Relatório Técnico

Núcleo de Computação Eletrônica

MIDIZ: Indexação e Recuperação Baseada em Conteúdo de Arquivos MIDI

Marcelo Trannin Machado[1]
Maria Cláudia Reis Cavalcanti[1,2]
Alessandro de Almeida Castro Cerqueira[1]
Nelson Sampaio Araujo Júnior[1]
Geraldo Xexéo[1,3]

(1) COPPE Sistemas; (2) NCE; (3) IM - DCC (UFRJ).

NCE - 13/99

Universidade Federal do Rio de Janeiro

MIDIZ: Indexação e Recuperação Baseada em Conteúdo de Arquivos MIDI

Marcelo Trannin Machado [1]
Maria Cláudia Reis Cavalcanti [1,2]
Alessandro de Almeida Castro Cerqueira [1]
Nelson Sampaio Araujo Júnior [1]
Geraldo Xexéo [1,3]

[1] COPPE Sistemas
[2] NCE
[3] IM - DCC
Universidade Federal do Rio de Janeiro

e-mail: {marcelot, yoko, aacc, nelsonjr, xexeo}@cos.ufrj.br

RESUMO

Este trabalho apresenta um sistema de busca em arquivos musicais via Internet, baseado na descrição de trechos musicais. A busca é feita de maneira a ser independente da tonalidade em que o trecho é descrito pelo usuário. Simultaneamente, o esquema também é capaz de admitir uma certa quantidade de erros na descrição do trecho musical, de acordo com um parâmetro controlável a partir da interface.

palavras-chave: multimídia, busca e recuperação por conteúdo

ABSTRACT

This paper presents a search engine for musical files in the Internet, based on the description of a musical fragment. The search engine allows that description to be pitch independent. Furthermore, the system admits some errors on the musical fragment description, according to a parameter available in the interface.

keywords: multimedia, content-based information retrieval

1. Introdução

A busca de uma música específica baseada em um pequeno trecho memorizado é um dos desejos mais comuns entre as pessoas. Para responder a essa pergunta, geralmente apenas duas alternativas estão disponíveis: uma busca demorada em alguma discoteca, ou recorrer à memória privilegiada de alguém.

Motivados pelo grande sucesso dos *sites* de busca na Internet quando utilizados para a busca de arquivos texto e pelo crescente desenvolvimento de técnicas de busca para informações não textuais, imaginamos a possibilidade de realizar um sistema capaz de indexar e recuperar arquivos musicais por meio da descrição de pequenos trechos. Apresentamos nesse artigo o resultado desse trabalho: um sistema (MIDIZ) capaz de armazenar, indexar, pesquisar e recuperar arquivos musicais, utilizando para a recuperação pequenos trechos musicais, desconsiderando tonalidade e admitindo uma série de erros na descrição do trecho.

As formas de armazenamento de música em meio digital vêm caminhando para uma

padronização. Armazenar dados de áudio em forma de onda consome muito espaço. O padrão MIDI ("Musical Instrument Digital Interface") [Heckroth95] de armazenamento gera arquivos extremamente pequenos quando comparados a arquivos gerados por amostragem de ondas sonoras. Isto se dá em virtude do arquivo MIDI não armazenar o sinal sonoro propriamente dito, mas sim os comandos necessários para gerá-lo. Os sons são gerados posteriormente pelos sintetizadores. Este formato facilita a análise da música, tornando mais simples a extração de padrões e a identificação de características referentes à seqüência de notas. O padrão MIDI é hoje um padrão *de facto* da indústria de instrumentos musicais eletrônicos e de programas musicais (editores e sequenciadores). Essas características tornam arquivos MIDI a forma mais aconselhável para o registro musical em computador e, consequentemente, o alvo mais adequado para sistemas de indexação musical.

Alguns dos trabalhos que buscam a indexação de arquivos MIDI o fazem através da análise de atributos ou textos descritivos presentes no cabeçalho do arquivo [Figueiredo97]. Esse tipo de pesquisa é interessante quando conhecemos atributos da música, como nome ou autor, mas não é capaz de auxiliar a busca baseada apenas na memória auditiva. Nesse caso, é necessária a indexação e recuperação por conteúdo, isto é, pelo registro musical disponível no arquivo. Em [Ghias95] os autores propõem uma forma de indexação da música a partir de uma abstração do relevo musical conhecido como UDS. Para obter este código, cada nota é substituída por um movimento relativo à nota anterior: para cima (U), para baixo (D), ou permanece (S). A principal falha deste tipo de abordagem é que a ausência de flexibilidade dos fragmentos musicais oferece apenas a similaridade escondida pela abstração utilizada, permitindo apenas descrições exatas.

Nosso trabalho é fortemente baseado na proposta de [Beeferman97], que apresenta uma preocupação maior com a flexibilidade. Nesse trabalho, trechos musicais são definidos por meio de uma janela deslizante sobre as notas musicais, transformados então em vetores, sendo estes armazenados em uma estrutura de dados. O autor porém, aplica esse processo apenas à informação correspondente à altura das notas musicais e não à sua duração.

A próxima seção descreve o processo usado para indexação dos arquivos MIDI. A seção 3 apresenta as estruturas de indexação consideradas. As seções 4 e 5 descrevem o sistema MIDIZ, sua arquitetura e implementação. Uma breve discussão sobre a performance do sistema MIDIZ é apresentada na seção 6. Finalmente, a última seção conclui e prevê as melhorias possíveis no sistema.

2. Identificando Pontos no Arquivo MIDI

Em aplicações de indexação e recuperação musical, deve-se permitir consultas não exatas. De fato, nem sempre a tonalidade em que se faz uma consulta é a mesmo em que esta foi gravada e armazenada em um repositório digital. Uma solução para permitir transposições de tonalidade de forma automática foi usada em [Dirst94]. Ao invés de identificar as músicas como uma seqüência de notas, utiliza-se a diferença entre as notas. Entretanto, para se atingir uma flexibilidade ainda maior, é preciso permitir que o usuário cometa pequenos erros ao efetuar uma consulta, como por exemplo, subir ou descer uma nota. Além disso, um outro requisito é a possibilidade de recuperar uma música a partir de qualquer um dos seus trechos.

O esquema de indexação proposto em [Beeferman97], usado com adaptações no sistema MIDIZ, procura atender a estes requisitos utilizando uma transformada *wavelet* e uma janela deslizante sobre as notas. A janela define uma seqüência de notas de tamanho 2^K e desliza por toda música, deslocando seu início de uma em uma nota. Cada trecho musical de 2^K notas enquadrado pela janela deslizante é convertido em um conjunto de 2^K-1 valores, segundo uma

transformada bem semelhante à transformada *wavelet* de Haar usada em [Faloutsos96].

A utilização da janela deslizante permite que todos os trechos da música de tamanho 2^K sejam indexados, cumprindo o requisito de indexar todos os trechos possíveis da música. Assim, se fixarmos k em *3*, teremos uma janela de tamanho *8*. Deslizando a janela por uma música com um total de T notas, teremos $T - 2^K + 1$ trechos identificados na música.

Para cada trecho identificado, aplica-se a transformada, segundo [Beeferman97], da seguinte forma:

1. Atribui-se o valor igual a 1 à primeira nota. O valor das demais notas do trecho é determinado em função da quantidade de notas na escala natural que as separam da nota inicial. Assim, se a nota é uma nota natural acima da primeira, recebe valor igual a 2; se está duas notas naturais acima, recebe valor igual a 3; e assim por diante.
2. Soma-se os valores das notas do trecho, duas a duas, gerando um segundo e um terceiro nível de valores. Ou seja, a partir do trecho (n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8), geramos as sequências de valores (n_9, n_{10}, n_{11}, n_{12}) e (n_{13}, n_{14}), onde:

   $n_9 = n_1+n_2$,
   $n_{10} = n_3+n_4$,
   $n_{11} = n_5+n_6$,
   $n_{12} = n_7+n_8$,
   $n_{13} = n_{10}+n_{11}$,
   $n_{14} = n_{12}+n_{13}$
3. Em seguida, na ordem inversa, faz-se a diferença entre os valores resultantes, obtendo-se as 2^K-1 coordenadas. Tomando K = 3, teremos as coordenadas (c_1, c_2, c_3, c_4, c_5, c_6, c_7), onde:

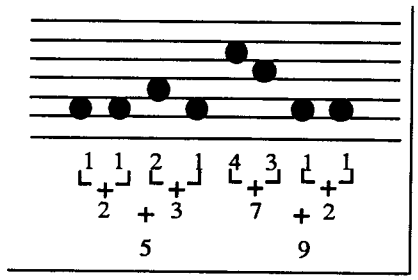
   $c_1 = n_{14} - n_{13}$,
   $c_2 = n_{12} - n_{11}$,
   $c_3 = n_{10} - n_9$,
   $c_4 = n_8 - n_7$,
   $c_5 = n_6 - n_5$,
   $c_6 = n_4 - n_3$,
   $c_7 = n_2 - n_1$

Tomemos como exemplo o trecho da música "Parabéns pra você", *(1,1,2,1,4,3,1,1)*. Ao aplicarmos a primeira fase da transformada teremos os números *(1,1,2,1,4,3,1,1)*, *(2,3,7,2)*, *(5,9)* (Fig. 2.1(a)). Após aplicarmos a segunda fase da transformada, obteremos os valores *(-4,5,-1,0,1,1,0)*, que correspondem às coordenadas em um espaço 7-dimensional (Fig. 2.1(b)). Assim, dada uma base de arquivos MIDI, cada arquivo corresponderá a vários pontos em um espaço com 7 dimensões.

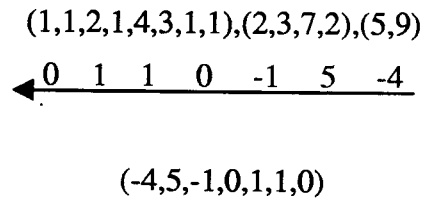
Entretanto, a forma de cálculo do vetor no sistema MIDIZ apresenta uma pequena diferença. No exemplo apresentado pela Figura 2.1, a determinação dos valores das notas foi feita somente de acordo com as notas da tonalidade. No MIDIZ, esta determinação leva em conta todas as notas da escala cromática, inclusive aqueles que não fazem parte da tonalidade. Com isso, os pontos que são gerados, tanto na indexação quanto na consulta, apresentam uma melhor precisão musical.

Para exemplificar, os valores dados às notas na Figura 2.1 foram *(1,1,2,1,4,3,1,1)*, gerando o ponto *(-4,5,-1,0,1,1,0)*; já segundo esta abordagem, os valores dados às notas

seriam *(1,1,3,1,6,5,1,1)*, gerando o ponto *(-7,9,-2,0,1,2,0)*.



(a) Somando as notas duas a duas (essas linhas não representam uma pauta musical) [Beeferman97]



(b) Obtendo a diferença na ordem inversa

Figura 2.1: Aplicando a transformada sobre um trecho da música "Parabéns pra você":

Ao submeter uma consulta, o usuário apresenta uma seqüência de notas, cujo tamanho corresponde ao tamanho da janela deslizante. Esta seqüência é transformada em um ponto, da mesma forma como foi descrito anteriormente, que deve então ser comparado com todos os pontos disponíveis na base de dados. É neste momento que o sistema deve oferecer flexibilidade e retornar ao usuário não somente as músicas que contenham aquele ponto, mas também todas as músicas que contenham pontos próximos. Então, para processar a consulta por similaridade, é preciso calcular a distância Euclidiana do ponto dado a cada um dos pontos existentes na base musical, e selecionar aqueles pontos que guardam uma distância menor que a considerada como erro máximo admissível. No sistema MIDIZ, esta distância máxima fica a cargo do usuário, isto é, o usuário define o grau de "erro" que quer considerar ao efetuar sua consulta.

A Tabela 2.1 apresenta um exemplo da obtenção da diferença Euclidiana entre dois pontos. A partir de um trecho musical correto, geramos um outro trecho incorreto, descendo um semitom na quarta e quinta notas. Obtemos os pontos respectivos e então calculamos a distância Euclidiana entre os dois pontos.

Trecho musical correto	Trecho musical com erro	Ponto Correto	Ponto com erro	Diferença Euclidiana
1 1 2 1 4 3 1 1	1 1 2 1 **3 2** 1 1	-4 5 -1 0 1 1 0	-2 3 -1 0 1 1 0	2,83

Tabela 2.1: Exemplo de diferença Euclidiana entre o ponto correto e um ponto com erro.

Para certos erros, o sistema gera pontos com uma Diferença Euclidiana maior, em relação ao ponto que se quer atingir. Nestes casos, para obter a música solicitada, o usuário será obrigado a aumentar o grau de "erro" da consulta, de forma que pontos mais distantes sejam recuperados.

Com o objetivo de complementar o trabalho de [Beeferman97], este trabalho propõe também um esquema de indexação de durações de notas. Nos arquivos MIDI, as durações são identificadas de acordo com a unidade de tempo definida no cabeçalho do arquivo. Considerando por exemplo, que a unidade de tempo apresenta uma resolução igual a *240*, e que a música apresenta o compasso *4/4*, a Tabela 2.2 mostra como os valores encontrados no arquivo MIDI são transformados para os valores que serão usados para identificar a sequência de durações.Para obter um ponto relativo a uma seqüência de durações extraída de um arquivo

MIDI, aplica-se a transformada *wavelet* de Haar sobre esta seqüência, da mesma forma como foi feito para a sequência de notas. Assim, se tomarmos novamente *k* igual a *3*, poderíamos ter por exemplo a seguinte seqüência de durações: *(8,4,4,4,8,8,8,16)*. Ao aplicarmos a transformada, obtemos o ponto *(-20,-8,4,-8,0,0,4)*.

Nota	Representação no arquivo MIDI	Valor correspondente
Semibreve	960	64
Mínima	480	32
Semínima	240	16
Colcheia	120	8
Semicolcheia	60	4
Fusa	30	2
Semifusa	15	1

Tabela 2.2: Tabela de tempos de nota, considerando que a unidade de tempo é 240

Concluindo, a indexação de um arquivo MIDI gera vários pontos. Os pontos originados a partir da análise das seqüências de notas, e das seqüências de durações destas, são armazenados em uma estrutura de dados na memória. A seção seguinte discute algumas das estruturas de indexação que podem ser usadas neste tipo de problema.

3. Estruturas de Indexação

Um requisito fundamental para um sistema de busca e recuperação de informação musical é que o resultado da consulta seja formado pela lista de músicas que contêm pontos próximos ao ponto consultado e não somente aquelas que contêm pontos iguais ao mesmo. Isto é, todos os pontos que se encontram dentro de uma região ao redor de um dado ponto devem ser considerados. Porém, é preciso considerar que em uma base contendo milhões de pontos, o tempo gasto pelo processamento do cálculo das distâncias entre os pontos da base e o ponto consultado inviabilizaria a consulta.

Assim sendo, é preciso utilizar estruturas de indexação que permitam uma organização dos pontos em grupos e subgrupos por proximidade, de tal forma que a consulta possa ser automaticamente direcionada à comparação de um subconjunto destes pontos e que, ao navegar por esta estrutura, seja possível identificar os grupos que representam os pontos mais próximos ao ponto consultado.

Os métodos de organização de dados que parecem melhor se adequar ao nosso caso são conhecidos como métodos de agrupamento multi-dimensional dinâmico [Derakhshan89] e podem ser caracterizados da seguinte forma:

(a) O espaço de dados é k-dimensional, por isso o termo multidimensional.

(b) Os dados correspondem a pontos em uma região particular, que por sua vez está associada a uma mesma página de dados, por isso o termo agrupamento. As regiões ou partições são também chamadas de hiper-retângulos, e possuem uma capacidade máxima.

(c) O conjunto de todas as páginas de dados cresce como resultado da subdivisão das páginas de dados que ultrapassam sua capacidade, por isso o termo dinâmico.

Os chamados *bucket methods* [Samet90] adaptam-se a esta descrição. Foram

desenvolvidos para colecionar pontos em conjuntos, chamados *buckets*, correspondendo a unidades de armazenamento do disco (i.e., páginas). A medida que os *buckets* chegam ao limite de sua capacidade de armazenamento, a sua estrutura é automaticamente reorganizada, criando novos *buckets* e redistribuindo os pontos entre estes. Para organizar o acesso a estes *buckets*, uma estrutura auxiliar chamada diretório é utilizada, de tal forma que seja possível, dado um registro, encontrar o *bucket* a que este pertence. *Grid files* [Nievergelt84] e *R-trees* [Guttman84] são exemplos de estruturas que utilizam esta abordagem.

As *R-trees* foram projetadas com o objetivo de representar pontos agrupados em regiões, que podem sobrepor-se (não disjuntas). Como uma estrutura k-dimensional, uma *R-tree* é formada por várias regiões k-dimensionais chamadas hiper-retângulos. Em cada nível da *R-tree* há um conjunto de hiper-retângulos, sobrepostos ou não, que podem conter outros hiper-retângulos pertencentes ao próximo nível da árvore. A Figura 3.1 (a) ilustra uma *R-tree* bidimensional, onde os retângulos maiores pertencem ao primeiro nível, e os retângulos menores pertencem ao segundo nível da árvore.

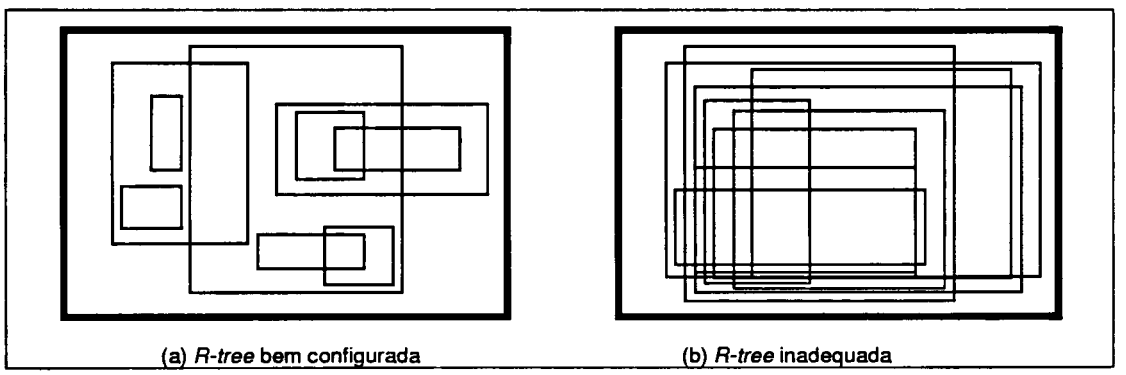


Figura 3.1: Exemplos de *R-trees*

Inicialmente, consideramos utilizar *R-trees*. Sabemos que cada música, após ser analisada, gera um conjunto de pontos. Se tomarmos como definição do hiper-retângulo, os pontos limítrofes da música (pontos com as menores e maiores coordenadas), poderíamos considerar que cada música corresponde a um hiper-retângulo. Assim, ao consultar um ponto, navegaríamos pela estrutura *R-tree* e teríamos como resultado todos os hiper-retângulos que contêm aquele ponto.

Por outro lado, nada garante que os hiper-retângulos gerados pelos pontos extraídos da análise das músicas serão representativos. Isto é, os hiper-retângulos podem estar definidos de tal forma que não identificam de forma diferenciada cada música no hiperespaço (Fig. 3.1(b)). Assim, ao consultarmos um ponto, teríamos como resultado uma quantidade pouco seletiva de músicas, isto é, alta revocação e baixa precisão[1].

Em sua proposta de *GRID files*, [Nievergelt84] deixa em aberto várias questões, como por exemplo a organização do diretório de acesso aos *buckets*, de tal forma que o espaço requerido para seu armazenamento varie linearmente com o número de registros armazenados. Neste sentido, entre os trabalhos desenvolvidos para resolver esta questão, o que melhor se destaca são os *BANG files* (*Balanced And Nested GRID*) [Freeston87], onde a

---

[1] Precisão é a porcentagem de respostas certas nas respostas dadas pelo sistema. Revocação é a porcentagem de respostas certas dadas pelo sistema dentro do número de respostas certas disponíveis na base. [Salton95]

estratégia de particionamento é tal que evita a criação de células vazias, mantendo no diretório referências apenas a *buckets* não vazios. Desta forma, os *BANG files* são mais adequados para casos em que a distribuição não é uniforme. A Figura 3.2 ilustra o comportamento dos *GRID* e *BANG files* quando esta situação ocorre. Note que o particionamento segue o mesmo algoritmo, porém os *BANG files* mantêm um número mínimo de células necessárias, adaptando-se à concentração de pontos.

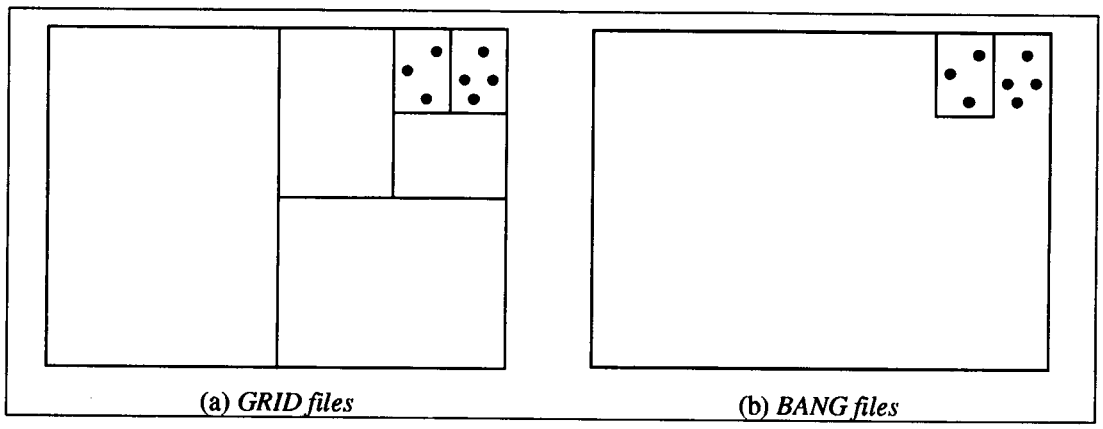


Figura 3.2: Particionamento em casos de distribuição não uniforme de pontos.

Como no problema tratado por este trabalho a distribuição dos pontos é desconhecida, optamos por utilizar *BANG files*. Estes também são conhecidos como *BD-trees* [Samet90], e os algoritmos correspondentes estão descritos em [Dandamudi86].

A localização de um ponto em uma *BD-tree* é dada por uma sequência binária, que descreve uma região do espaço multidimensional. Esta região é gerada a partir da simulação de particionamento deste espaço. Dado um ponto, este é analisado da esquerda para a direita, de forma cíclica em relação às suas coordenadas. Para cada coordenada do ponto, o espaço disponível é particionado no ponto médio. Se o ponto for inferior ao ponto médio, a sequência binária é acrescida de um "*0*" à direita. Caso contrário, a sequência é acrescida de um "*1*". Isto é, a sequência simplesmente diz qual lado de cada linha divisória sucessiva está o ponto. Esta sequência recebe o nome de DZE, ou *Discriminator Zone Expression*, e representa todo o conjunto de pontos pertencente àquela região.

Cada nó da *BD-tree* é rotulado por um DZE. A raiz não tem rótulo. Cada nó da árvore representa um subconjunto do espaço. Este espaço é especificado recursivamente como se segue. A raiz representa o espaço completo. Se um nó x representa um subconjunto do espaço s, e ele é rotulado com um DZE d, então o filho esquerdo de x representa a interseção de s e o espaço representado por d, e o filho direito de x representa a interseção de s com o complemento do espaço representado por d (Fig. 3.3).

Podemos identificar dois tipos de nós em uma *BD-tree*: nó interno e nó externo. Os nós internos são os nós não folha, e os nós externos são as folhas da árvore, que correspondem aos *buckets*, onde se armazenam os pontos. Como foi citado anteriormente, a *BD-tree* tem um comportamento dinâmico auto-ajustável. Assim, a *BD-tree* pode ser construída por uma série de inserções. Todo nó externo armazena uma lista de pontos, e todo nó interno armazena um DZE.

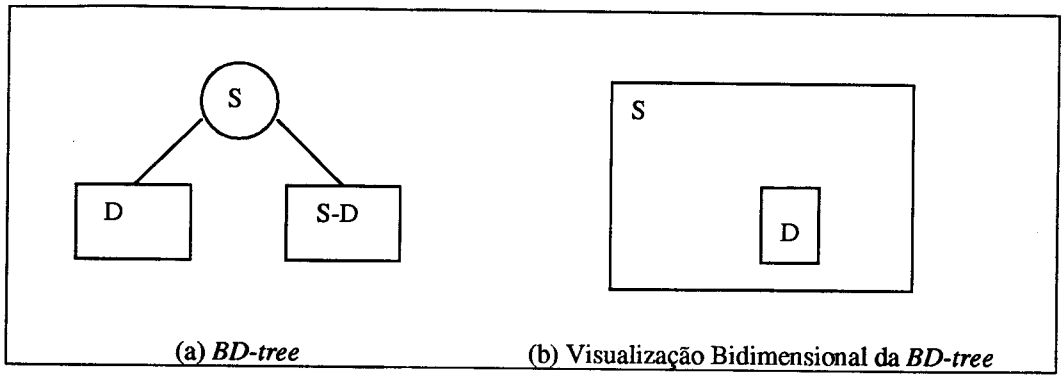


Figura 3.3: Esboço de uma *BD-tree*.

4. O Sistema MIDIZ

O sistema está dividido em dois módulos: Indexação e Consulta. O módulo de Indexação faz uma análise léxica de um conjunto de arquivos no formato MIDI e gera a estrutura em árvore (*BD-tree*) e a lista de ocorrências, para posterior consulta. O módulo de Consulta permite que o usuário faça consultas por notas e/ou pela duração destas, utilizando os arquivos gerados pelo módulo de Indexação, para identificar os arquivos que satisfazem à consulta feita pelo usuário.

Mais adiante, as seções descrevem em mais detalhes os módulos do sistema. Porém, para compreendê-los melhor, primeiramente apresentamos o esquema de dados do sistema MIDIZ (Fig. 4.1), que representa a *BD-tree*, através da notação OMT [Rumbaugh91]. O esquema é formado basicamente pelas seguintes classes: `ÁrvoreBD`, `Nó`, `Ponto`, `PosiçãoPontoMúsica` e `Música`. A classe `ÁrvoreBD` é composta por um conjunto de `Nós`. Cada `Nó` pode ser do tipo interno ou externo. Um `Nó` interno está ligado a dois outros, um à direita, chamado OUT, e um à esquerda, chamado IN. Um `Nó` do tipo externo possui um conjunto de `Pontos`. Cada `Ponto` pode apresentar mais de uma ocorrência (`PosiçãoPontoMúsica`) na mesma `Música`, ou em `Músicas` diferentes. Cada ocorrência informa em que música e em que posição o trecho musical ocorreu.

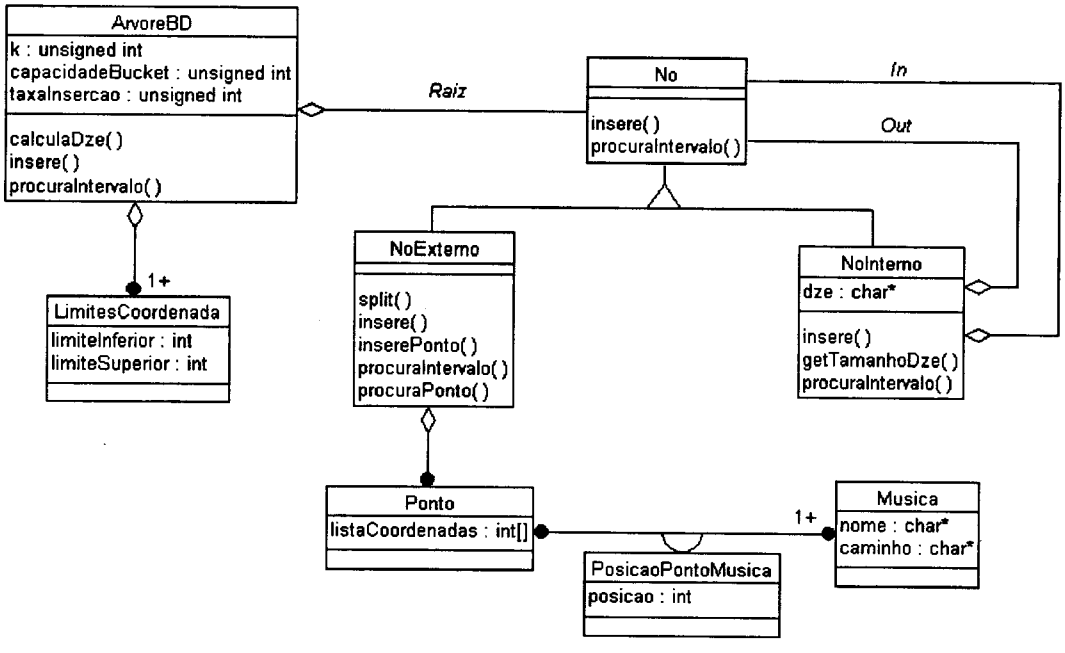


Figura 4.1: Modelo de dados MIDIZ

4.1. Indexação

O módulo de Indexação subdivide-se em dois módulos: Análise Léxica e Geração da Árvore, como mostra a Figura 4.2.

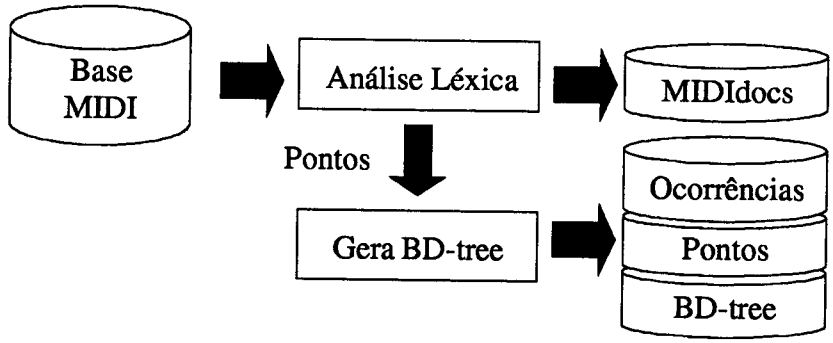


Figura 4.2: Módulo de Indexação

O módulo de Análise Léxica é responsável pela leitura de um ou mais arquivos no formato MIDI, selecionados pelo usuário. Os nomes desses arquivos são gravados no arquivo MIDIdocs. Cada arquivo MIDI é analisado, selecionando-se somente as notas e respectivas durações, e desconsiderando-se as outras informações presentes no padrão MIDI. De oito em oito notas e respectivas durações, uma função é chamada para realizar a transformada descrita na seção 2, retornando dois pontos com 7 coordenadas. Cada ponto é passado adiante para ser inserido na árvore correspondente (árvore de notas ou árvore de durações).

No módulo de Geração da Árvore, a cada ponto gerado pelo módulo de Análise Léxica a árvore é percorrida até encontrar-se o *nó externo* cuja posição corresponda aos valores de suas coordenadas. Como cada nó tem uma capacidade limite fixa, caso este esteja completo, uma rotina recursiva de subdivisão é chamada e re-estrutura a árvore, criando novos nós, filhos do nó em questão, agora transformado em *nó interno*.

Uma vez analisados todos os arquivos MIDI e construída a árvore, esta é armazenada em memória secundária, percorrendo-a em pré-ordem. Os arquivos gerados são: `Nós`, `NósInternos`, `Pontos` e `Ocorrências`. O arquivo `Nós` contém a sequência de nós internos e externos da estrutura em árvore, ordenados em pré-ordem. O arquivo `NósInternos` contém os DZEs dos nós internos. O arquivo `Pontos` contém, para cada nó externo da estrutura em árvore, a sua lista de pontos. O arquivo `Ocorrências` lista, para cada ponto, os códigos das músicas em que estes estão presentes, a trilha, o canal e a posição relativa em que o ponto se encontra na música.

4.2. Consulta

Inicialmente, este módulo carrega os arquivos gerados pelo módulo de indexação para a memória, reconstruindo a árvore. O servidor de consulta fica ativo, esperando que uma consulta seja solicitada. Ao receber uma cadeia de consulta, este módulo inicia a análise léxica da cadeia, identificando a seqüência de notas e respectivas durações. Em seguida, efetua a transformação das seqüências recebidas para os pontos correspondentes, segundo a fórmula citada na seção 2. Calcula-se os limites do subespaço que se quer considerar como região válida a partir do grau de similaridade escolhido pelo usuário: limite superior (soma-se o grau de similaridade às coordenadas do ponto consultado), e limite inferior (diminui-se o grau de similaridade das coordenadas do ponto consultado). Tomando estes dois pontos como referência, as árvores de nota e duração são percorridas. O resultado desta busca é um conjunto de pontos válidos, que são ordenados pelo grau de similaridade que apresentam em

relação ao ponto consultado.

Uma vez que a consulta simultânea de uma seqüência de notas e suas respectivas durações gera duas listas de pontos válidos, é preciso fundir estas duas listas em uma única lista ordenada. A ordenação final destas listas é feita aplicando-se uma fórmula de ponderação a cada par de pontos, isto é, para cada ponto de nota, há o seu correspondente ponto de duração (e vice-versa). Esta fórmula utiliza coeficientes baseados nos pesos fornecidos pelo usuário para as seqüências de notas e durações. A fórmula abaixo (Fig. 4.3) descreve como obter o coeficiente de similaridade para cada par de pontos (nota e duração), de modo a permitir a ordenação final.

$$coef\_similaridade = \sqrt{peso\_nota*(distância\_pt\_nota)^2 + peso\_duração*(distância\_pt\_duracão)^2}$$

Figura 4.3: Fórmula de ponderação para obter o coeficiente de similaridade a partir de uma sequência de notas e durações.

Para cada ponto do resultado, o arquivo de Ocorrências é consultado para obter a lista de músicas relacionadas com o ponto. E, para cada música selecionada, o arquivo MIDIdocs é consultado para obter o nome do arquivo MIDI. Além do nome do arquivo, a lista informa também em que canal, trilha e posição a sequência consultada foi encontrada na música. Todas estas informações são formatadas em HTML e devolvidas ao usuário que solicitou a consulta. A Figura 4.4 ilustra o relacionamento entre os módulos da Consulta.

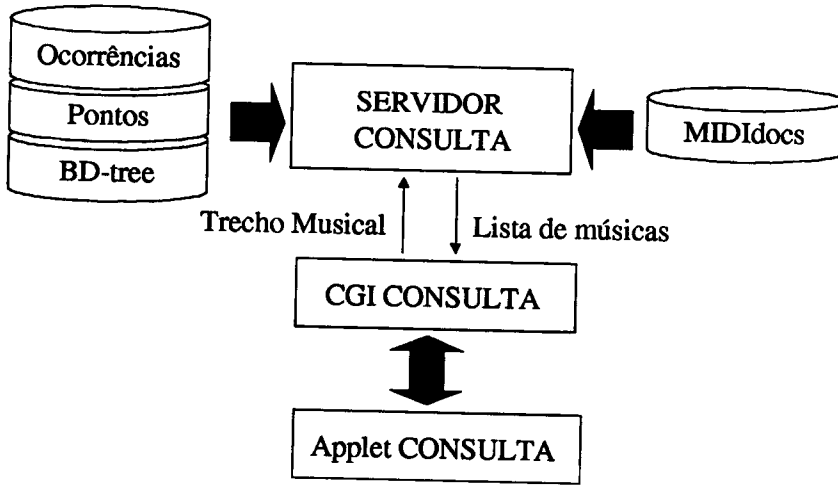
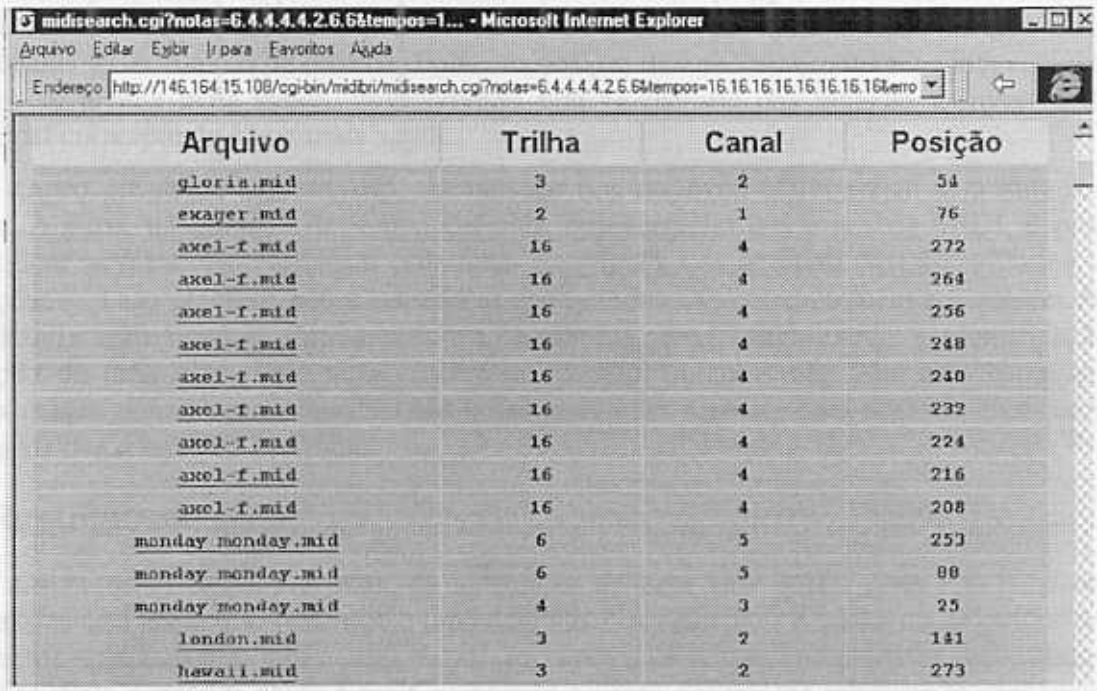


Figura 4.4: Módulo de Consulta

O Applet de Consulta oferece uma tela com uma barra de notas musicais: breve, mínima, semínima, etc. (Fig. 4.5), que o usuário seleciona e inclui em uma pauta sem referencial musical (não há a presença da clave). Isto é, ao estabelecer sua consulta, o usuário preocupa-se apenas com a disposição relativa das notas em uma pauta. Além disso, o usuário tem a opção de definir um grau de erro para sua consulta, para permitir uma flexibilidade maior no momento de realizar a consulta. Se houver uma certeza sobre a exatidão da consulta, este grau deve se aproximar de 0. Caso o usuário deseje uma recuperação mais abrangente, o grau de erro deve ser aumentado.

Figura 4.5: Tela do applet de consulta do MIDIZ

O CGI funciona apenas como interface entre o applet e o servidor de consultas. Ao receber o resultado gerado pelo servidor de consultas, já devidamente formatado em HTML, o CGI passa adiante a página HTML para o navegador web que solicitou a consulta. A Figura 4.6 ilustra o resultado de uma consulta.



Arquivo	Trilha	Canal	Posição
gloria.mid	3	2	54
exager.mid	2	1	76
axel-f.mid	16	4	272
axel-f.mid	16	4	264
axel-f.mid	16	4	256
axel-f.mid	16	4	248
axel-f.mid	16	4	240
axel-f.mid	16	4	232
axel-f.mid	16	4	224
axel-f.mid	16	4	216
axel-f.mid	16	4	208
monday_monday.mid	6	5	253
monday_monday.mid	6	5	88
monday_monday.mid	4	3	25
london.mid	3	2	141
hawaii.mid	3	2	273

Figura 4.6: Tela com a página HTML resultante de uma consulta feita ao MIDIZ

5. Implementação do Sistema MIDIZ

O Sistema MIDIZ foi implementado nas linguagens C++ e Java 1.1, em ambiente Unix. O módulo de Indexação e o servidor de Consultas foram implementados em C++, com o objetivo de garantir uma boa performance. O esquema de dados (*BD-Tree*) também foi

implementado em C++, a fim de ser utilizado tanto pelo módulo de Indexação quanto pelo servidor de Consultas. A interface da consulta foi implementada como um applet Java, que se comunica com um CGI também escrito em C++.

Durante a implementação do sistema MIDIZ, o tratamento dos arquivos MIDI usados em nossa base musical apresentou algumas questões: identificação da melodia, superposição de notas, durações inexatas e notas com intensidade próxima de zero. Essas questões originam-se no fato de que a maioria dos arquivos da base musical usada foi gerada a partir de gravações diretas de dispositivos MIDI. Já os arquivos MIDI originados a partir de programas de edição musical tendem a apresentar um nível mais alto de qualidade, como por exemplo, arquivos gerados a partir do software ENCORE©. A seguir discutimos tais questões, apresentando as soluções dadas em cada caso.

Um arquivo MIDI apresenta várias trilhas. Cada trilha contém uma seqüência de notas. No sistema MIDIZ a consulta é sempre feita a partir da entrada de uma seqüência simples de 8 notas, referentes à melodia principal da música. Desta forma, seria interessante que extraíssemos do arquivo MIDI somente a trilha que contém a melodia principal das músicas. Entretanto, o padrão MIDI não oferece meios para identificar de forma automática em qual trilha estão as notas que correspondem à melodia principal, o que dificulta bastante o processo de extração. Assim, optamos por analisar todas as trilhas, com exceção do canal usado pelo instrumento percussão (canal 10). Esta solução, embora simples, aumenta bastante a necessidade de recursos computacionais, como memória, disco e processamento, degradando o desempenho do sistema.

Uma ocorrência freqüente no formato MIDI é a superposição de notas, isto é, notas que soam simultaneamente. Em nosso processo de indexação optamos por extrair somente seqüências monofônicas de música. Portanto, sempre que encontramos este caso, escolhemos extrair só a nota mais aguda, considerando que na maioria das músicas a linha melódica principal corresponde à voz mais aguda.

Outro problema característico encontrado nos arquivos MIDI refere-se à duração das notas. A partir da unidade de tempo definida no cabeçalho dos arquivos MIDI, é possível identificar as durações relativas a cada nota. Entretanto, nos arquivos MIDI analisados, estas durações não são precisas, isto é, os valores encontrados não correspondem aos valores exatos da classificação musical (breve, mínima, semínima, etc.), dificultando a identificação da duração de uma nota. Para solucionar esta questão, optamos por usar uma estratégia de aproximação dos valores, considerando notas pontuadas somente para notas de duração igual ou maior que a unidade de tempo.

6. Resultados

Os números obtidos a partir da indexação de uma base MIDI com 100 músicas são mostrados na Tabela 6.1. Note que o total de pontos identificados é muito maior do que o total de pontos armazenados. Isso ocorre devido ao número de pontos coincidentes encontrados nas músicas. Esta diferença ainda é mais acentuada no caso dos pontos relativos às seqüências de durações de notas.

Com relação à consulta, fizemos algumas simulações para avaliar a eficiência do uso da Distância Euclidiana como coeficiente de erro na consulta de uma seqüência de notas e da sequência de durações destas. Nestas simulações introduzimos alguns erros, que consideramos mais comuns, descritos na Tabela 6.2. Os erros introduzidos na seqüência de notas resultam da subida uniforme ou crescente de no máximo 3 semitons. A subida uniforme

significa subir a mesma quantidade de semitons em todas as notas com erro, enquanto a subida crescente significa subir um número cada vez maior de semitons em cada nota com erro. Os tipos de erro considerados mais graves são os tipos G, H e J. Os erros introduzidos na sequência de durações resultam do aumento ou redução da duração das notas.

Total de músicas analisadas	10	20	40	100
Tempo de processamento (seg)	90	169	366	799
Total de arquivos MIDI processados (em Kbytes)	336	671,3	1.289	3.116
Tamanho total dos arquivos de índice gerados (em Kbytes)	1.215	2.228	4.394	16.328
Arquivo Nós (notas)	147	213	361	695
Arquivo Nós Internos (notas)	648	1.264	3.009	9.153
Arquivo Ocorrências (notas)	317.092	616.562	1.252.128	5.864.326
Arquivo Pontos (notas)	369.704	681.336	1.448.944	2.901.608
Arquivo Nós (duração)	113	139	199	463
Arquivo Nós Internos (duração)	358	614	1.248	4.685
Arquivo Ocorrencias (duração)	293.684	568.658	1.131.060	5.712.098
Arquivo Pontos (duração)	205.780	345.860	601.144	1.835.548

Tabela 6.1: Avaliação do processo de indexação (totais dos arquivos expressos em bytes)

Tipo de Erro	Descrição
A	Subida uniforme de semitons em apenas uma nota
B	Subida uniforme de semitons em um par de notas (2 notas com 2 intervalos)
C	Subida uniforme de semitons em 3 notas (1ª, 3ª e 5ª notas)
D	Subida uniforme de semitons em um par de notas (2 notas com 1 intervalo)
E	Subida uniforme de semitons em 3 notas (1ª, 2ª e 5ª notas)
F	Subida uniforme de semitons em um par de notas (2 notas seguidas)
G	Subida uniforme de semitons em 4 notas seguidas
H	Subida crescente de semitons em 2 notas seguidas
J	Subida crescente de semitons em 4 notas seguidas
M	Erro na duração de uma nota
N	Erro na duração de duas notas seguidas
P	Erro na duração de três notas seguidas
Q	Erro na duração de quatro notas seguidas

Tabela 6.2: Descrição dos tipos de erro introd uzidos na sequência de notas e durações

O gráfico da Figura 6.1 mostra o distanciamento dos pontos, à medida que cada tipo de erro é introduzido, com diferenças de 1, 2 e 3 semitons. Nota-se que na maioria dos erros com diferença de 1 semitom, a distância entre os pontos com erro e o ponto correto não ultrapassa 4 unidades. Quando a diferença de semitons aumenta, a curva torna-se mais acentuada, especialmente na região onde aparecem os tipos de erro mais graves (G, H e J).

O gráfico da Figura 6.2 mostra o distanciamento dos pontos, à medida que cada tipo de erro é introduzido, com o dobro, a metade e a quarta parte da duração original. Nota-se que na maioria dos tipos de erro em que se reduz a duração pela metade, a distância entre os pontos com erro e o ponto correto não ultrapassa 20 unidades. Quando os tipos de erro envolvem a

redução da duração à sua quarta parte, a distância já apresenta valores mais significativos.

Seja d_1 a diferença absoluta entre o dobro de um valor x e x, d_2 a diferença absoluta entre a metade de um valor x e x e d_3 a diferença absoluta entre a quarta parte de um valor x e x. Podemos dizer então que $d_2 < d_3 < d_1$. Sendo assim, em termos matemáticos, podemos dizer que os erros por redução de duração são considerados menos graves que os erros por aumento da duração. De acordo com este raciocínio, a distância Euclidiana mostra-se coerente, pois as distâncias geradas pelos erros em que ocorre duplicação da duração são bem maiores que as geradas por redução da duração.

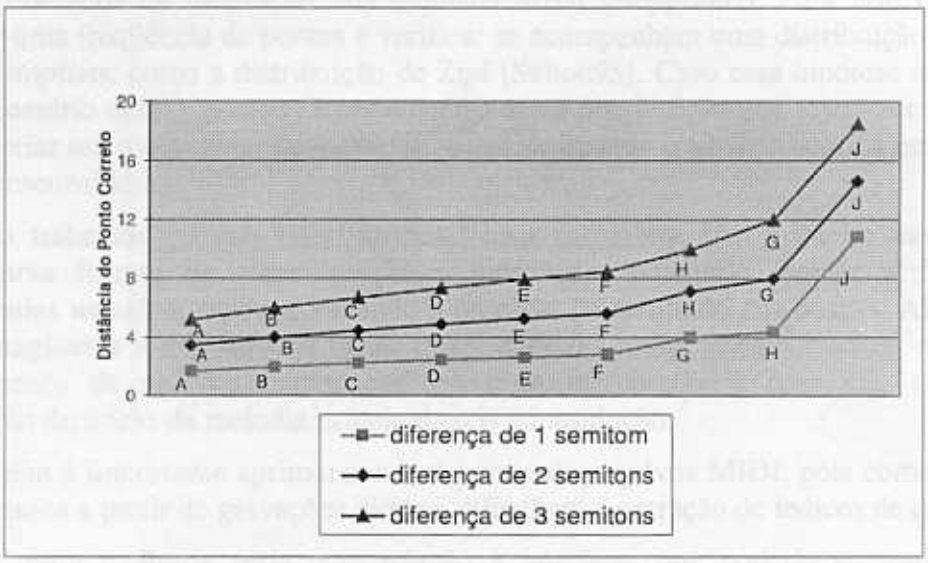


Figura 6.1: Comportamento da Distância Euclidiana com a introdução de erros na sequência de notas

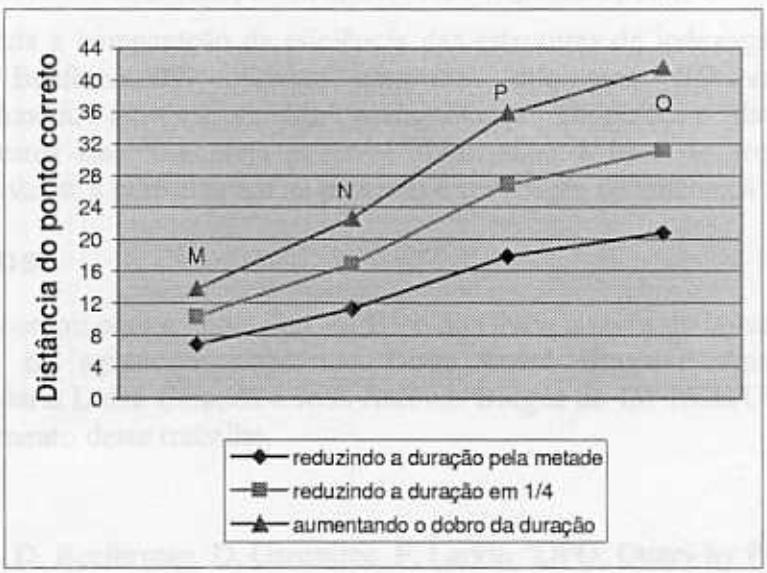


Figura 6.2: Comportamento da Distância Euclidiana com a introdução de erros na sequência de durações

7. Conclusão

Neste trabalho desenvolvemos um protótipo para indexação e recuperação por conteúdo de arquivos MIDI. A unidade de indexação usada, a "janela deslizante", tem como objetivo prover flexibilidade de consulta. No entanto, esta abordagem compromete a caracterização da

música. Isto é, ao considerarmos todos os trechos de uma música, podemos estar considerando trechos pouco significativos. Assim, ao submetermos uma consulta, nosso resultado é de alta revocação e baixa precisão.

Uma possível solução para a melhor caracterização das músicas, ainda não implementada nesta versão do MIDIZ, seria a avaliação da distribuição dos pontos pelas músicas. Os pontos que aparecem em grande número de músicas são sérios candidatos a compor uma *stoplist*. Isto é, assim como nas aplicações textuais as preposições e conjunções são desconsideradas na indexação dos textos (*stopwords*), os pontos populares também devem ser desconsiderados na indexação dos arquivos MIDI (*stoppoints*). Para isso é necessário estabelecer uma freqüência de pontos e verificar se acompanham uma distribuição adequada à criação de *stoplists*, como a distribuição de Zipf [Salton95]. Caso essa hipótese seja verdade, ainda é necessário definir pontos de corte dentro dessa distribuição que sejam aceitáveis pelos usuários e criar um mecanismo de *feedback* que avise quando o usuário solicita um trecho que o sistema desconsidera.

Como trabalhos futuros identificamos vários caminhos. Um primeiro caminho seria estudar outras formas de representação e indexação, buscando manter a flexibilidade fornecida pelas *wavelets*, mas aumentando a precisão do resultado das buscas. Além disso, é possível imaginar a introdução de maior quantidade de conhecimento musical por meio de reconhecimento de padrões como: padrões mais memorizáveis (por ex., refrões) e a determinação do início da melodia principal após a introdução.

Também é importante aprimorar o tratamento de arquivos MIDI, pois como são na sua maioria gerados a partir de gravações diretas, dificultam a extração de índices de qualidade.

Uma outra melhoria seria com relação à interface, que poderia ser estendida para permitir que o usuário cantarolasse um "lá lá lá", e recuperasse as músicas. Para isso, seria necessária a tradução do "lá lá lá" em padrão MIDI, e deste para os pontos.

Sugerimos ainda a comparação da eficiência das estruturas de indexação. Os sistemas MIDIZ e QPD [Beeferman97] utilizam estruturas diferentes, *BD-trees* e *R-trees*, respectivamente. Assim, com a devida permissão e cooperação das equipes de desenvolvimento destes sistemas, seria possível uniformizar a base de arquivos MIDI e realizar testes para avaliar a performance na precisão e revocação de ambos os sistemas.

Agradecimentos

Este trabalho contou com o apoio da CAPES e do CNPq através de bolsas de formação. Gostaríamos ainda de agradecer a Márcio Dias, André Braga e Ana Miccolis da COPPE/UFRJ e a Maria Luiza Campos e José Antônio Borges do IM-NCE/UFRJ pelo apoio dado no desenvolvimento deste trabalho.

Bibliografia

[Beeferman97] D. Beeferman, D. Greentree, P. Larkin "QPD: Query by Pitch Dynamics", 15-829 Course Project, Universidade Carnegie Mellon, 1997 (http://www.link.cs.cmu.edu/qpd/doc/note.html)

[Chou96] T. Chou, A. Chen, C. Liu - "Music Databases: Indexing Techniques and Implementation" - Anais do International Workshop on Multimedia Database Systems, pp. 46-53, 1996

[Dirst94] M. Dirst, A. Weigend - "On Completing J. S. Bach's Last Fugue" - Em A. Weigend & N. A. Gershenfeld (ed.), "Time Series Prediction:

Forecasting the Future and Understanding the Past", pp. 151-172, Addison Wesley, 1994

[Dandamudi86] S. Dandamudi & P. Sorenson, "Algorithms for BD Trees", Software Practice and Experience, Vol. 16(12), pp 1077-1096, Dezembro, 1996

[Derakhshan89] M. Derakhsham, "A Developmente of the *Grid File* for the Storage of Binary Relations", Tese de doutorado, Universidade de Londres, Março, 1989

[Faloutsos94] C. Faloutsos, M. Ranganathan, Y. Manolopoulos "Fast Subsequence Matching in Time-Series Databases", SIGMOD'94, Minneapolis, Minnesota, EUA, 1994

[Faloutsos96] C. Faloutsos, "Searching Multimedia Databases by Content", Kluwer Academic, 1996

[Figueiredo97] M. Figueiredo & C. Traina Jr & A. Traina, "Representação e Recuperação Baseada em Conteúdo de Partituras Musicais em Bases de Dados Orientadas a Objetos", IV Simpósio Brasileiro de Computação e Música, Brasília, DF, 1997

[Freeston87] M. Freeston, "The *BANG file*: a new kind of *grid file*", Anais do ACM SIGMOD, 1987

[Ghias95] A. Ghias & J. Logan & D. Chamberlin & B. Smith, "Query by Humming – Musical Information Retrieval in an Audio Database", ACM Multimeida 95, San Francisco, California, 1995

[Guttman84] A. Guttman, "*R-trees*: A Dynamic Index Structure for Spatial Searching", Anais SIGMOD, pp. 47-57, Junho, 1984

[Heckroth95] J. Heckroth, "Tutorial on MIDI and Music Synthesis" - The MIDI Manufacturers Association, 1995

[Nievergelt97] J. Nievergelt & P. Widmayer, "Spatial Data Structures Concepts and Design Choices", em "Algorithms Foundations of GIS", eds. van Kreveld, Nievergelt, Roos, Widmayer, Capítulo 6, pp 153-198, LNCS 1340, Springer Verlag, 1997

[Rumbaugh91] J. Rumbaugh et al, "Object-Oriented Modeling and Design", Prentice Hall, 1991

[Salton95] G. Salton, "Automatic Text Processing", Addison-Wesley, Reading, Massachusetts, 1995.

[Samet90] H. Samet, "The Design and Analysis of Spatial Data Structures", Addison-Wesley, 1990