de lançamento, entre outros). Entretanto, como discutido, a leitura de informação melódica em arquivos de áudio é complexa e envolve um alto grau de imprecisão. Se o objetivo é atingir uma grande abrangência, se torna inevitável escolher o uso de áudio. Porém, durante o decorrer do projeto, inclinou-se à decisão de trabalhar com arquivos MIDI, devida sua facilidade de leitura. Assim, buscou-se reduzir o número de variáveis que possam prejudicar a eficácia do sistema. Logo, o *dataset* de músicas do protótipo foi construído a partir de arquivos MIDI, buscando maior acurácia dos resultados. Apesar de não serem instâncias musicais (não representarem performances oficializadas), arquivos MIDI possuem grande potencial para fornecer visualidade de melodias, visto que todos seus valores são intrinsecamente discretos.

6.1.2. Acesso e Notoriedade

Na formação do *dataset* do sistema, é primordial o cuidado em selecionar arquivos que permitem o acesso e análise de seu conteúdo. Além disso, é interessante selecionar um conjunto de obras musicais que possui grande

notoriedade e são de autoria de artistas consolidados e aclamados pela comunidade musical, sendo de conhecimento geral do público-alvo.

6.1.3. Discografia The Beatles em MIDI

Considerando os pontos anteriores, a decisão final foi trabalhar com a discografia MIDI da banda *The Beatles*, disponível gratuitamente pelo site midiworld [21]. O site contém mais de duzentas musicais da banda, disponibilizadas em arquivos MIDI. A banda é uma grande referência para músicos de experiências diversas. Para cada música, seu arquivo MIDI foi pré-analisado manualmente de forma a descobrir quais faixar contém a melodia principal.

6.2. Leitura e Codificação

Como o *dataset* escolhido contém apenas arquivos MIDI, a leitura realizada durante o fluxo de treinamento seguiu o estudo do componente de leitura MIDI. Contudo, para o fluxo de recuperação, decidiu-se aceitar tanto arquivos áudio quanto MIDI. Para tal, uma codificação intermediária compartilhada entre ambos tipos de entrada precisou ser definida.

A codificação escolhida foi tratar uma sequência melódica como uma lista ordenada de intervalos (dada a normalização de tom). Assim, a ordenação entre as notas é importante, porém a informação de duração é descartada. A codificação pareceu servir como simplificação intermediária entre a implementação de Mongeau e Sankoff e a de Liu, Wu, Chen.

6.3. Extração de Padrões Melódicos

O algoritmo de *extração de padrões melódicos* desenvolvido seguiu, em linhas gerais, a abordagem de teoria de sequências e programação dinâmica, sobretudo, por essa possuir uma maior visualidade e controle sobre as comparações realizadas. Entretanto, algumas alterações foram propostas. Primeiramente, como a codificação apenas expressa os intervalos entre cada nota, o peso de cada transformação não considera a duração. Além disso, observou-se que sempre é o caso de a entrada de interpretação melódico pelo usuário ser um fragmento (parte contida) da melodia principal. Portanto, a função de distância *d(a, b)* sugerida não mais é comutativa, e assume que *a* seja maior (em quantidade de notas) que *b.* Assim, o parâmetro *a* recebe uma instância do *dataset* como argumento, e *b* recebe uma entrada do usuário.

Dessa forma, foi descoberto que zerando o peso de remoção de *a* para *b*, pode-se priorizar a relação de inclusão, diminuindo o alto valor de distância que seria retornado caso todas as remoções tivessem um peso não nulo associado. Entretanto, deve-se tomar cuidado para não diminuir a distância em casos indesejados. Considerando apenas remoção e identidade, essa alteração no algoritmo provoca a ocorrência de três casos onde a distância é a menor possível (zero): um caso ideal, um caso intermediário e um caso desfavorável.

O caso ideal é desejado e se dá quando as remoções ocorrem exclusivamente antes e depois da ocorrência da submelodia, ou seja, o "lixo" a ser desconsiderado não aparece entre as transformações identidade (troca por termo igual). O retorno da função é zero, como esperado. A figura 27 ilustra o caso ideal, na qual a primeira linha representa a codificação da instância do *dataset*, a segunda linha representa o peso de cada transformação (zero para remoção e identidade), e a terceira linha representa a codificação da entrada do usuário. As linhas verticais separam a parte relevante da parte “lixo”, representada pelas ocorrências do caracter x.

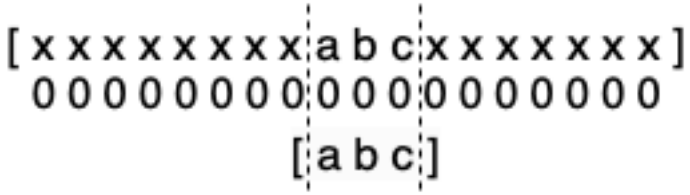

Figura 27 - Caso ideal.
Fonte: elaborada pelo autor.

O caso intermediário se dá quando há algumas poucas remoções entre as transformações identidade. Nesse caso, após o retorno do valor zero pela função, é necessário corrigir seu resultado, acrescentando um valor proporcional ao número de inserções nessa região. A figura 28 ilustra o caso.

Figura 28 - Caso intermediário.
Fonte: elaborada pelo autor.

Por fim, o caso desfavorável é o extremo do caso intermediário, no qual a comparação entre as sequências deveria retornar um alto valor de distância e, no entanto, retorna zero. Deve ser corrigido da mesma forma, após a execução da função de distância. A figura 29 ilustra o caso.

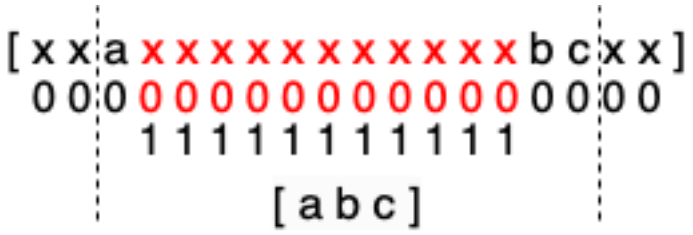

Figura 29 - Caso desfavorável.
Fonte: elaborada pelo autor.

Idealmente, a correção seria feita durante a execução do algoritmo — e não em um momento posterior — mas, no momento desta escrita, ainda não foram encontradas soluções factíveis para isso.

6.4. Recuperação

Para acelerar o processo de busca, a recuperação foi inspirada na abordagem de histograma, implementada pela estrutura de árvore R da biblioteca rtree [15], como discutido anteriormente. A lógica está no fato de que pares de fragmentos melódicos com menor distância têm distribuição de alturas (intervalos) similares. Portanto, dado um fragmento de entrada, faz sentido priorizar o acesso e comparação de instâncias cujos histogramas se encontram mais próximos.

Para atingir a transparência de comparações desejada, decidiu-se apresentá-las de forma gráfica. Como exploração inicial, decidiu-se plotar, após calcular o caminho ótimo de *d(a, b),* os valores de intervalo da sequência *a* antes e depois das transformações realizadas. Dessa forma, o usuário tem uma ideia básica da semelhança encontrada pelo algoritmo. A figura 30 ilustra, para um exemplo de sequência melódica *a*, o gráfico de intervalos antes das transformações aplicadas. O eixo x determina o índice do intervalo e o eixo y, o valor do intervalo. A figura 31 ilustra a mesma sequência depois de aplicadas as transformações, o que resulta na apresentação da sequência *b*. Os gráficos foram gerados com auxílio da biblioteca Matplotlib [13]. Uma vantagem de usar

essa biblioteca é que ela permite animar o gráfico, possibilitando a visualização passo a passo do algoritmo.

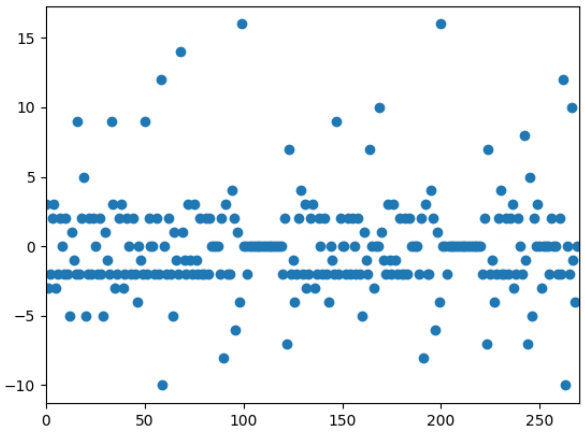

Figura 30 - Intervalos originais.
Fonte: elaborada pelo autor.

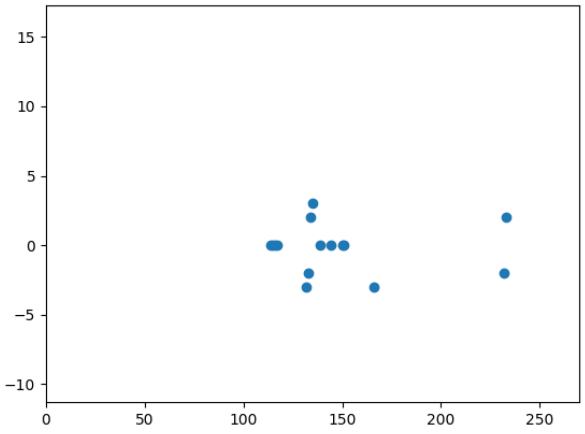

Figura 31 - Intervalos após transformações aplicadas.
Fonte: elaborada pelo autor.

7. Testes e Resultados

Esta seção apresentara um conjunto de testes planejados para medir a acurácia, performance e transparência da etapa de recuperação do protótipo. Para a etapa de treinamento, foram selecionadas 51 músicas da coleção MIDI da banda *The Beatles* [21] para compor o *dataset* do protótipo, apresentadas na seção de anexo.

7.1. Planejamento e execução de testes funcionais

Primeiramente, foram realizados testes a partir de entrada em formato MIDI, geradas por um teclado sintetizador, pois desejou-se verificar a eficácia do algoritmo com a menor quantidade de imprecisões de entrada possível. Cada teste partiu de uma música alvo escolhida, para qual foi tocada uma interpretação melódica de parte de sua melodia. A interpretação melódica buscou ser o mais fiel possível ao fragmento melódico desejado, mas não necessariamente no tom e andamento original. O protótipo, então, ranqueou, a partir da função de distância, as 51 instâncias do sistema pelo seu grau de semelhança com a entrada. Na maioria dos casos, a música esperada ficou em primeiro lugar, isto é, o algoritmo identificou corretamente a semelhança desejada. Em outros casos, a música esperada ficou próxima ao primeiro lugar, isto é, apesar de semelhante, o algoritmo encontrou outra instância com menor distância da entrada. Em poucos casos, a classificação da entrada com a instância esperada foi bem baixa. Nessa situação, três fatores podem ter aumentado indesejavelmente o valor de distância: a interpretação melódica de entrada pode não ter sido fiel ao fragmento melódico; a leitura MIDI da instância no *dataset* pode conter imprecisões; ou o algoritmo não detecta bem a semelhança desejada.

Para alguns testes, foram realizadas mais de uma tentativa, de forma a regravar a interpretação melódica em MIDI, buscando diminuir o valor de distância para a instância esperada. Em especial, reparou-se que o fragmento melódico interpretado deve expressar um fragmento representativo da melodia principal, contendo uma quantidade suficiente de notas. Por exemplo, para T3, a primeira tentativa continha uma interpretação da sequência melódica correspondente a parte cantada “*Words are flowing out like endless rain into a paper cup"* da melodia inicial de *“Across The Universe”*. Entretanto, esse pequeno segmento melódico não foi representativo o suficiente para que o algoritmo identificasse a semelhança. Na segunda tentativa, a interpretação melódica foi estendida para o fragmento correspondente a *"Words are flowing*

out like endless rain into a paper cup. They slither wildly as they slip away across the universe". Nessa tentativa, a quantidade de informação melódica foi suficiente para atingir o resultado esperado.

Nota-se que algumas músicas recebem classificação menor pelo fato de que o algoritmo está descartando informação rítmica. Em algumas obras, como *"Come Together"*, a identidade melódica única é marcada mais pelo ritmo do que pelos intervalos. Esse não é o caso de músicas como *"Eleanor Rigby",* nas quais os intervalos são muito expressivos.

A tabela 1 apresenta os resultados de T1 a T20, indicando a música alvo que desejava-se alcançar a partir da interpretação melódica, o número de tentativas até atingir o melhor resultado, e o melhor resultado do número de classificação da instância esperada, no qual o número 1 significa que a instância esperada foi a mais semelhante à entrada.

	Música Alvo	**Tentativas**	**Resultado**
T1	Come Together	2	2
T2	Let It Be	1	1
T3	Across The Universe	2	1
T4	All My Loving	1	2
T5	And I Love Her	1	1
T6	Because	1	1
T7	Don't Let Me Down	1	11
T8	Eleanor Rigby	1	1
T9	For No One	1	1
T10	Here Comes The Sun	1	9
T11	Hey Jude	1	1
T12	If I Fell	1	1
T13	Long and Winding Road	1	1
T14	Michelle	1	1
T15	Norwegian Wood (This Bird Has Flown)	1	1
T16	Ob-La-Di, Ob-La-Da	2	1
T17	Something	1	2
T18	The Fool on the Hill	1	1

	Música Alvo	Tentativas	Resultado
T19	While My Guitar Gently Weeps	1	2
T20	Yesterday	1	6

Tabela 7 - Resultado dos testes em MIDI.
Fonte: elaborada pelo autor.

Para uma amostragem de vinte testes, o algoritmo identificou, com êxito total, treze músicas (65%). Quatro dos vinte testes (20%) responderam a música esperada como segunda opção mais semelhante. Estipulando que o sistema protótipo pode apresentar uma lista de até cinco músicas similares à entrada (cerca de 10% do *dataset* treinado), a acurácia para os testes foi de 85%, com três testes (15%) falhando em retornar a música esperada.

O algoritmo apresentou razoável performance, com tempo médio de execução (sem considerar o tempo de leitura dos arquivos) de aproximadamente 0.34 segundos, considerando uma entrada na ordem de dezenas de intervalos. Entretanto, mais testes teriam de ser feitos afim de avaliar a escalabilidade do protótipo, tendo em vista que o *dataset* atual é muito pouco abrangente.

7.2. Transparência

No momento desta escrita, admite-se que não foi possível avaliar por completo a transparência do algoritmo. Em um momento futuro, é importante realizar mais testes de exploração gráfica, e possivelmente até sonora, das comparações realizadas pelo algoritmo, provendo ao usuário um maior entendimento sobre as similaridades encontradas. Pode-se pensar na reprodução de trechos sonoros que contenham os padrões melódicos de entrada e saída que se assemelham. Pode-se pensar na apresentação de pares de entrada e saída através da representação gráfica de arquivos MIDI, ou até mesmo da representação de pauta musical. Contudo, deve-se certificar que a forma escolhida de apresentar as comparações melódicas seja compreendida pelo público-alvo, isto é, para pessoas relacionadas com a arte musical de diferentes experiências e domínio musical teórico.

8. Considerações Finais

Considera-se que o projeto desenvolvido possa a vir a constituir alguma contribuição acadêmica para o campo da ciência da computação, dada sua ênfase em compilar e discutir diferentes materiais teóricos sobre o estudo de *extração de padrões melódicos* e *recuperação de música baseada em conteúdo melódico*. A proposta de construção de uma ferramenta de software que busca apresentar similaridades melódicas entre o material do músico e referências no material de outros músicos tem como objetivo final capacitar o conhecimento do artista, sem tirar de sua mão o controle sobre as decisões criativas tomadas durante o processo criativo. Como músico, o autor considera essa proposta mais interessante do que, por exemplo, a de as ferramentas emergentes de estímulo criativo que funcionam a partir de inteligência artificial generativa.

Apesar de primitivo, considera-se que o protótipo apresentado atingiu o objetivo proposto, principalmente dada as dificuldades encontradas. A maior dificuldade no processo de desenvolvimento do protótipo se deu na implementação das técnicas estudadas, já que são escassas as referências de código disponíveis abertamente para o público. Pelo fato de existir limitações no acesso da fundamentação teórica, como por exemplo, o fornecimento exclusivo e limitado de pseudocódigos, foi necessário exercer a capacidade interpretativa.

A partir do atual estágio do projeto, detecta-se diversas oportunidades de desdobramentos que gostaria de empreender adiante. Dentre elas, destaca-se uma melhor exploração da apresentação transparente das comparações melódicas para o usuário, que pode estar interessante em estudar semelhanças particulares. A apresentação pode conter feedbacks visuais, sonoros e até animações gráficas. Para tal, é preciso realizar um levantamento aprofundado sobre como músicos de diversos níveis de proficiência concebem melodias. Possivelmente, é interessante considerar o embarcamento da ferramenta como plugin em uma DAW, visto que esse é o mais comum ambiente de produção musical na contemporaneidade. Por último, para que o projeto sirva de referência aberta de um sistema que implementa o processo de *recuperação baseada em conteúdo* de forma minimamente abrangente, é positiva a exploração da abordagem de rede neural.

Pessoalmente, o desenvolvimento de um projeto voltado para a pesquisa acadêmica, no qual é possível propor soluções para um problema computacional em aberto, me interessou enquanto estudante. O processo de busca por artigos e referências sobre um problema computacional específico de interesse, assim como o desenvolvimento de estudos e métodos foram transformadores na minha experiência como cientista da computação.

9. Referências Bibliográficas

[1] WALLACE, Wanda. **Memory for Music: Effect of Melody on Recall of Text**, 1994.

[2] MONGEAU, Marcel; SANKOFF, David. **Comparison of Musical Sequences**, 1990.

[3] ROLLAND, Pierre-Yves. **Discovering Patterns in Musical Sequences**, 1999.

[4] LIU, Ning-Han; WU, Yi-Hung; CHEN, Arbee L.P. **An Efficient Approach to Extracting Approximate Repeating Patterns in Music Databases**, 2005.

[5] KUMAR, Krishna. **Song stuck in your head? Just hum to search**, Google. Disponível em: https://blog.google/products/search/hum-to-search/. Acesso em: Abril, 2023.

[6] SOUNDHOUND, INC. **SoundHound - Music Discovery**. Disponível em: https://apps.apple.com/us/app/soundhound-music-discovery/id355554941. Acesso em: Abril, 2023.

[7] APPLE. **Shazam: Music Discovery**. Disponível em: https://apps.apple.com/us/app/shazam-music-discovery/id284993459. Acesso em: Abril, 2023.

[8] FRANK, Christian. **The Machine Learning Behind Hum to Search**, Google Research. Disponível em: https://blog.research.google/2020/11/the-machine-learning-behind-hum-to.html?m=1. Acesso em: Setembro, 2023.

[9] MED, Bohumil. **Teoria da Música.** Brasília: Musimed, 1996.

[10] TCHEEUBE, Renaud; ENS, Jeffrey; PLUT, Cale; PASQUIER, Philippe; SAFI, Maryam; GRABIT, Yvan; ROLLAND, Jean-Baptiste. **Evaluating Human-AI Interaction via Usability, User Experience and Acceptance Measures for MMM-C: A Creative AI System for Music Composition**, 2023.

[11] SANKOFF, David; KRUSKAL, Joseph. **Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison**, 1983.

[12] APPLE. **Logic Pro**. Disponível em: https://www.apple.com/logic-pro/. Acesso em: Outubro, 2023.

[13] HUNTER, J. D. **Matplotlib: A 2D Graphics Environment**, 2007.

[14] BECKMANN, Norbert; KNEGEL, Hans-Pete; SCHNEIDER, Ralf; SEEGER, Bernhard. **The R*-tree: An Efficient and Robust Access Method for Points and Rectangles+**, 1983.

[15] GILLES, Sean; BUTLER, Howard. **Rtree: Spatial indexing for Python.** Disponível em: https://rtree.readthedocs.io/en/latest/. Acesso em: Setembro, 2023.

[16] KIM, Jong Wook; SALAMON, Justin; LI, Peter; BELLO, Juan Pablo. **CREPE: A Convolutional Representation for Pitch Estimation**, 2018.

[17] ROBERT, James. **Pydub.** Disponível em: https://github.com/jiaaro/pydub. Acesso em: Agosto, 2023.

[18] ROUARD, Simon; MASSA, Francisco; DÉFOSSEZ, Alexandre. **HYBRID TRANSFORMERS FOR MUSIC SOURCE SEPARATION**, 2018.

[19] BACK, David. **Standard MIDI-File Format Spec. 1.1, updated**, The International MIDI Association. Disponível em: http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html. Acesso em: Julho, 2023.

[20] BJØRNDALEN, Martin. **Mido - MIDI Objects for Python**. Disponível em: https://mido.readthedocs.io/en/stable/#. Acesso em: Julho, 2023.

[21] LIPSCOMB, Eric. **The Beatles - midiworld**, Electronic Forum for Campus Computing Newsletter Editors. Disponível em: https://www.midiworld.com/files/995/. Acesso em: Outubro, 2023.

10. Anexos

10.1. Lista de Músicas do Protótipo

Esse anexo contém o subconjunto final de 51 músicas selecionadas da coleção MIDI *The Beatles* para compor o *dataset* do protótipo.

- Across The Universe
- All My Loving
- And I Love Her
- Because
- Blackbird
- Come Together
- Don't Let Me Down
- Drive My Car
- Eleanor Rigby
- For No One
- Get Back
- Golden Slumbers
- HELP!
- Here Comes The Sun
- Here, There and Everywhere
- Hey Bulldog
- Hey Jude
- I Feel Fine
- I Me Mine
- I Want to Hold Your Hand
- I Want You (She's So Heavy)
- I Will
- I've Just Seen a Face
- If I Fell
- In My Life
- Julia
- Let It Be
- Long and Winding Road
- Long, Long, Long
- Love Me Do
- Michelle

- Norwegian Wood (This Bird Has Flown)
- Nowhere Man
- Ob-La-Di, Ob-La-Da
- Octopus's Garden
- Oh Darling
- Paperback Writer
- Penny Lane
- Sgt. Pepper's Lonely Hearts Club Band
- She Loves You
- She's Leaving Home
- Something
- Taxman
- The Fool on the Hill
- Ticket to Ride
- Two of Us
- When I'm Sixty-Four
- While My Guitar Gently Weeps
- With A Little Help From My Friends
- Yellow Submarine
- Yesterday