Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Indexação de Multimédia com base no SOLR

Por

Marco Alexandre Figueira Teixeira

N.º 28128

Dissertação apresentada na Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa
para obtenção do grau de Mestre em Engenharia Informática

Orientador

Prof. Doutor João Miguel da Costa Magalhães

Lisboa

2010

Agradecimentos

Em primeiro gostaria de agradecer a toda a minha família e em especial aos meus pais, pelo seu apoio constante nos momentos mais complicados por que passei na elaboração desta dissertação, pelas horas mal dormidas, pelos transtornos que lhes causei, em suma obrigado por tudo o que proporcionaram quer financeiramente, quer moralmente, sem vocês não seria possível alcançar este grande objectivo.

Em segundo lugar, gostaria de prestar o meu agradecimento ao meu orientador, Professor João Magalhães, não só por possibilitar-me a realização de uma dissertação na área de Multimédia, mas também pela sua incansável paciência e dedicação demonstrada ao longo deste ano. Queria ainda pedir-lhe as minhas sinceras desculpas pelas dores de cabeça que lhe causei, nas constantes revisões da dissertação.

Como não poderia deixar de ser, um agradecimento especial a todos os meus amigos não só pelo facto de me terem aturado todos estes anos, mas pelo apoio constante demonstrado nos momentos mais complicados, estando sempre dispostos a ajudar no que fosse preciso. Refiro-me a todos sem excepção, mas gostaria de expressar esse agradecimento especialmente aqueles que estiveram directamente envolvidos, nomeadamente: André Simões, Nuno Martins, André Rosa, Daniel Ramos, Joana Dias, Gustavo Marquês, Filipe Afonso, Pedro Boavida. Pessoal o meu Muito Obrigado!

Por último, queria fazer uma pequena dedicatória a dois grandes homens que, infelizmente, não se encontram entre nós. Refiro-me aos meus avôs Júlio Teixeira e José Figueira. Apesar da sua ausência física, eles contribuíram, em muito, na minha luta constante de alcançar este objectivo, relembrando-me sempre dos seus ensinamentos e perseverança, aos quais moldaram as suas vidas e que tento, nas minhas modestas possibilidades, continuar com o seu legado. O meu agradecimento profundo e até sempre!

Sumário

A pesquisa eficiente de multimédia é hoje em dia uma necessidade que cresce com a vasta quantidade de informação disponível nos diversos serviços *Web*, e.g., *YouTube*[1], *Flickr*[2], *Olhares*[3].

Tanto as bases de dados tradicionais, com é o caso do *MySQL*, como as bases de dados de texto (e.g. *SOLR*), não fornecem uma solução ideal para o tipo de dados multimédia. Deste modo, o objectivo deste projecto consiste em estender o motor de pesquisa *SOLR* para permitir o suporte de dados multimédia.

Assim, iremos desenvolver estruturas de dados que permitam a indexação e procura desses dados multimédia. Para tal, será estudado um conjunto alargado de estruturas de dados que permitam uma procura rápida e medidas de semelhança entre documentos multimédia que permitam localizar os documentos mais semelhantes. De notar que as medidas de semelhança partem do conceito genérico de distância num espaço de características multimédia. Vários factores influenciam a escolha das estruturas de dados e as medidas de semelhança. Desta forma, será identificada, qual das diversas estruturas e medidas de semelhança é a combinação ideal para responder a uma dada pesquisa multimédia.

Este estudo será seguido do desenvolvimento de estruturas de dados adequadas a pesquisas rápidas e precisas para diversos tipos de dados multimédia. Será também desenvolvida uma aplicação de pesquisa com uma interface para visualização dos dados que permitirá testar os diferentes métodos investigados e realizar testes de utilizador.

Palavras-chave:

- Multimédia
- Bases de dados
- *SOLR*
- Medidas de semelhança
- Indexação de multimédia
- Procura em multimédia

[1] http://www.youtube.com
[2] http://www.flickr.com
[3] http://www.olhares.com

Abstract

The efficient search in multimedia is today a necessity that grows with the vast amount of information available in the various *Web* services, e.g., *YouTube*[4], *Flickr*[5], *Olhares*[6].

Either traditional databases, as is the case of *MySQL*, or text databases (e.g. *SOLR*), do not provide an ideal solution for the multimedia data type. Therefore, the objective of this dissertation is to extend the *SOLR* search engine to enable support for multimedia data.

Thus, we will analyze data structures that allow the indexing and search of multimedia data. This will study a broad range of data structures that allow a quick search and measures of similarity between multimedia documents that allow the location of the most similar documents. Note that the measures of similarity departing from the generic concept of distance in an area of multimedia features. Several factors influence the choice of data structures and measures of similarity. Therefore, it will be identify, which of the various structures and measures of similarity is the ideal combination to answer to a given multimedia search.

In this dissertation, the study will be followed by implementation of data structures suitable for fast and accurate searches for various types of multimedia data. It will also develop a search application with an interface for viewing data that will test the different methods investigated and perform user tests.

Keywords:

- Multimedia
- Databases
- *SOLR*
- Measures of similarities
- Multimedia Indexing
- Search in multimedia

[4] http://www.youtube.com
[5] http://www.flickr.com
[6] http://www.olhares.com

Índice

Índice de Tabelas

Índice de Figuras

1

Introdução

Ao longo dos tempos, as sociedades sofreram grandes evoluções no seu comportamento e no seu modo de estar devendo-se, essencialmente, às inovações tecnológicas. Hoje em dia, a tecnologia continua a ter este papel de mudança de hábitos. Estando nós actualmente numa era de globalização, onde a troca de informação e a comunicação entre indivíduos espalhados pelo globo são factores extremamente importantes, os conteúdos multimédia assumem um papel proponderante nessa comunicação e interacção. Esta importância pode ser verificada através de sites de agências noticiosas, de redes sociais, de compras online, Figura 1.

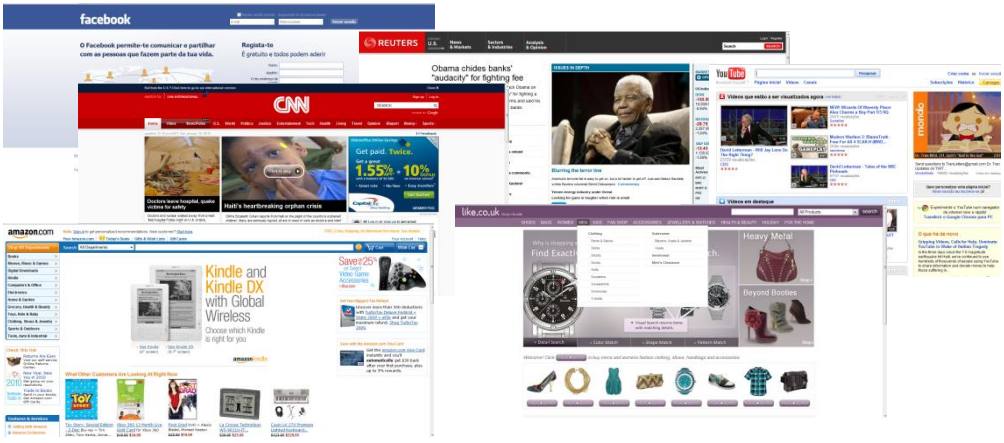

Figura 1 – Exemplos de websites que contêm conteúdos multimédia e onde é possível realizar pesquisas sobre estes.

O processo de gestão e acesso a estes conteúdos baseia-se, sobretudo, na utilização da informação textual associada. Este facto constitui uma limitação das potencialidades de pesquisa sobre estes recursos, uma vez que não tiram partido de todo o conhecimento que os conteúdos multimédia podem oferecer. Por exemplo, pesquisas através de cor, textura, objecto/s representativo/s, contraste, etc., permitiriam obter um conjunto de documentos

similares que fosse ao encontro do que os utilizadores pretendiam (e.g. utilizar uma imagem como *query* sobre um repositório e obter-se, de seguida, todas as imagens que lhe sejam semelhantes).

Verificando este tipo de problemas, foi desenvolvido um sistema para resolver estas questões, denominado por *QBIC - Query by Image and Video Content* (Flickner, Sawhney et al. 1995). Este sistema foi concebido para pesquisar conteúdos multimédia (imagens e vídeos), através de dois paradigmas diferentes. O primeiro paradigma possibilita a utilização de imagens ou vídeos como exemplos do tipo de conteúdo pretendido, enquanto que o segundo permite a utilização de uma linguagem gráfica para realizar as pesquisas (e.g. desenhos, selecções, etc.).

Estes dois paradigmas de pesquisa são suportados por duas fases de processamento de multimédia do sistema *QBIC*: *database population* e *database query*. Durante a fase de *population*, os descritores (*features*) que descrevem o conteúdo das imagens e dos vídeos, são extraídos e guardandos no repositório. Na fase de *query*, o utilizador produz uma *query gráfica* (i.e., elabora um desenho, utiliza uma dada imagem, utiliza uma cor, etc.) e envia-a para um motor de comparação, de modo a obter imagens e vídeos cujos descritores são semelhantes aos da *query*.

Relativamente ao tipo de descritores, é habitual usar o valor médio das cores, para identificar imagens e/ou objectos para uma dada cor dominante. Não obstante este método, é possível utilizar histogramas de cor, retornando conteúdos com a mesma distribuição de cores. O processo de verificação de semelhança é realizado por meio de funções de distância entre os descritores da *query* e os do repositório.

Finalmente, torna-se ainda possível realizar pesquisas, de modo a obter vários objectos em imagens que contêm múltiplas cores e texturas, onde os resultados obtidos derivam da combinação das distâncias e das cores.

Compras online são aplicações recentes destas tecnologias. O site "like.com" utiliza técnicas de pesquisa que permitem obter artigos de vestuário que apresentem padrões semelhantes, por exemplo de cores e texturas, aos descritos na pesquisa, Figura 2.

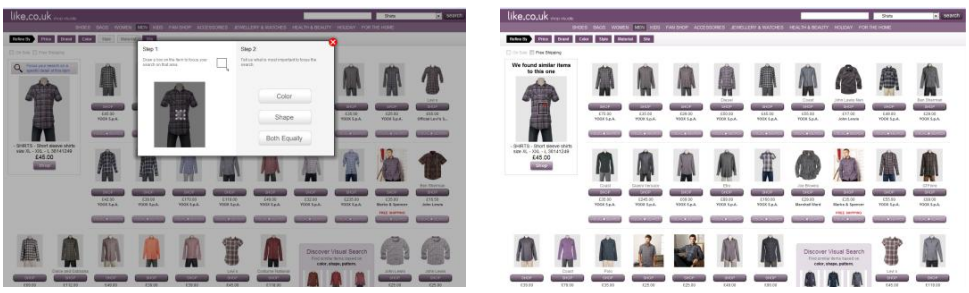

Figura 2 – Pesquisas por semelhança utilizada no site "like.com" (esquerda), seguido do resultado obtido, utilizando a cor como *query* de pesquisa (direita)**.**

1.1 Tipos de Pesquisas

É fundamental que o repositório contenha o seu conteúdo bem indexado, sendo necessário conhecer, à partida, o tipo de *queries* que será utilizado e o modo como os dados devem estar organizados. Isto permite optimizar o processo de acesso aos dados, quer em termos de eficácia[7], quer em termos de eficiência[8] na obtenção dos resultados.

Na análise realizada a vários sistemas de pesquisa, verificámos quais os tipos de pesquisa suportados por aplicações de gestão de multimédia. Um exemplo destas pesquisas pode ser observado na imagem abaixo:

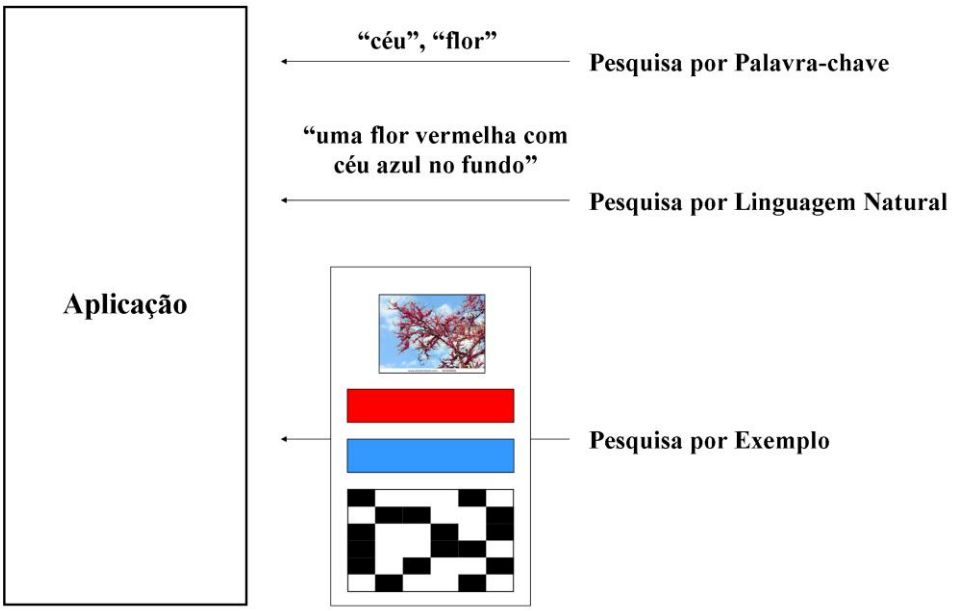

Figura 3 – Ilustração dos 3 paradigmas de pesquisa que os utilizadores podem explorar para obter o resultado desejado.

1.1.1 Pesquisa por Palavras-Chave

A tendência genérica de acesso aos dados num repositório de dados, baseia-se na utilização de *tags* e/ou *keywords* (também designadas por *descritores de alto nível*). No caso dos repositórios com conteúdo multimédia, a utilização deste tipo de *queries* na obtenção de resultados também é comum. Contudo, consiste num processo algo limitado, pois não possibilita a utilização mais alargada dos descritores existentes nos dados multimédia.

Para ultrapassar esta limitação, novos tipos de *queries* são necessários para a colmatar (e.g. a utilização de linguagem natural, ou a utilização de exemplos multimédia - ex: a utilização de descritores de cores e de texturas, etc., permitindo que as pesquisas sejam mais fidedignas e

[7] Eficácia - precisão dos dados gerados.
[8] Eficiência - rapidez na obtenção de dados.

precisas.

1.1.2 Pesquisa por Língua Natural

A transposição da linguagem utilizada pelos humanos na sua comunicação para a construção de *queries* sobre um/a dado/a repositório/base de dados, tem vindo a ser criado como uma tentativa de melhorar a interacção dos utilizadores com as aplicações. No entanto, este tipo de *queries* apresentam uma dificuldade acrescida na construção dos algoritmos de pesquisa, uma vez que estes algoritmos não estão, na sua base, preparados para tratar com este tipo de linguagem.

Uma forma encontrada para resolver este problema consiste, na identificação de palavras-chave (*tags*/*keywords*) existentes na *query* e, assim, executar as pesquisas através das mesmas.

No entanto, existem outras técnicas podem ser igualmente aplicadas para solucionar o problema que advém destas pesquisas. A solução encontrada sugere a utilização de ontologias (Town and Sinclair 2001). Para tal, o processo utilizado permite detectar os conceitos que estão envolvidos na *query* e, a própria ontologia, guarda a informação desses conceitos e das suas relações.

1.1.3 Pesquisa por Exemplo

Apesar da aproximação anterior favorecer a diversificação de pesquisas num repositório, não permite que abranja a totalidade dos descritores associados a dados multimédia.

A utilização de exemplos em *queries* sobre aplicações que gerem informação multimédia e os respectivos descritores, disponibiliza aos utilizadores uma elevada variedade de possibilidades para realizar pesquisas sobre esse repositório.

Uma das possibilidades consiste na construção de *queries* utilizando cores (ou seja, indicar uma cor – e.g. em formato hexadecimal - e descobrir quais os conteúdos multimédia que contêm essa cor), texturas, formas de objectos (e.g. elaborar um esboço de um objecto e identificar os recursos que contêm objectos que se aproximam desse esboço), histogramas (ou seja, a distribuição de cores numa imagem).

Para além de utilizar esta informação nas pesquisas destes recursos, uma outra possibilidade será a utilização imagens do repositório na construção de *queries* de pesquisa, o que torna o tempo de resposta mais curto pois a imagem foi previamente processada.

1.2 Pesquisa por Imagem Semelhante

Esta dissertação irá explorar o conceito de semelhança entre imagens, através da quantificação do "*grau de proximidade*" entre diferentes imagens. O Capítulo 3 apresentará uma aplicação de demonstração de pesquisa por semelhança. O Capítulo 4 descreverá o

serviço *Web* implementado para suportar as técnicas de pesquisa em multimédia onde os descritores de baixo nível (cores, texturas, contraste, etc.) contêm a informação necessária para estabelecer graus de semelhança através de diferentes funções de distâncias.

1.2.1 Pesquisas Lineares e Complexidade Computacional

Para o cálculo da semelhança entre duas imagens é necessário ter em conta a dimensão do vector do descritor das imagens. Numa primeira abordagem, poder-se-ia realizar este cálculo de semelhança entre a imagem exemplo fornecida pelo utilizador e todas as outras imagens do repositório, através de uma função de distância. Contudo, ao analisarmos este processamento em concreto, verificamos que é inviável.

Imaginemos a seguinte situação. O nosso repositório é constituído por um conjunto de imagens (na ordem dos milhares) e cada imagem contém um descritor com 100 dimensões. Ao processarmos cada imagem para verificar se é semelhante ou não, levaríamos $O((n \times m) \times d)$, onde n corresponde ao número de imagens existentes no repositório, m o número de imagens a comparar ($m = n - 1$) e d a dimensão do descritor. Caso o tamanho dos descritores aumentasse ainda mais, então este processo torna-se incomportável.

1.2.2 Indexação de Alta Dimensão

O exemplo da secção anterior ilustra como pesquisas lineares não são soluções viáveis para dados de alta dimensão[9] como é o caso de dados multimédia. A solução consiste na utilização de estruturas de indexação de dados de alta dimensão em que as imagens estejam organizadas por semelhança. Tratam-se de estrutras em árvore ou em *clusters* que permitem navegar na base de dados com um custo sub-linear. Esta é a solução tradicionalmente usada em dados de baixa dimensão[10] onde se utilizam *B-Trees* ou *R-Trees* (ver Capítulo 2). No entanto, não existe uma solução que se destaque para dados de alta dimensão como é o caso de dados multimédia.

Consequentemente, a dificuldade desta solução reside na identificação da árvore que melhor reflecte a organização por semelhança que um humano faria e com um custo computacional baixo. O Capítulo 5 desta dissertação apresenta uma análise exaustiva de diferentes estruturas de indexação.

1.3 Objectivo

O objectivo desta dissertação é a implementação de uma solução de software completa para indexar e pesquisar imagens por semelhança. Para este fim, irei implementar (i) uma

[9] Os dados são representados segundo um vector com *n* dimensões/posições.
[10] Os dados são representados por meio de informação textual.

aplicação de pesquisa por semelhança, (ii) um serviço *Web* que possa ser utilizado por vários clientes para pesquisar um repositório e, (iii) implementar e comparar várias estruturas para indexação de dados de alta dimensão.

1.4 Organização

Esta dissertação está organizada do seguinte modo:

- **Capítulo 2: Trabalho Relacionado** – Neste capítulo serão analisadas diferentes funções de distâncias, o motor de pesquisa de texto *SOLR*, e estruturas de indexação de dados de alta dimensão incidindo a nossa análise no processo de construção e de pesquisa das mesmas.

- **Capítulo 3: Aplicação de Navegação e Pesquisa em Multimédia** – Este capítulo destina-se à apresentação da interface desenvolvida para visualização e interacção entre os metadados[11] de multimédia (textuais e de baixo nível).

- **Capítulo 4: Serviço de Indexação e Pesquisa** – Este capítulo descreve o motor de pesquisa desenvolvido e como suporta pesquisas de dados textuais e dados multimédia. Neste capítulo será ainda descrito as *APIs* de cada serviço.

- **Capítulo 5: Análise de Métodos de Indexação** – Este capítulo apresenta um conjunto de resultados referentes a um conjunto de testes efectuados sobre as estruturas de indexação de dados de alta dimensão, segundo a descrição do Capítulo 2.

- **Capítulo 6: Conclusões e Trabalho Futuro** – Por fim, serão apresentadas as conclusões referentes aos testes efectuados, destacando os principais aspectos desta dissertação e proposta de alguns melhoramentos sobre o trabalho realizado.

[11] Conjunto de informação que permite espicificar a informação dos conteúdos, como também a relação entre conteúdos.

2

Trabalho Relacionado

A breve análise ao funcionamento do sistema *QBIC*, realizada no capítulo anterior, colocou a questão de se utilizar outros processos de pesquisa na procura de conteúdo multimédia, para além do simples uso de conteúdo textual (palavras-chave ou linguagem natural). Dessa análise, é possível inferir que a utilização de descritores de baixo nível (ex: cores, texturas, contraste, etc…) permitem realizar os processos de pesquisa descritos no *QBIC*.

Neste capítulo, o nosso estudo irá estar centrado no modo de acesso à informação de baixo nível, definindo o espaço de procura, como também, relacionar a informação de baixo nível entre conteúdos multimédia, conseguido por meio de funções de semelhança ou, por outras palavras, através do cálculo de distâncias entre conteúdos. Ao mesmo tempo, os resultados obtidos dessas funções, permitem que seja possível construir índices baseados em estruturas de dados especialmente desenhadas para o tipo de espaço, onde os conteúdos multimédia estão inseridos (espaço particionado[12]), que também serão analisadas neste capítulo.

Ao analisarmos as bases de dados tradicionais e de texto, como são os casos do *MySQL*, da *Lucene* e outras, utilizam principalmente três métodos para correspondência de *queries*:

- **Exact match query** – permite descobrir, se para uma dada *query*, existe pelo menos um objecto que respeite as seja idêntico à *query*.

- **Range query** – consiste num tipo de pesquisa, onde, para uma dada *query*, é possível identificar todos os objectos que estão compreendidos até uma dada distância da *query*. Do mesmo modo, as *queries* por semelhança (*Similarity query*) estão incluídas neste grupo de pesquisas.

[12] O conteúdo é subdividido de modo a optimizar a funcionalidade dos índices, segundo o desempenho e o espaço que ocupam em disco.

- **Nearest-Neighbor search** – para uma dada *query*, é possível encontrar o objecto/item mais próximo ou semelhante do objecto/item da *query*.

2.1 Espaços de Procura

Na análise dos diferentes tipos de *queries*, a correspondência com os dados no repositório é extremamente importante. Para o caso em estudo, é necessário analisar os tipos de correspondências existentes:

- **Espaço de procura** – os conteúdos multimédia (d), são representados por um vector

$$d = \left(d_{f,1}, \dots, d_{f,L}\right) \in [0,1]^L,$$

com L *features*, de um dado conteúdo multimédia. Cada componente de d, corresponde à probabilidade de uma *feature* f_i existir nesse conteúdo.

- **Cálculo do descritor** – o cálculo do descritor pode ser realizado automaticamente ou manualmente.

- **Cálculo da semelhança** – dado um espaço de *features*, definido por meio de um descritor, define-se a desigualdade entre dois documentos multimédia como,

$$dissim: [0,1]^L \times [0,1]^L \rightarrow \mathbb{R}_0^+,$$

a função no espaço L dimensional, retornando a distância/semelhança entre dois descritores.

2.2 Funções de Semelhança

As funções de desigualdade assumem três tipos de espaços (os geométricos[13], os baseados em histogramas[14] e os baseados em espaços probabilísticos[15]). Estas funções assumem, também, que o espaço é linear ou que os descritores são independentes. O cálculo das desigualdades é baseado em funções $D(a, b)$, não correspondendo directamente ao conceito geral de distância, uma vez que podem violar uma das seguintes condições:

[13] São espaços aos quais a relação entre conteúdos estabelecesse por meio da noção directa de distância.
[14] O espaço é dividido segundo um conjunto de *bins*, onde cada *bin* pretende definir uma propriedade dio conteúdo.
[15] Espaços nos quais a noção de distância não por meio de um número real positivo, mas por meio de funções de distribuição.

1. ***Não negatividade*** – $D(a,b) \geq 0$
2. ***Simetria*** – $D(a,b) = D(b,a)$
3. ***Desigualdade triangular*** – $D(a,b) \leq D(a,c) + D(c,b)$
4. $D(a,b) = 0$, para $a = b$

Com a utilização precisa de palavras-chave fornecidas pelos utilizadores, é possível aferir a precisão da semelhança semântica sobre o cálculo do descritor, assim como determinar o quanto foi devido às funções de desigualdade.

A computação hierárquica de desigualdade em todo o repositório de dados é bastante dispendiosa, uma vez que o processo tem complexidade linear. No entanto, por meio da identificação de amostras representativas, é possível diminuir custos do cálculo das desigualdades.

2.2.1 Espaços Geométricos

As funções de semelhança geométrica, são executadas em espaços de alto nível e cada função está implementada como uma função de distância, segundo algumas especificações.

2.2.1.1 Distância de Minkowski

A *distância de Minkowski* entre uma *query* q e um documento do repositório d_W, é definida por:

$$D_{Minkowski}(q_W, d_W) = L_p(q_W, d_W) = \left[\sum_{i=0}^{L} \left|q_{W,i} - d_{W,i}\right|^p\right]^{1/p},$$

onde o índice i diz respeito ao conceito i e p corresponde a um parâmetro livre $p > 1$.

L_p constitui numa métrica muito pouco realista, para valores $p < 1$, uma vez que viola a desigualdade triangular. Contudo oferece resultados bastante interessantes.

Foi provado também (Howarth and Rüger 2005) que para valores compreendidos entre [0.0, 1.0], ofereciam melhor desempenho para vários tipos de descritores.

2.2.1.2 Distância de Manhattan

A *distância de Manhattan* $(p = 1.0)$, é nem mais nem menos que soma acumulada das distâncias em cada dimensão,

$$D_{Manhattan}(q_W, d_W) = L_1(q_W, d_W) = \sum_{i=0}^{L} \left|q_{W,i} - d_{W,i}\right|.$$

Trata-se de uma distância em que considera comprimento de todos os caminhos mais curtos que ligam a q_W e d_W, segundo linhas paralelas.

2.2.1.3 Distância Euclidiana

A *distância euclidiana* (ou seja, a *distância de Minkowski* com $p = 2.0$) corresponde à noção de distância entre dois pontos num espaço de coordenadas real.

$$D_{Euclidean}(q_W, d_W) = L_2(q_W, d_W) = \sqrt{\sum_{i=0}^{L} (q_{W,i} - d_{W,i})^2}$$

2.2.1.4 Chebyshev

A *distância de Chebyshev* (ou seja, a *distância de Minkowski* com $p = \infty$) mede a distância máxima em cada uma das dimensões existentes.

$$D_{Chebyshev}(q_W, d_W) = L_\infty(q_W, d_W) = \max_{0 \le i \le L} |q_{W,i} - d_{W,i}| .$$

2.2.1.5 Distância de Co-seno

Na semelhança de co-seno, é possível definir a independência entre dois vectores por meio do ângulo definidos por estes e, deste modo, permite que quaisquer pontos dos dois vectores apresentem ou não direcções similares. Deste modo, a função terá o seguinte fórmula:

$$D_{Cosine}(q_W, d_W) = cos(q_W \measuredangle d_W) = 1 - \frac{q_W . d_W}{\|q_W\| . \|d_W\|}$$

A correlação geométrica pode também ser utilizada para medir essa independência.

A *distância co-seno* corresponde a um caso especial do *Coeficiente da correlação de Pearson*, onde os dados são normalizados para obter a média zero.

2.3 Indexação de Texto

Estando a análise desta dissertação centrada na utilização de descritores de baixo nível, a exclusão da informação textual seria de um certo modo um retrocesso, na medida em que não seria possível analisar o comportamento destes dois tipos de informação quando integradas em conjunto. Assim, recorreu-se a um dos motores de pesquisa textual existentes que pode ser utilizado em múltiplas plataformas, *SOLR*[16].

[16] Página oficial - http://lucene.apache.org/solr/

2.3.1 SOLR – Searching On Lucene w/ Replication

O *SOLR* consiste num servidor baseado no motor de pesquisa de texto *Lucene*, sendo facilmente integrado em aplicações *Web*. Contém um método de pesquisa facetado, que podes ser aplicado ao seu processo normal de pesquisa[17], permitindo o realce e suporte a múltiplos formatos de output, onde estão incluídos *XML/XSLT* e *JSON*. É incluída uma interface de administração *Web* e um conjunto de funcionalidades básicas, que podem ser estendidas consoante as necessidades da aplicação.

As colecções de objectos utilizados pelo *SOLR* são indexadas através das definições fortemente estruturadas sobre os campos (*fields*[18]). Cada documento é representado por um conjunto de um ou mais campos, onde cada um corresponde a um conteúdo ou metadados. Os campos podem tomar o valor de *strings*, *números*, *booleanos*, *datas*, entre outros tipos, que podem ser adicionados ao índice. Para além da definição destes tipos, os campos também podem ser descritos através de um conjunto de opcionalidades, indicando ao *SOLR* o modo de tratamento dos dados durante o processo de indexação e de pesquisa.

2.3.1.1 Schema

A construção do índice está dependente do *schema*. Através deste ficheiro é possível definir todos os dados a serem utilizados, de modo a possibilitar a representação dos documentos a indexar. De um modo geral, o *schema* está organizado em três secções: *types*, *fields* e *outras declarações* – podendo ser representado através do esquema definido na Figura 4.

2.3.1.1.1 Types

Esta secção permite informar o *SOLR*, sobre a forma de processar os campos definidos no *schema*. São utilizados valores semelhantes a *sint*, *boolean*, entre outros, cujo principal objectivo é guardar tipos de dados primitivos no *SOLR*.

Através desta informação estruturada nos campos, permite que se defina o procedimento a efectuar sobre os conteúdos durante a fase de indexação, sem a intervenção do programador.

2.3.1.1.2 Fields

Uma vez definidos os *types*, pode-se proceder à elaboração dos *fields* que irão ser utilizados no repositório. Esta parte do *schema* é composta por um ou mais campos, conforme necessidade requerida pelo repositório que se pretende instalar. Com base nesta informação, os diversos campos são utilizados para representar os documentos durante a fase de indexação e de pesquisa.

---

[17] As descrições e procedimentos serão estudados, mais à frente, na secção **2.4.1.4**.
[18] Trata-se de uma *keyword* de *sintax* do *SOLR*, cuja sua descrição e funcionalidades serão descritas mais à frente (secção **2.4.1.1.2**).

Analisando cada campo, é possível inferir o modo como estes irão ser processados:

- **Indexed (indexados)** – são campos que permitem a pesquisa e ao mesmo tempo a ordenação. Por outro lado, é possível aplicar um processo de análise que permita alterar o conteúdo e, deste modo, melhorar ou alterar os resultados.

- **Stored (guardados)** – o conteúdo de um campo, deste tipo, é guardado no índice. É bastante útil para obter conteúdo para visualização.

- **Analyzed (analisados) –** dependendo do que foi definido nos *types*, o *SOLR* irá processar o conteúdo de acordo com as operações que foram especificadas nesses tipos.

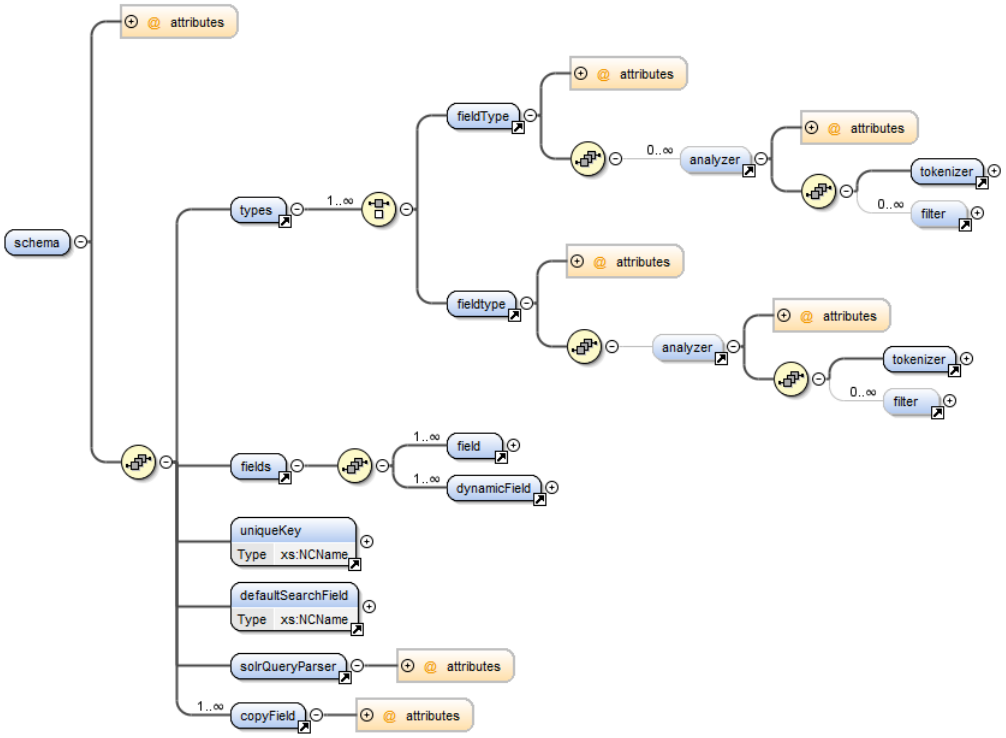

Figura 4 – Representação dos principais componentes do ficheiro *schema*.

Para além da definição dos tipos primitivos, é possível definir outros tipos de atributos especiais, nos quais se encontram:

- **multiValue** – permite que um documento possa ter mais do que um campo com o mesmo identificador.

- **omitNorms** – utilizado para omitir normas (permite desactivar a normalização do comprimento do campo, como também melhorar o tempo de indexação do

mesmo), salvaguardando assim, a memória em campos que não afectam o desempenho (e.g. no cálculo de facetas).

- **dynamicField** – consiste num campo especial, que pode ser adicionado a qualquer momento num documento, com os atributos definidos através de uma declaração de um *field*. Desta forma, torna-se desnecessário criar este campo *a priori* no *schema*.

2.3.1.1.3 Outras declarações

A última secção do *schema*, contém várias declarações relacionadas com os campos e com as possíveis pesquisas a serem realizadas, sendo a principal declaração a *uniqueKey*. Através da declaração deste identificador, o *SOLR* determina quando é que um documento é adicionado ou actualizado no repositório. Para tal, o *SOLR* verifica se a *uniqueKey* no documento, pertence ou não a um documento no repositório.

Uma outra declaração, denominada por *copyField*, permite criar campos (*fields*) "*all*", sem a necessidade de adicionar manualmente todo o conteúdo do documento, para um dado *field* em separado. Assim, a cópia dos campos de um documento, torna-se num processo conveniente para a indexação do mesmo conteúdo com diferentes métodos de análise de texto.

2.3.1.2 Análise de Texto

O *SOLR* disponibiliza métodos de pré-processamento de texto que a aplicação cliente pode executar antes de proceder à sua indexação. Em traços gerais, o *Analyzer*[19] é composto por um *tokenizer* com, pelo menos, um filtro para *tokens*. Esse *tonkenizer* fica responsável pela obtenção das palavras a indexar e, após a aplicação dos filtros, modificam-se ou removem-se essas palavras, antes de se proceder à indexação. De um modo global, se a aplicação necessitar de uma análise mais específica, o *SOLR* disponibiliza mais *tokenizers* e filtros, de modo a ir ao encontro das necessidades da mesma.

Por sua vez, os *Analysers* são igualmente utilizados no processamento de *queries* durante a pesquisa. Para tal, essa análise tem de ser executada tanto na *query*, como no documento a ser indexado. Uma exemplificação de um *Analyser* pode ser observada através da Figura 5.

Analisando a Figura 5 observa-se a existência de dois *Analysers*, um para o processo de indexação e outro para o processo de pesquisa. Em ambos, utilizam o mesmo *tokenizer* e filtros. No caso que se está a analisar, o *tokenizer* utilizado obtêm todas as palavras que estejam separadas por espaços. Uma vez obtidas todas as palavras, são aplicados filtros para eliminar todas as terminações que possam existir nesse grupo de palavras existentes. Após o

[19] Consiste num componente que realiza pré-processamento de texto, durante a indexação e/ou pesquisa de documentos.

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <!-- in this example, we will only use synonyms at query time [2 lines]
    <!-- Case insensitive stop word removal. [3 lines]
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="stopwords.txt"
            enablePositionIncrements="true"
            />
    <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
        generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0" splitOnCaseChange="1"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SnowballPorterFilterFactory" language="English" protected="protwords.txt"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
    <filter class="solr.StopFilterFactory"
            ignoreCase="true"
            words="stopwords.txt"
            enablePositionIncrements="true"
            />
    <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
        generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0" splitOnCaseChange="1"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SnowballPorterFilterFactory" language="English" protected="protwords.txt"/>
  </analyzer>
</fieldType>
```

Figura 5 – Definição do tipo "*text*", com os respectivos *Analysers* para o tratamento de dados, consoante a operação que seja desencadeada.

procedimento do primeiro filtro, é aplicado um outro filtro para obter as palavras escritas em minúsculas e, por último, mais um filtro que permite evitar a redução de duas palavras que não se relacionam na sua palavra base (*SnowballPorterFilterFactory*).

2.3.1.3 Operações de Indexação

Tanto a indexação, como a procura são desencadeadas por meio do envio de pedidos *HTTP* para a aplicação *Web* do *SOLR*, existente num *servlet*. O *SOLR* ao captar o pedido, determina o *SolrRequestHandler*[20] apropriado e só depois é que processa esse pedido. As respostas são enviadas via *HTTP* pelo mesmo modo. Por omissão, as respostas são formatadas em *XML*[21], existindo a possibilidade de alterar para outro formato (ex: *JSON*[22]).

Na fase indexação, a aplicação obtém os dados textuais (e. g., excertos de textos, palavras-chave, etc…), enviando-os para o *SOLR* através de mensagens *XML*, via *HTTP Post*. Estas mensagens podem ser dos seguintes tipos:

- **add/update** – permite que se insira ou que se actualize um documento no *SOLR*. Essas alterações só estarão disponíveis nas pesquisas, após o comando *commit* ter sido realizado.

[20] Consiste num *plugin* do *SOLR*, que define a execução lógica de cada pedido.
[21] *eXtensible Markup Language*, desenvolvido por *World Wide Web Consortium* (*W3C*).
[22] *JavaScript Object Notation*, desenvolvido por *Douglas Crockford*.

- **commit** – indica ao *SOLR* que todas as alterações solicitadas, desde o último *commit* efectuado, sejam realizadas. A partir deste instante, os dados estão disponíveis para pesquisa.

- **optimize** – reestrutura os índices *Lucene* de modo a melhorar o desempenho das pesquisas. Se existirem actualizações frequentes, é necessário escalonar esta optimização para diminuir os tempos de utilização, pois consiste num processo bastante demorado. Por outro lado, um índice não precisa de ser optimizado de modo a funcionar correctamente.

- **remove** – pode ser especificado por meio de um/a identificador/chave ou através de uma *query*. Uma remoção por identificador elimina o documento com o identificador específico, ao passo que por meio de uma *query* são eliminados todos os documentos indicados por esta.

2.3.1.4 Operações de Pesquisa

Uma vez inseridos alguns documentos, estes ficam disponíveis para a pesquisa. Como foi referido anteriormente, o *SOLR* aceita mensagens de *HTTP Get* e de *HTTP Post*. De modo a processar as *queries* é atribuído um *SolrRequestHandler* apropriado. Em cada pesquisa, é possível definir o modo de execução da mesma, através dos seguintes atributos:

- **Boolean operator** – por omissão, o operador utilizado para combinar os termos da procura é o *OR*. Ao colocá-lo para *AND*, requer que todos os termos estejam no documento a comparar. O valor deste parâmetro encontra-se definido no ficheiro de *schema*, na secção corresponde às outras declarações.

- **Number of results** – especifica o número máximo de resultados a retornar.

- **Start** – indica o ponto de início no conjunto de resultados, tornando-se bastante prático para a paginação.

- **Highlight** – permite combinar os campos do documento.

Uma vez submetida a *query*, se tudo estiver correcto e havendo correspondência nos documentos, o *SOLR* retorna uma resposta em *XML* contendo, nos seus resultados, a informação relevante e alguns metadados relacionados com a *query* em causa.

Através do painel representado pela Figura 6, é possível verificar o conjunto opções existentes de modo a executar uma operação de pesquisa.

Observando com maior detalhe a figura em causa (Figura 6), a pesquisa vai muito para

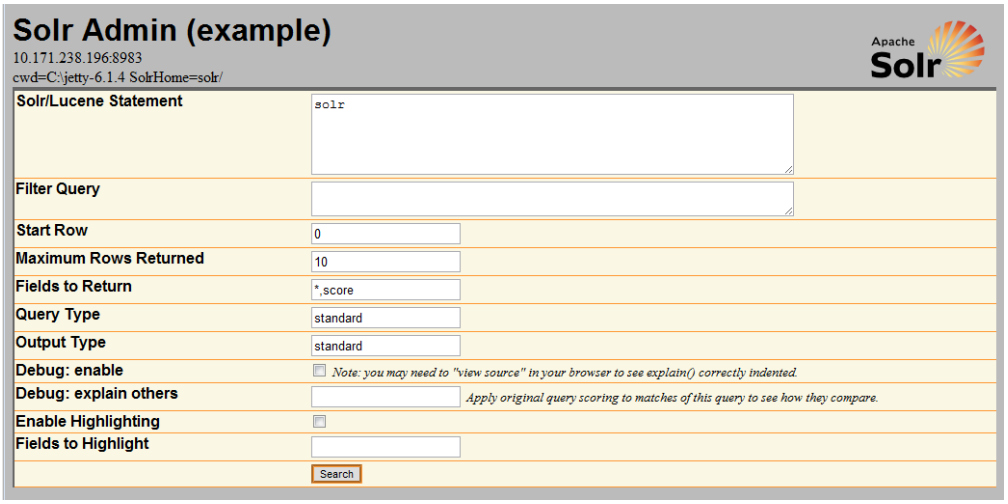

Figura 6 – Painel de pesquisa da interface de administração do *SOLR*.

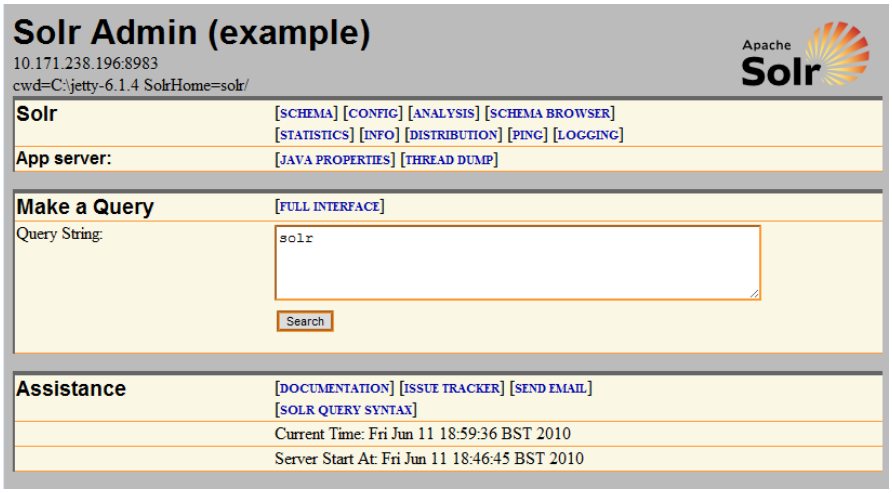

Figura 7 – Painel principal da interface de administração do *SOLR*.

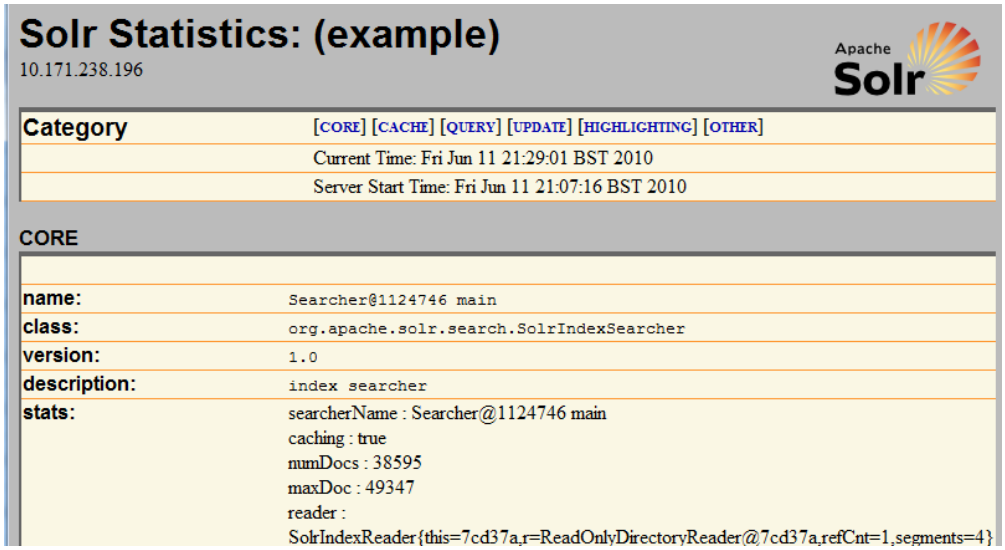

Figura 8 – Painel de visualização do estado do repositório.

além da definição do conteúdo textual a pesquisar. Existem um conjunto de parâmetros, disponíveis ao utilizador, aos quais possibilitam a obtenção de resultados que vão de encontro ao que é pretendido por estes (por exemplo a definição do inicio da visualização dos resultados e o número gerado, o tipo de representação dos mesmos – *JSON* ou *XML/XSLT*[23], etc.).

2.3.1.5 Interface de Administração

O *SOLR* na sua configuração dispõe de uma interface de administrador, permitindo assim a quem gere o repositório, verificar não só o estado do mesmo, como também permitir realizar pesquisas sobre o mesmo através do painel exemplificado na Figura 6.

Um programador ao recorrer a esta interface depara-se com um painel exemplificado na Figura 7. Através desta interface, o administrador do repositório pode observar o *schema* gerado para a aplicação em causa, como também verificar o estado do mesmo, através do número de documentos que este contém, o número de pesquisas efectuadas e outros tipos de propriedades que esta aplicação detém, como podemos verificar através da Figura 8.

2.3.2 Índices Invertidos

No processo de construção dos seus índices, este define uma estrutura de dados para indexação de informação que possibilita o mapeamento do conteúdo existente no repositório através palavras ou números. Através deste mapeamento, é possível identificar a localização do documento ou conjunto de documentos no repositório - *Índices Invertidos*. A utilização deste tipo de índices permite a execução de pesquisas textuais, em todo o repositório, de um modo rápido e bastante eficiente. No entanto, a construção deste tipo de estrutura acrescenta um custo significativo no momento de inserção de um documento, ou seja, ao introduzir um novo documento no repositório, toda a informação textual nele contida é processada e inserida no índice invertido. Desta forma, ao ser executada uma pesquisa, o processo de procura irá incidir sob o índice invertido e não propriamente no repositório, considerando-o muitas vezes como sendo índice do repositório, devido à sua importância em identificar os documentos que contêm essa/s palavra/s. O processo de pesquisa usa habitualmente a distância do co-seno descrita anteriormente.

Para o tipo de índices aqui referidos, existem duas grandes variantes: *inverted file* index, ao qual cada palavra contém uma listagem das referências para os documentos que as detêm; e *full inverted index*, que para além da listagem das referências para os documentos correspondentes, contem ainda a posição de cada palavra nesse documento. No caso em concreto do *SOLR* o tipo de índice invertido utilizado consiste no *inverted file index*, cuja

[23] *eXtensible Stylesheet Language for Transformation*, consiste numa linguagem que permite definir a apresentação dos documentos *XML* nos *Web Browsers* e outros aplicativos que a suportem.

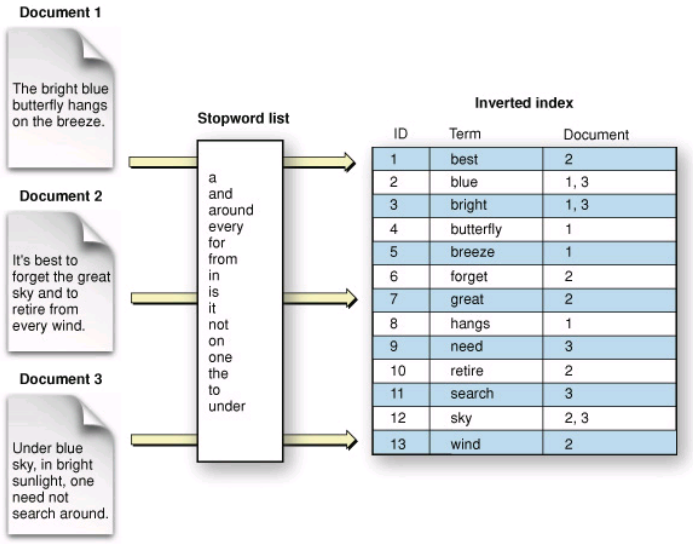

Figura 9 – Exemplo de índice invertido.

referência do documento corresponde ao identificador dos documentos, estando estes ordenados de acordo com a frequência dessa palavra no documento.

Embora este tipo de índices apresentem uma funcionalidade muito eficiente para o tipo de dados em causa (dados textuais), a sua transposição para conteúdo multimédia torna-se impossível de realizar. Como foi referido, cada posição do índice é constituída por uma listagem dos documentos que contêm um dado elemento. Ao transpor este tipo de índices para os dados multimédia, considerando uma dada característica associada a esses dados (e. g., uma dada cor) verifica-se que todos irão contê-la, ou seja, todos os dados irão conter uma percentagem referente a essa característica e por mais pequena que seja, esse conteúdo multimédia será associado à listagem dessa caracteristica. Deste modo, a utilização destes índices em dados multimédia não obtem eficiência verificada no *SOLR*, tornando-se necessário recorrer à utilização de estruturas de dados específicas para este tipos de dados, de modo a obter a eficiência pretendida.

2.4 Indexação de Multimédia

Devido há grande variância dos descritores dos conteúdos multimédia, a utilização de espaços particionados garante estabelecer um nível hierárquico na inserção destes objectos nas regiões por este definido, de modo a não produzir sobreposição. Sendo o espaço dividido em regiões não sobrepostas, permite que as pesquisas sejam rápidas e fáceis de realizar.

Em algumas aplicações não é possível mapear os objectos segundo vectores de *features* (descritores), por exemplo, a grande maioria dos websites com conteúdos multimédia (e.g.,

YouTube[24], *Flickr*[25], *Olhares*[26], etc.). Como vimos, é possível existir a noção de semelhança por meio do conceito de distância entre os objectos (ou seja, os objectos colocados num espaço métrico). Assim, os valores das distâncias podem ser utilizados automaticamente, durante a fase de avaliação das *queries*.

Várias estruturas de dados, para este tipo de espaço, têm vindo a surgir, sendo uma delas as *árvores de Burkhard-Keller* (proposta em *1973*). Este tipo de árvores utiliza as distâncias para poder obter o menor número de valores discretos. Um objecto aleatório é escolhido para ser raiz da árvore e os restantes objectos são distribuídos em pequenos conjuntos que irão corresponder a um ramo da árvore.

Outras variantes, consideram a utilização de árvores baseadas em *queries* fixas (e.g. a utilização de objectos como *pivots*[27]).

Existem vários tipos de estruturas que aplicam o conceito de *pivots* (ou seja, indexação baseadas em funções de distâncias contínuas, de modo a estabelecer a relação de proximidade entre objectos) na sua construção**.**

No esquema que se segue, pretende demonstrar as relações existentes entre as várias estruturas de dados definidas para o tipo de espaço que estamos a analisar.

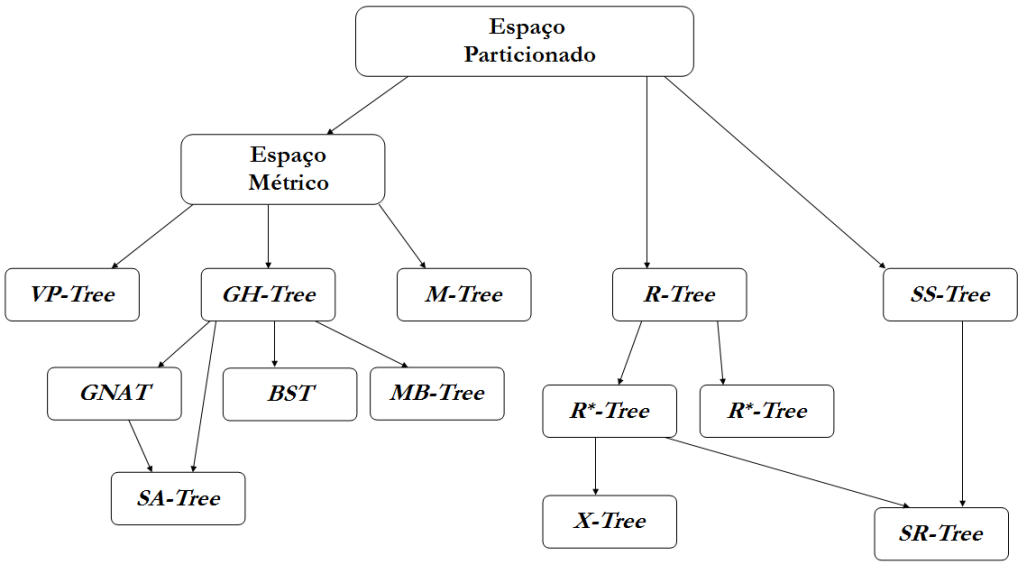

Figura 10 - Taxonomia do espaço particionado.

Analisemos, então, cada uma destas estruturas mais relevantes para o trabalho executado nesta dissertação (*VP-Tree*, *GH-Tree*, *GNAT*, *SA-Tree*, *M-Tree*). Por outro lado, foram consideradas para este estudo a *R-Tree* e *SS-Tree*, na medida em que apresentam uma elevada

[24] http://www.youtube.com/
[25] http://www.flickr.com/
[26] http://olhares.aeiou.pt/
[27] Constitui na selecção de objectos, a partir de um conjunto de objectos no espaço, onde cada um irá corresponder à raiz de uma sub-árvore que está a ser analisada.

relevância na representação de informação geográfica (no Anexo 1 encontra-se uma explicação detalhada das restantes estruturas identificadas na Figura 10).

2.4.1 Vantage Point Tree (*VP-Tree*)

A *VP-Tree* trata-se de uma estrutura de indexação, onde um objecto é escolhido aleatoriamente, do conjunto de objectos, determinando-se assim o *pivot* a ser utilizado. Uma vez obtido o *pivot*, determina-se a mediana de todas as distâncias em relação a este e os restantes objectos são divididos equitativamente em dois subconjuntos (denominaremos por S_1 e S_2). As expressões que se seguem pretendem demonstrar essa distribuição:

$$S_1 = \{o \in S \setminus \{p\} \mid d(p,o) < r\}$$

$$S_2 = \{o \in S \setminus \{p\} \mid d(p,o) \geq r\}$$

Os objectos em S_1 encontram-se dentro do raio de alcance que o *pivot* (p) consegue abranger, enquanto que os objectos em S_2 encontram-se fora desse raio. Ao aplicar esta regra recursivamente, obtém-se uma árvore binária, cujo *pivot* é colocado em cada nó com as respectivas sub-árvores esquerda e direita (encontrando-se os objectos dentro e fora do alcance desse *pivot*). Finalmente, as folhas têm a capacidade de guardarem mais do que um objecto, estando dependente do que é pretendido para o problema em causa.

2.4.1.1 Selecção do Pivot

Um método de obtenção de *pivots* é a escolha aleatória de um objecto. Contudo, pode por vezes acontecer, que não seja possível obter um objecto em melhor posição para exercer o papel de *pivot* (Yianilos 1993).

Assim, para solucionar este problema, (Yianilos 1993) propõe escolher aleatoriamente um conjunto de objectos e a partir dai, seleccionar o melhor objecto de todos e torná-lo *pivot*, ou seja, seleccionar aquele cuja distribuição de distâncias seja a melhor, em relação a todos os outros objectos escolhidos aleatoriamente.

No entanto, caso estejamos a trabalhar num espaço euclidiano e se os objectos estão dispostos uniformemente num *hipercubo*, a escolha dos *pivots* irá recair em pontos próximos dos vértices desse *hipercubo*. Ao seleccioná-los nessa área, implica que o raio de actuação dos *pivots* diminui no *hipercubo*, tornando a pesquisa mais eficiente (Yianilos 1993).

Uma outra possibilidade de construção das *VP-Trees* foi proposta por (Chávez, Navarro et al. 2001) que remete para a divisão dos objectos num ponto médio (obtido pelas distâncias do objecto mais próximo e do mais afastado), em relação ao *pivot*, obtendo-se assim subconjuntos de comprimentos iguais. Desta forma, permite que o particionamento seja melhorado, abrangendo mais objectos do que através da técnica da mediana. Contudo, esta

técnica não permite que sejam geradas árvores binárias perfeitamente equilibradas, como acontece na técnica da mediana. Para tal, a introdução de mecanismos de reorganização nas *VP-Trees*, pode facilitar esse processo (e. g., nas inserções e/ou remoções dinâmicas).

2.4.1.2 Procura

Os tipos de algoritmos pesquisa utilizados neste tipo de estrutura, correspondem ao *range query* e *search hierarchy*, uma vez que estes algoritmos são indiferentes do modo como os *pivots* e os seus alcances são obtidos.

Na pesquisa por *range query* é necessário observar viabilidade de percorrer uma sub-árvore de um dado nó para um dado *pivot*. Isto é conseguido, impondo limites inferiores a cada sub-árvore e, durante pesquisa, verificar se o alcance atribuído à *query* possibilita que esses limites sejam intersectados. Caso se verifique, então a pesquisa procederá nesse subconjunto; Caso contrário, essa sub-árvore é excluída. Podemos observar este processo, na figura abaixo:

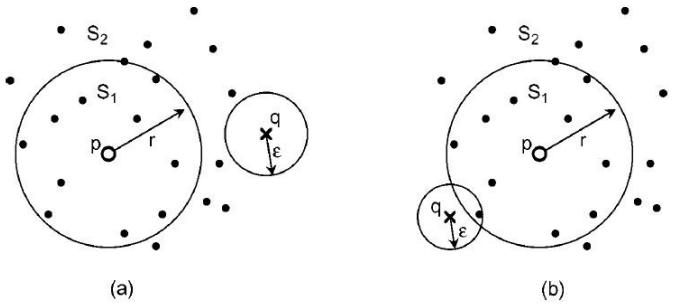

Figura 11 - Exemplos de pesquisas através da utilização do algoritmo *range query*: (a) Sem intersecção dos alcances (b) Com intersecção dos alcances (Hjaltason and Samet 2003).

Analisando a figura anterior, para um dado *raio* ε atribuído a q, verifica-se que não é necessário proceder a pesquisa por S_1 **(a)**; enquanto na situação em **(b)** já é necessário, uma vez que o alcance da *query* intersecta S_1.

Estudando o caso da pesquisa por *search hierarchy*, é necessário definir os tipos na *VP-Tree* (o *tipo 0* vai corresponder aos objectos, enquanto o *tipo 1* vai corresponder aos nós).

Observando a figura que se segue, poderemos compreender melhor o algoritmo em questão:

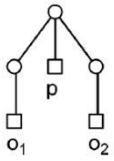

Figura 12 - Um exemplo de uma *VP-Tree* para três objectos, $\boldsymbol{p}$, $\boldsymbol{o_1}$ e $\boldsymbol{o_2}$ (Hjaltason and Samet 2003).

Como podemos observar, um nó produz um elemento do *tipo 0* (*pivot*) e outros dois elementos do *tipo 1* (ou seja, os dois filhos do nó).

Uma vez construída a estrutura, é necessário definir as funções de distância para cada um dos tipos que se encontram na árvore. Para os elementos do *tipo 0*, a sua função de distância vai corresponder ao valor da distância desse objecto à *query*, ao passo que se estivermos a analisar os elementos do *tipo 1*, a sua função de distância estará dependente do limite inferior do valor da distância de qualquer objecto da sub-árvore, em relação à *query*.

2.4.2 Generalized Hyperplane Tree (GH-Tree)

A *GH-Tree* (Uhlmann 1991) utiliza particionamento generalizado de *hiperplanos*, ou seja, em vez de utilizar um objecto para o particionamento (*VP-Trees*), são utilizados dois objectos como *pivots*, dividindo os restantes objectos pelo *pivot* que se encontre mais próximo:

$$S_1 = \{o \in S \setminus \{p_1, p_2\} \mid d(p_1, o) \leq d(p_2, o)\}$$

$$S_2 = \{o \in S \setminus \{p_1, p_2\} \mid d(p_2, o) < d(p_1, o)\}$$

Ou seja, os objectos em S_1 estão próximos de p_1 do que p_2 ou equidistante dos dois, ao passo que os objectos em S_2 estão mais próximos de p_2 do que de p_1. Esta regra é aplicada recursivamente, resultando numa árvore binária onde S_1 corresponde à sub-árvore esquerda e S_2 à sub-árvore direita (S_1 vai conter todos os objectos que respeitem a condição $d(p_1, o) - d(p_2, o) \leq 0$).

O termo "*particionamento de hiperplano generalizado*", deriva do facto de os objectos serem pontos no *espaço euclidiano n-dimensional*, resultando num particionamento equivalente a um *hiperplano (n-1)-dimensional*, constituído pelo conjunto de todos os pontos o que satisfazem a condição $d(p_1, o) = d(p_2, o)$.

Contudo, a sua representação directa num espaço métrico arbitrário não é possível de realizar, visto que a única informação disponível são as distâncias entre objectos.

Para aumentar a informação disponível em cada nó da árvore, é necessário incluir em cada *pivot*, a distância máxima a um objecto na sub-árvore (resultando numa *BST* ou *bisector tree*, que irá ser estudada mais à frente). Assim, a expressão para encontrar essa distância, é definida como sendo o raio $r_i = \max_{o \in S_i}\{d(p_i, o)\}$, idêntico ao que acontece nas *VP-Trees*.

Na Figura 13, encontra-se uma representação desta estrutura.

2.4.2.1 Procura

Neste tipo de estrutura, o algoritmo mais utilizado é o *search hierarchy*. Para tal, é constituída por objectos e nós, onde cada nó contem dois *pivots*, p_1 e p_2. O limite inferior para a função de distância (d_1) é definido por meio dos dois nós filhos do nó (S_1 e S_2),

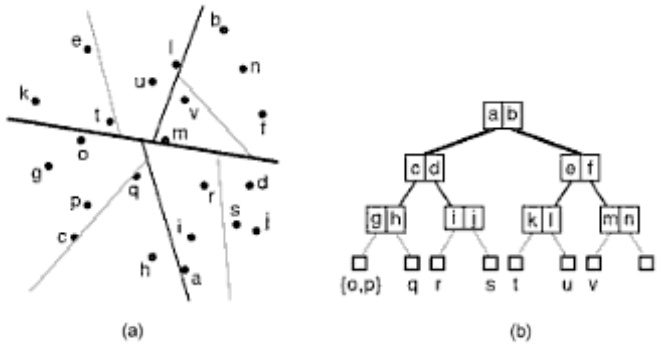

Figura 13 - Exemplo de uma representação de uma *GH-Tree*, para um dado conjunto de pontos: (a) Representação no espaço eucladiano; (b) Representação em árvore(Hjaltason and Samet 2003).

obtendo-se a seguinte expressão:

$$d_1(q,e') = \begin{cases} max\left\{\frac{d(q,p_1)-d(q,p_2)}{2}, d(q,p_1)-r_1, 0\right\}, se\ e' \text{é}\ filho\ de\ e\ de\ S_1 \\ max\left\{\frac{d(q,p_2)-d(q,p_1)}{2}, d(q,p_2)-r_2, 0\right\}, se\ e' \text{é}\ filho\ de\ e\ de\ S_2 \end{cases}$$

Contudo, não é possível determinar o limite superior da função de distância através deste algoritmo. Um modo de ultrapassar esse contratempo, consite em tirar partido do raio r_i para obtermos esse limite, resultando na geração da seguinte expressão:

$$\hat{d}_1(q,e') = \begin{cases} d(q,p_1)+r_1, se\ e' \text{ é}\ filho\ de\ e, em\ S_1 \\ d(q,p_2)+r_2, se\ e' \text{ é}\ filho\ de\ e, em\ S_2 \end{cases}$$

Utilizando os princípios subjacentes nesta estrutura, foram definidas outras estruturas como são os casos das que se seguem.

2.4.3 GNAT

A *GNAT* (*Geometric Near-neighbor Access Tree*) é uma generalização da *GH-Tree*, idealizada por (Brin 1995)[28].

Esta estrutura possibilita escolher mais do que dois *pivots* para o particionamento, ou seja, dado um conjunto de *m pivots*, os objectos serão divididos por *m* subconjuntos, dependendo da sua proximidade para com um dado *pivot*. Se existir um empate o objecto é indexado no *pivot* que tiver menor índice. Ao aplicarmos este processo recursivamente, irá obter-se uma *árvore m-ária*. Por outro lado, o valor *m* é passado por parâmetro, como também a possibilidade de definir quantos *pivots* estarão associados a um dado nó, sendo o critério, neste caso, a cardinalidade de cada subconjunto. Os *pivots* são escolhidos aleatoriamente.

[28] Sergey Brin, um dos co-fundadores da Google.

Uma vez encontrados, é escolhido o primeiro *pivot* aleatoriamente a partir desse subconjunto. De seguida, escolhe-se um outro *pivot* que esteja mais afastado do primeiro. O *pivot* seguinte, é determinado pelo seu afastamento relativamente aos anteriores e assim sucessivamente.

Os nós da *GNAT* para além de guardarem os *pivots* e os apontadores para os seus filhos, guardam também as distâncias entre os *pivots* e os objectos em cada sub-árvore, melhorando o processo de pesquisa.

Num espaço euclidiano é possível indicar as *células de Voronoi*[29] e determinar o limite inferior da distância que vai da *query* para os diversos *pontos/objectos* que estão contidos na célula. No entanto, em espaços métricos arbitrários, determinar esse limite torna-se pouco fiável, uma vez que não é possível representar genericamente essas células definidas pelo *pivot* em causa. Uma solução seria utilizar as funções de distância definidas para as *GH-Trees*.

2.4.3.1 Procura

A procura neste tipo de estrutura, é realizada por meio de *range query*, ou seja, para uma dada *query*, a pesquisa é feita por meio de *depth-first*. Ao processar um nó, as distâncias entre a *query* e os *pivots* são analisadas uma a uma, eliminando as possíveis sub-árvores. Os filhos de cada nó só são processados a partir do ponto em que as distâncias de todos os *pivots* tenham sido determinadas. O processo inicia-se com um conjunto, contendo todos os *pivots* do nó. A cada passo, retira-se um *pivot*, cuja distância à *query* ainda não foi determinada e calcula-se. De seguida, verifica se esse *pivot* fará parte do resultado da *query* ($d(q, p_i) \leq \varepsilon$).

O algoritmo de *nearest neighbor* pode também ser aplicado a esta estrutura por meio do *MaxDist(NearList)*, ou seja, através da determinação a distância do vizinho mais afastado do *pivot*.

2.4.4 SA-Tree

Embora a *GH-Tree* e a *GNAT*, sejam baseadas no particionamento de *células de Voronoi*, as *SA-Trees* (Navarro 1999; Navarro 2002) utilizam um particionamento hierárquico, onde em cada nível, o espaço particionado é colocado em duas ou mais regiões das *células de Voronoi*, tendendo a aproximar-se de um *grafo de Delaunay*.

A construção da *SA-Tree* é realizada, escolhendo um objecto aleatório para ser a raiz da árvore, denominado por a. De seguida, é identificado um pequeno conjunto, que irá conter todos os objectos que estão mais próximos da raiz. Esse pequeno conjunto, é designado pelo conjunto vizinho de a e todos os seus objectos são vizinhos da raiz. Assim, para o conjunto vizinho de a, todos objectos estão mais próximos de a do que dos restantes objectos desse conjunto (do mesmo modo, todos os restantes objectos estão mais próximos de um dos

[29] Regiões do espaço, que constituem no *Diagrama de Voronoi*, onde definem o raio de acção dos objectos que nelas estão contidas.

objectos do conjunto vizinho do que da própria raiz). Os objectos que restaram serão associados a um dos filhos mais próximos de a e a sub-árvore é construída recursivamente, pelo mesmo método. A distância da raiz de cada sub-árvore (S_b) até ao objecto mais distante de S_b, pode ser guardado em b.

A Figura 14 apresenta um pequeno exemplo da *SA-Tree* num espaço bidimensional com pontos $a - w$ antes de ser construída (a) e após a construção (b), sendo o ponto a o escolhido para ser a raiz.

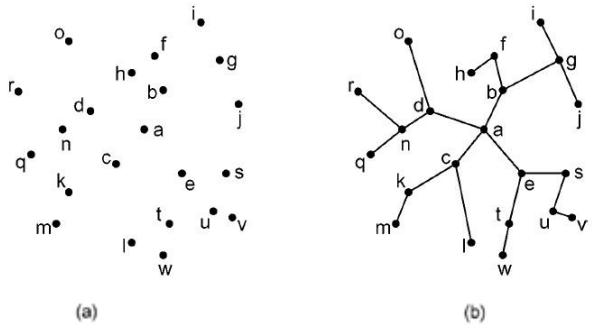

Figura 14 - Exemplo para uma possível representação de uma *SA-Tree*, para um dado conjunto de pontos: (a) Antes da construção; (b) Após a construção (Hjaltason and Samet 2003).

O facto de se determinar o conjunto de vizinhos de um dado nó, implica que este processo seja muito dispendioso, traduzindo-se num problema *NP-Completo*.

(Navarro 2002), sugeriu uma heurística para determinar esses conjuntos, considerando todos os objectos e determinar os mais próximos dele. A construção final, por meio desta heurística, pode ser vista na Figura 14.

No entanto, podem ocorrer situações onde não é possível determinar o conjunto mínimo de vizinhos, como poderemos verificar na figura que se segue:

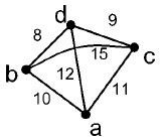

Figura 15 - (Hjaltason and Samet 2003).

Ao analisarmos a Figura 15, observamos que o conjunto vizinho mínimo de a é d, contudo a heurística indica que seria b e c. Apesar de a heurística não encontrar sempre o conjunto de vizinhos mínimo, ela é determinista, ou seja, para o mesmo valor de distâncias o mesmo conjunto é sempre obtido. Isto sucede, uma vez que a representação da estrutura é definida a partir do momento que a raiz da árvore é determinada. Desta forma, para diferentes raízes iremos ter diferentes estruturas de árvore.

2.4.4.1 Procura

Utilizando a *SA-Tree*, será fácil de executar o *exact match queries*, porque utiliza o mesmo princípio de um *grafo de Delaunary*. O procedimento consiste em escolher um ponto arbitrário no total de pontos no espaço. Posteriormente, é encaminhado para um ponto da vizinhança que esteja mais próximo da *query*, ou seja, até onde seja possível prosseguir. Uma vez encontrado, todos os pontos que estão agregados no seu grupo de vizinhança, encontram-se mais afastados da *query* do que em relação a esse ponto, uma vez que o objecto está mais próximo da *query*.

As pesquisas de *nearest neighbor* e *range search*, podem ser efectuadas na *SA-Tree*. Se o objecto escolhido for a raiz da árvore a, então é escolhido um objecto.

Ao executarmos o algoritmo de *range search* utiliza o limite inferior das distâncias, determinadas anteriormente, para melhorar a pesquisa. Deste modo, é possível verificar se a pesquisa é idêntica ao *depth-first transversal*, iniciando-se o processo pela raiz. Caso o nó escolhido não seja a raiz da árvore, então determina-se um objecto pertencente a um conjunto de vizinhos de um objecto vizinho da raiz, de modo que a sua distância à *query* seja o mínimo possível. Se o nó for a raiz, então o objecto fará parte do conjunto da raiz. Posteriormente, é executada a pesquisa de *depth-first transversal*, visitando cada filho do conjunto de vizinhos mais próximos, caso a diferença média das distâncias de cada filho à *query* com a distância do objecto à *query* não ultrapasse o alcance da mesma.

A *SA-Tree* é uma estrutura estática, pois temos de saber à partida todos os dados a serem introduzidos, na fase de construção. (Navarro and Reyes 2002), apresentaram uma versão dinâmica, que suporta inserções e remoções. Para manterem o desempenho das pesquisas, foi necessário tornar menos restritivo os conjuntos de vizinhos de cada nó, por outras palavras, dado um objecto a introduzir na árvore, num dado instante t, as propriedades definidas anteriormente dos antecessores e dos irmãos dos antecessores a esse objecto, só permanecem caso tenham sido inseridos antes desse instante t. Isto possibilita, que ao avaliarmos as distâncias no limite inferior dos objectos da sub-árvore com raiz num nó b, só poderemos utilizar a informação dos antecessores e os irmãos dos antecessores do nó b, aos quais estes foram inseridos antes de b.

Da análise realizada até ao momento a estas estruturas, verifica-se que o processo de indexação não considera as situações de informação dinâmica, ou seja, para se poder realizar a indexação, toda a informação tem de estar presente, pois, caso contrário, irá provocar ineficiência nas pesquisas, havendo a necessidade de reconstruir a estrutura, mas agora com a nova informação adicionada. Assim, a estrutura que iremos estudar de seguida, tenta ultrapassar este problema.

2.4.5 M-Tree

A *M-Tree* (Ciaccia, Patella et al. 1997; Ciaccia and Patella 2002), é um método de indexação baseado na distância, onde pretende resolver o problema das estruturas anteriores, relativamente a grandes quantidades de informação dinâmica. Para tal, o seu objectivo é combinar o dinamismo, balanceando com a indexação utilizada nas *B-Trees* com a capacidade de índices estáticos, baseados nas distâncias.

Nas *M-Trees*, os objectos a indexar são referenciados nas folhas, enquanto nos nós, são guardados apontadores para os nós no nível imediatamente abaixo, como também, a informação sumária sobre os objectos da sub-árvore que está a apontar.

Os *pivots* na *M-Tree* apresentam a mesma funcionalidade que os *pivots* na *GNAT*, mas ao contrário das *GNAT's*, todos os objectos são guardados nas folhas da *M-Tree*, podendo ser referenciados múltiplas vezes ao longo da árvore.

Uma representação desta estrutura encontra-se demonstrada na Figura 16.

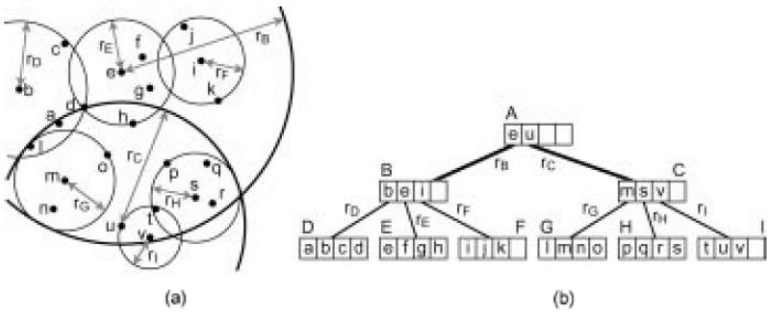

Figura 16 - Exemplo de uma representação de uma *M-Tree*, para um dado conjunto de pontos: (a) Representação da estrutura no espaço euclidiano; (b) Representação em árvore (Hjaltason and Samet 2003).

Cada nó tem como entradas (p, r, D, T), sendo p o *pivot*, r a *covering ball*, D o valor da distância entre p e o objecto pai desse nó e T uma referência para o nó filho. De modo similar, as entradas nas folhas são constituídas por (o, D), onde o corresponde ao objecto em causa e D à distância entre o e o objecto pai desse nó. Como é de esperar, a raiz não tem qualquer pai, logo $D = \infty$.

Ao ser uma estrutura dinâmica, pode ser construída à medida que novos dados surgem. Na inserção, o novo objecto é encaminhado para a folha respectiva. Isto é conseguido por meio de uma heurística em que, à medida que percorre a árvore, é escolhido o filho que melhor combina com o objecto que está a ser inserido. Por exemplo, a heurística pode procurar o *pivot* cuja a sua *covering ball* contém o objecto ou, então, procura o *pivot* mais próximo do objecto, caso exista mais do que um *pivot* com que emparelhar.

Durante a inserção, pode ocorrer uma situação de *overflow*[30] do nó. Nesse caso, o conteúdo do nó é dividido e um novo *pivot* é calculado levando que este possa vir a ser propagado até à raiz da árvore. Desta forma, concluímos que a *M-Tree* cresce de baixo para cima. (Ciaccia, Patella et al. 1997), considera um número de heurísticas para escolher o nó filho para introduzir e dividir pelos nós de *overflow*. Estratégias de *bulk-loading*[31] foram também desenvolvidas para este tipo de árvores, mas apenas consideram o conjunto de dados existente.

O particionamento da *M-Tree* a partir da raiz, para um conjunto de objectos, pode ser exemplificado por meio da Figura 17.

Observando a Figura 17, esta traduz um possível particionamento de alto nível de um conjunto de objectos (oriundos de um espaço bidimensional) numa *M-Tree*. Os objectos que caiem em mais do que uma "*ball*", como é o caso do o, podem ser inseridos numa das subárvores correspondentes.

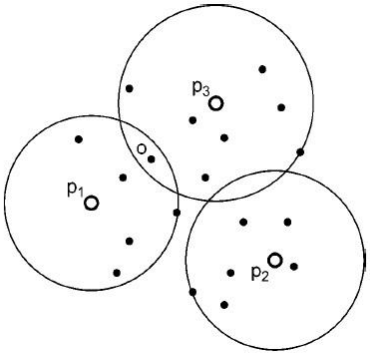

Figura 17 - Particionamento de uma *M-Tree*, para um conjunto de objectos (Hjaltason and Samet 2003).

2.4.5.1 Slim-Tree

(C. Traina, Traina et al. 2002), introduziu a *Slim-tree*, uma variante da *M-Tree* com algoritmos de inserção e de divisão de nós mais rápida. Esta estrutura melhora o espaço utilizado, conseguido por meio de um algoritmo designado por *Slim-down*. Este algoritmo tenta reduzir a sobreposição das regiões dos nós, movendo as entradas entre nós irmãos de modo iterativo. Considerando as folhas, ao aplicarmos o algoritmo, um objecto o é candidato a mover-se, se esse objecto estiver muito afastado do nó pai e a região da folha irmã também envolver o objecto. Ao encontra-se esse candidato, então este é movido para a folha irmã e a área de cobertura da folha é reduzida, dependendo da distância que existe entre o *pivot* e o próximo objecto mais afastado que a contém. Através deste método,

[30] O conteúdo de uma folha ou nó estão totalmente preenchidos, não havendo a possibilidade de inserir uma nova entrada nessa folha ou nó.
[31] Recurso à utilização de um ficheiro para importar os dados para o índice.

verificou-se reduções significativas de acesso a disco, comparativamente com as *M-Trees* tradicionais (C. Traina, Traina et al. 2002).

2.4.5.2 Procura

A pesquisa por meio do algoritmo de *range queries*, pode ser efectuada na *M-Tree*. Para tal, é utilizado o método de pesquisa *depth-first transversal*, que se inicia na raiz da àrvore.

Considerando os nós com entradas (p, r, D, T), existem duas possibilidades para a pesquisa:

5. Se $|d(q,p') - D| - r > \varepsilon$, então a sub-árvore que é apontada por T não necessita de ser percorrida, excluindo essa entrada. O critério que está subjacente, deriva do facto de se tratar do limite inferior da distância de qualquer objecto existente na sub-árvore referenciada em T. Se o valor for superior a ε, nenhum objecto na sub-árvore estará no alcance da *query*.

6. Se $|d(q,p') - D| - r \leq \varepsilon$, então não será possível evitar o processamento de $d(q,p)$. Contudo, continua a ser possível evitar a visita um nó apontado por T, caso o limite inferior da distância da *query*, a qualquer objecto em T, for maior que ε.

No caso das folhas, o processo é similar. Para cada entrada (o, D) de um nó com um pivot pai p', verifica-se se $|d(q,p') - D| \leq \varepsilon$ e, de seguida, para as entradas que satisfazem essa condição, determina-se se $d(q,o) \leq \varepsilon$.

Ao analisarmos o algoritmo, podemos concluir que as distâncias em relação aos pais permitem eliminar percursos, evitando a necessidade de calcular as distâncias aos objectos.

Um outro algoritmo de pesquisa proposto para este tipo de estrutura, consiste no *k-nearest neighbor search*, (Ciaccia, Patella et al. 1997). Este algoritmo pretende utilizar a distância do candidato mais afastado do *kth nearest neighbor* em detrimento de ε, para definir as condições de exclusão. Contudo, estas condições não podem ser aplicadas ao algoritmo anterior, uma vez que o número de objectos resultantes não é conhecido à partida. Para ultrapassar este problema, é necessário introduzir dois novos tipos elementos (objectos aproximados e nós aproximados) no algoritmo *search hierarchy*.

Estes elementos fornecem um modo simples de ordenar a subsequência, processando os elementos das folhas e dos nós, sem que seja necessário determinar as distâncias desses elementos em relação à *query*. (Hjaltason and Samet 2000; Hjaltason and Samet 2003), indicaram que os resultados obtidos são melhores do que o algoritmo de *k-nearest neighbor*.

Deste modo, no *search hierarchy*, são definidos 4 tipos: *tipo 0* para os objectos, *tipo 1* para os objectos aproximados, *tipo 2* para os nós e *tipo 3* para os nós aproximados. Os elementos do *tipo 1* e *3* são o resultado do processamento nas folhas e nos nós, respectivamente. Um

elemento do *tipo 0* é gerado a partir do processamento de um elemento do *tipo 1*, do mesmo modo para os do *tipo 2* que derivam dos de *tipo 3*. Assim as funções de distância de limite inferior, para os elementos do *tipo 1*, *2* e *3*, são definidas da seguinte forma:

$$d_1(q, e_1) = max\{|d(q, p') - D|, 0\},$$

$$d_2(q, e_2) = max\{d(q, p) - r, 0\},$$

$$d_3(q, e_3) = max\{|d(q, p') - D| - r, 0\},$$

onde p' é o objecto pai e D a distância correspondente para uma entrada num nó, onde e_1 e e_3 são gerados; p e r são o *pivot* e o *alcance* para o nó correspondente a e_2 e e_3.

Para o limite superior, temos as seguintes funções para os elementos do *tipo 1*, *2* e *3*:

$$\hat{d}_1(q, e_1) = d(q, p') + D,$$

$$\hat{d}_2(q, e_2) = d(q, p) + r,$$

$$\hat{d}_3(q, e_3) = d(q, p') + D + r.$$

Para facilitar a determinação das distâncias em relação aos descendentes, é necessário que os elementos do *tipo 1* incluem a identidade do objecto correspondente; que o elemento do *tipo 2* inclua um apontador para o nó correspondente e a distância $d(q, p')$, onde p' é o objecto pai do nó e que o elemento do *tipo 3* inclua p, r e T.

Assim, garante-se que ao executar d_1 e d_3, a informação sobre as distâncias já se encontra disponível, uma vez que *D* foi calculado durante a construção da *M-Tree* e $d(q, p')$ durante o processamento da *query* e guardada em e_2, ou seja, no nó onde e_1 e e_3 são gerados. Ao aplicar o *range query* ao *search hierarchy*, os elementos do *tipo 1* e *3*, cujas as distâncias excedem o alcance da *query*, podem ser excluídos da pesquisa sem que seja necessário calcular as respectivas distâncias. Desta forma, podemos dizer que o algoritmo de *range query* e o de *search hierarchy* são equivalentes, pois produzem o mesmo efeito.

2.4.6 R-Tree

A *R-Tree* (Guttman 1984) trata-se de uma estrutura de indexação que utiliza delimitação das suas regiões por meio de rectângulos mínimos (*MBR – Minimum Bounding Rectangles*[32]). O *MBR* não é mais do que um intervalo multidimensional de dados no espaço, onde congrega um conjunto de pontos cujos valores são próximos. Para tal, em cada área da *superfície (d-1)-*

[32] Consiste numa expressão que define o alargamento máximo de um objecto bidimensional (e.g. ponto, linha, polígono, etc…) existente num sistema de coordenadas (x, y), ou seja, é definido segundo $\max(x)$, $\min(y)$ e $\max(y)$. Trata-se de um caso especial do *minimum bounding box - MBB*, utilizado em espaços com dimensionalidade maior.

dimensional, irá conter no mínimo um *datapoint* (ou seja, região de página[33]). Contudo, o particionamento do espaço não é completo nem disjunto, ou seja, partes do espaço de dados podem não estar totalmente contidas nas regiões de páginas. Por outro lado, é possível efectuar a sobreposição dessas regiões, tendo como desvantagem prejudicar o processo de pesquisa.

A região de um *MBR* pode ser descrita por meio da definição dos seus limites superiores e inferiores. Deste modo, é possível determinar as funções MINDIST[34], MINMAXDIST[35] e MAXDIST[36] usando de forma eficiente as métricas L_p.

Na sua concepção, as *R-Trees* foram projectadas para repositórios espaciais (e. g. para manipular objectos *2D*, como são o caso dos polígonos e dados geográficos). No índice, estes objectos são representados pela *MBR* correspondente.

Em traços gerais, as *R-Trees* indexam *datapoints*, mas na zona que correspondente à directoria[37]. As regiões de página são tratadas como objectos que são espacialmente estendidos, a partir dos nós dos seus pais. Assim, a directoria de páginas não pode ser dividida enquanto não ficar sobreposta com as novas páginas.

Existem duas heurísticas, que garantem o controlo das operações de inserção (ex: a escolha da melhor página para inserir um ponto e o controlo da sobreposição de páginas).

2.4.6.1 Procura

Na pesquisa da melhor página, um de três factores pode acontecer:

- O ponto está contido numa única região, logo essa página é utilizada.
- O ponto está contido em várias regiões, logo a região escolhida é a mais pequena.
- Nenhuma região contém o ponto. Neste caso será escolhida a região que realizará o menor alargamento. Se existirem várias regiões que contêm esse mínimo a região que for mais pequena será a escolhida.

2.4.6.2 Inserção

O algoritmo de inserção inicia-se na raiz e em cada passo é escolhido um filho, através dos critérios descritos anteriormente.

---

[33] Consiste região que delimita o espaço para congragar o conjunto de dados que estão mais próximos.
[34] Distância mínima entre um ponto da *query* e um qualquer ponto existente no *MBR*.
[35] Retorna o valor mínimo de todas as distâncias que partem de um ponto da *query* ao ponto mais afastado, da face mais próxima do *MBR*.
[36] Distância máxima entre um ponto da *query* e um qualquer ponto existente no *MBR*.
[37] Uma directoria consiste num *MBR* que contém todos os rectângulos existentes nas regiões de páginas, na sub-árvore correspondente.

Devido à existência de sobreposição de páginas, a resolução deste problema está na divisão de páginas. Para tal, existem *4 algoritmos* para identificar as dimensões correctas para a divisão e o *hiperplano* que irá sofrer a divisão. Estes algoritmos diferenciam-se em termos da sua complexidade temporal e na capacidade não uniforme das páginas. Assim, os algoritmos são os seguintes: Algoritmo exponencial; Algoritmo quadrático; Algoritmo linear e Algoritmo de *Greene* [1989].

Na inserção, o tempo necessário para encontrar a página ideal para o objecto a introduzir, anda na ordem $O(\log n)$, uma vez que é escolhido um só caminho. Caso a procura considerar vários caminhos, então a complexidade temporal aumenta para $O(n \log n)$.

Contudo, a inserção dos dados pode levar a introdução de dados em páginas erradas, nomeadamente em situações de empate. Podemos observar essa situação na Figura 18.

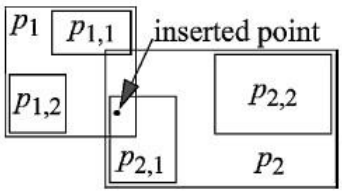

Figura 18 - Exemplo de uma possível inserção de um objecto numa página errada (Böhm, Berchtold et al. 2001).

Como podemos verificar, o modo como o algoritmo está construído implica que o ponto a introduzir fará parte da página p_1, uma vez que no primeiro nível só são consideradas as páginas p_1 e p_2 e nesse processo p_1 é mais pequena que p_2. Uma vez determinada, a região que será escolhida será a $p_{1,2}$, estendendo-a de forma a poder englobar o objecto. Contudo através de uma análise directa sobre a figura verificamos que a região a ser escolhida seria a $p_{2,1}$, pois esta engloba a zona na qual o ponto se encontra. No entanto, isto não se verifica, uma vez que a região faz parte da página p_2 e p_2 é maior que p_1.

Quando acontece uma sobreposição neste tipo de estruturas, pode, por vezes, impossibilitar a divisão de páginas em níveis mais altos, uma vez que todas as páginas do filho são independentes no modo como crescem nas regiões de páginas. Para tal, uma sobreposição que possibilita uma divisão livre necessita de utilizar a dimensão, de modo a identificar a zona onde não existe qualquer tipo de sobreposição. Contudo, segundo (Berchtold, Keim et al. 1996), à medida que a dimensão aumenta, torna-se cada vez menos possível encontrar tal localização. Isto deve-se à projecção da página de cada filho para uma dada dimensão arbitrária não ser tão pequena como a projecção correspondente da página do filho.

2.4.7 SS-Tree

A *SS-Tree* consiste numa estrutura que utiliza esferas[38] para caracterizar as regiões de página (White and Jain 1996). Para manter a eficiência da estrutura, as esferas são obtidas a partir de um *ponto centroid*[39] utilizado para definir o centro da esfera, e um *raio mínimo*, que é determinado, de modo a todos os objectos estejam incluídos na esfera. Assim, a descrição da região é definida por meio de um *ponto centroid* e por um *raio*.

Com esta informação, torna-se possível determinar as funções MINDIST e MAXDIST, mas não a MINMAXDIST (estas funções são idênticas às que foram definidas na *R-Tree*. A única diferença encontra-se na utilização das *MBS*'s no lugar dos *MBR*'s).

A inserção é realizada de modo descendente. É escolhido um nó filho cujo *centroid* é o mais próximo do ponto, independentemente do seu volume ou do aumento no caso de sobreposição. Posteriormente, é determinado um novo *centroid* e um novo *raio*. Caso ocorra um *overflow*, é executada a operação *forced reinsert*. Nesta situação, todos os objectos mais afastados do *centroid* são excluídos do nó (cerca de 30% dos objectos), actualizando as descrições nós. Após esta operação, os objectos são novamente inseridos no índice.

O modo de como se determina a divisão é baseado num critério de variância, ou seja, é determinado inicialmente o eixo de divisão à medida que compreende a maior variância. De seguida, o plano de divisão é determinado pela identificação da posição de todas as possíveis divisões que garantam a maior taxa de utilização de espaço. A soma de todas estas variâncias, em cada lado do plano de divisão, é minimizada.

O maior problema da utilização de esferas, é que estas não respeitam um modo de divisão livre de uma simples sobreposição, como podemos verificar na Figura 19.

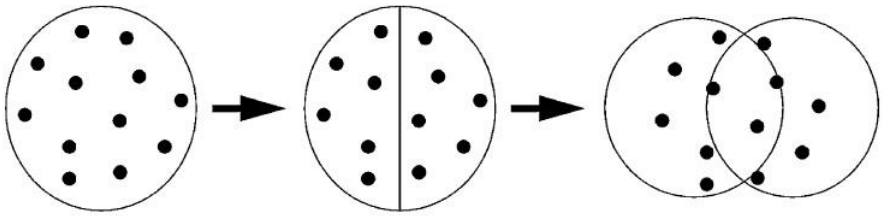

Figura 19 - A divisão livre, em caso de sobreposição, não é possível (Böhm, Berchtold et al. 2001).

2.4.8 Sumário

Estando analisadas todas as estruturas para indexação em espaços particionados, seria fundamental realizar um apanhado das principais características destas estruturas, como

[38] Trata-se de *MBS – Minimum Bounding Spheres*, ou seja, são esferas com um raio mínimo que permite conter um conjunto de objectos num espaço n-dimensional.
[39] Determinado a partir do valor médio do vector do descritor de cada nó filho.

também, considerar um conjunto de parâmetros para análise comparativa entre as mesmas.

A tabela que se segue (Tabela 1), tenta recriar essa análise.

Analizando a tabela em questão (Tabela 1) verificam-se os possíveis problemas que advêm da implementação destas estruturas na aplicação que irei desenvolver (nomeadamente, no que se referem às estruturas acentes sobre o espaço métrico). Como é possível observar, estas estruturas acentam, maioritariamente, na determinação dos seus *pivots*, recorrendo a um processo selecção aleatório de imagens. O problema que se coloca através da utilização deste processo, consiste na possibilidade de selecionar *pivots* que não correspondem na melhor solução para o conjunto de imagens no repositório (desta forma, poderão advir problemas no processo de pesquisa, i.e., obter tempos de pesquisa altos).

Uma outra problemática encontra-se subjacente na construção da *SA-Tree*. Na construção desta estrutura, é necessário definir os conjuntos vizinhos de cada imagem. O problema que se coloca é na obtenção desses conjuntos e evitar, ao mesmo tempo, que sejam produzidos falsos caminhos no processo de construção do *grafo de Delaunay* (o que poderá vir a trazer custos significativos na construção do índice, nomeadamente em termos do tempo de construção do índice). Apesar desta desvantagem, estando o *grafo de Delaunay* construído, o processo de pesquisa tornar-se-á mais rápido, na medida em que os caminhos para cada imagem já se encontram definidos.

Posto isto, embora apresentem alguns aspectos que contrariem a sua implementação, estas estruturas constituem numa boa base para podermos avaliar o seu comportamento, em termos práticos, e identificar qual a estrutura melhor adaptada para o tipo de dados utilizados.

Tabela 1 – Estruturas de indexação em espaço particionado com as respectivas propriedades e a comparação qualitativa entre estas estruturas.

Nome	Região	Inserção	Divisão	Reinserção	Problemas	Pesquisas	Utilização do espaço	*Fanout*/Tamanho das entradas
VP-Tree	*Pivots*.	Distância ao *pivot*.	Mediana das distâncias.	Não	Escolha aleatória do pivot	Range Search Hierarchy	Boa	Aumenta com o número de níveis.
GH-Tree	Hiper-planos.	Distância aos *pivots*.	Não	Não	Restrito ao espaço Euclidiano	Search Hierarchy	Boa	Grande
GNAT	Subconjuntos, cada um com um *pivot*.	*Pivot* mais próximo.	Não	Não	Representação das células de *Voronoi*	Range NN[ii]	Boa	Depende do número de filhos.
M-Tree	*Covering balls*.	*Pivot* mais próximo.	*Overflow* – um novo *pivot*.	*Slim-down*	Os problemas das *B-Trees*. O número de nós tem de ser conhecido.	Range Depth-first transversal kth NN[ii] Search Hierarchy	Boa	Os nós são definidos por (***p***, ***r***, ***D***, ***T***) As folhas são definidas por (***o***, ***D***)
SA-Tree	*Células de Voronoi*.	Escolha dos objectos mais próximos a um objecto.	Não	Não	Heurística pode ser ineficiente. Conhecimento do número de objectos a introduzir.	NN[ii] Exact match Range Depth-first transversal	Boa	Conhecimento dos nós que são próximos.
R-Tree	*MBR*[i]	Aumento do volume	Múltiplos algoritmos	Não	Algoritmo de divisão que deteriora as directorias	NN[ii] Region Range	Pouca	Fraca, dependente da dimensão
SS-Tree	Esferas	Proximidade ao *centroid*	Variância	Sim	Elevada sobreposição na directoria	NN[ii]	Média	Muito boa, independente da dimensão

[i] *Minimum Bounding Rectangle*
[ii] *Nearest-Neighbor*

3

Aplicação de Navegação e Pesquisa de Multimédia

Tendo analisado as várias estruturas específicas para a indexação de conteúdos multimédia e os vários sistemas que tiram partido dos descritores de baixo nível (e.g. *QBIC* e *Like.com*), este capítulo irá descrever a implementação de uma aplicação para ensaiar e estudar o comportamento de algumas das estruturas estudadas anteriormente. Assim, o objectivo deste capítulo consiste na apresentação das funcionalidades criadas para a aplicação em causa.

A sua construção está assente no padrão de arquitectura *MVC – Model-View-Controller* – Figura 20, uma vez que através deste esquema, torna-se possível reproduzir a interacção existente entre as várias componentes da aplicação.

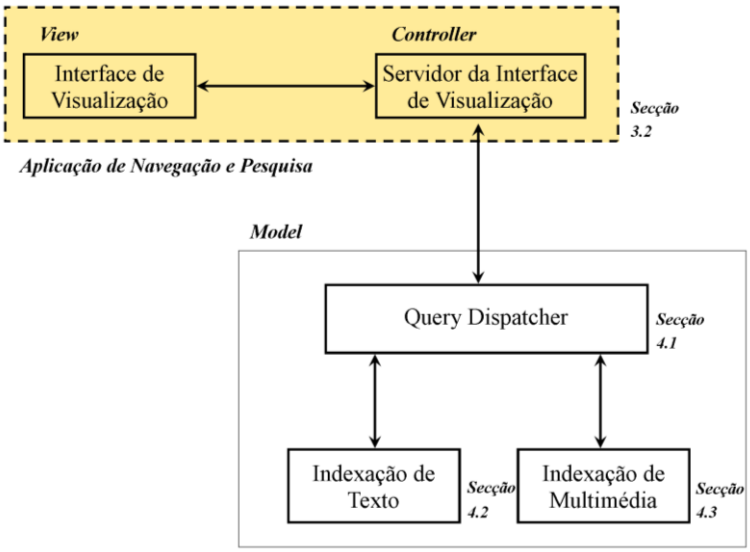

Figura 20 - Esquema da aplicação de navegação e pesquisa.

Analisando esta arquitectura, verificamos que a aplicação é constituída por duas grandes componentes:

- **Aplicação de Navegação e Pesquisa** – esta componente é constituída por uma interface de visualização, apresentada ao utilizador (correspondendo à componente *View* da arquitectura *MVC*), juntamente com um servidor, para controlo de todas as operações que são executadas pelos utilizadores sobre essa interface (correspondendo assim, à componente *Controller* do *MVC*).

- **Model** – esta componente do *MVC*, irá conter todos os serviços essenciais para a obtenção da informação essencial à execução das operações de pesquisa desencadeadas pelos utilizadores. Tal como ilustrado na Figura 20, esta componente é constituída por:

  - **Query Dispatcher** – corresponde a um serviço desenvolvido para possibilitar a interacção com os diversos serviços indexação existentes.

  - **Indexação de Texto** – trata-se de um serviço que possibilita a indexação de informação textual relativa às imagens existentes no repositório.

  - **Indexação de Multimédia** – trata-se de um serviço desenvolvido para permitir a indexação de descritores de baixo nível, nomeadamente cores e texturas, de modo a possibilitar a realização de pesquisas por semelhança.

Os items correspondentes ao modelo serão analisados em maior detalhe no Capítulo 4. A motivação subjacente a esta organização prende-se com a possibilidade de integração de outros tipos de serviços com o *Query Dispatcher*, permitindo assim a extensão para novos métodos de indexação que possam vir a ser integrados.

As secções que se seguem irão apresentar a tecnologia utilizada para o desenvolvimento da interface de utilizador (*framework GWT – Google Web Toolkit*[40]), assim como a implementação da Aplicação de Navegação e de Pesquisa, as suas funcionalidades e os paineis/janelas correspondentes a essas funcionalidades.

3.1 GWT – Google Web Toolkit

A interface de utilizador desenvolvida nesta dissertação teve como base a utilização da

[40] *Framework* desenvolvida pela Google (http://code.google.com/intl/pt-PT/webtoolkit/)

framework GWT – Google Web Toolkit e suas extensões (no caso especifico tratou-se do *SmartGWT*[41]).

A utilização desta *framework* possibilita a qualquer programador, desenvolver aplicações *Web* através do uso da linguagem *Java*, tal como acontece no processo de desenvolvimento de conteúdo *JavaScript* e, ao mesmo tempo, obter o benefício de se utilizar *debug* e executar código passo a passo, características existentes na linguagem *Java*.

Na construção de uma aplicação *Java* em *GWT* destacam-se duas componentes: um lado cliente, onde é definida a interface de utilizador da aplicação; e um lado servidor, onde irão estar definidas todas as funções a serem executadas através de pedidos realizados no lado cliente da aplicação. Um esquema de uma possível aplicação pode ser visualizado através da Figura 21 onde é possível observar a separação entre o código que é executado no lado do cliente e no lado do servidor. Note-se que a parte da implementação correspondente à interface de utilizador é compilado para *Javascript* enquanto que a parte da implementação correspondente à execução no servidor é compilado para *Java bytecode*. No fundo, a separação *View-Controller* é feita automaticamente pelo *Framework GWT*.

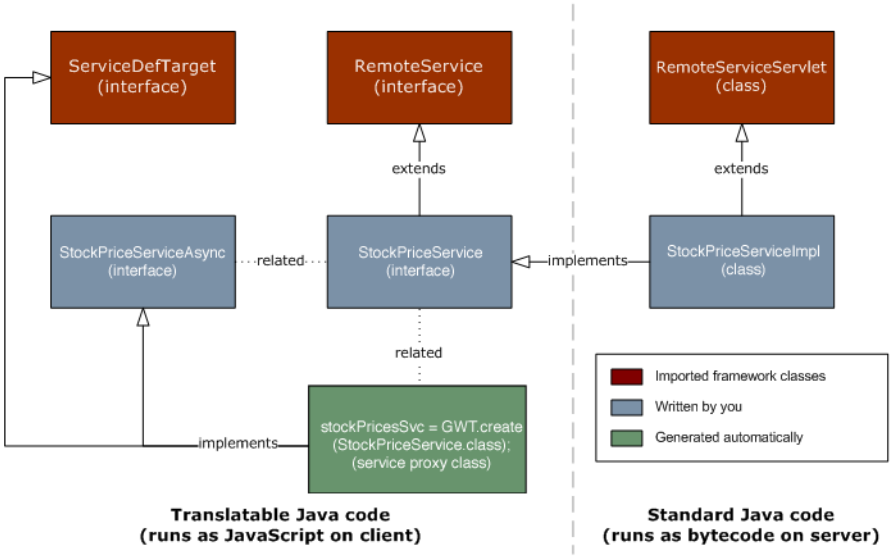

Figura 21 - Esquema apresenta a construção do *GWT*, focando em 3 elementos que envolvem a chamada de procedimentos em servidores remotos (http://code.google.com/intl/pt-PT/webtoolkit/doc/latest/tutorial/RPC.html).

As comunicações entre o servidor e o cliente são realizadas por meio de *Remote Procedure Calls – GWT RPC*, permitindo que todas as comunicações *Java* sejam particularmente fáceis e eficientes de serem efectuadas. Trata-se de um tipo de comunicação muito semelhante ao

[41] Pagina oficial - http://code.google.com/p/smartgwt/;
Exemplos - http://www.smartclient.com/smartgwt/showcase/;

Java RMI[42], bastando para tal definir uma interface que especifique os métodos a serem utilizados. Estando essa interface de utilizador definida, quando um dado método é chamado, a partir do browser, o *GWT RPC* serializa automaticamente os argumentos, invoca o método adequado no servidor, desserializa esses argumentos e retorna os resultados ao lado cliente da aplicação.

Ao mesmo tempo que apresenta estas funcionalidades, o lado servidor da aplicação permite ainda estabelecer ligações/comunicações com outro tipo de aplicações/serviços (e.g., *Web Services*, repositórios, etc.), e traduzir o formato de representação dos pedidos/respostas entre o cliente e o servidor. Esta funcionalidade é especialmente útil para conversões entre *JSON* (Javascript Objects), *XML* etc.

Por outro lado, o facto de se utilizar a linguagem *Java* na construção de aplicações *GWT* permite ainda utilizar qualquer tipo de ferramentas de desenvolvimento (*IDE*s) (e.g., *Eclipse*, *IntelliJ*, *JProfiler*, *JUnit*), oferecendo aos seus utilizadores ganhos significativos na produtividade, quer em termos de automatismos de *refactoring* do *Java*, quer na completude do código.

Finalmente, caso as bibliotecas do *GWT* não forneçam a informação necessária, esta *framework* permite que seja possível introduzir código *JavaScript* no código do *Java*.

3.2 Aplicação de Navegação e Pesquisa

A principal motivação para o desenvolvimento desta aplicação é a partilha de imagens entre utilizadores do sistema, permitindo que se realizem pesquisas idênticas às descritas no sistema *QBIC* e no site *Like.com*, por meio de uma interface simples e intuitiva. O conjunto de operações permitidas encontra-se definido na Figura 22.

As funcionalidades incluídas nesta aplicação são desencadeadas do lado interface do utilizador - *View* (ou seja, por meio dos utilizadores), através de uma chamada assíncrona a uma função no controlador (*Controller*), enviando ao mesmo tempo os dados fundamentais para a execução dessa operação. Estando na posse desses dados, o controlador reencaminha esses dados para o *Query Dispatcher*, por meio de uma chamada à função correspondente à operação em causa. Ao executar essa função, o *Query Dispatcher* obtém os argumentos provenientes do controlador e realiza um tratamento específico para o tipo de operação que é pretendida (e.g. autenticação de utilizadores, inserção, actualização, remoção e pesquisas de imagens).

Para uma maior compreensão da complexidade existente na elaboração desta Aplicação de Navegação e de Pesquisa, a Figura 23 apresenta um diagrama de classes definindo o

[42] *Java Remote Method Invocation*, é uma interface de programação que permite a execução de chamadas remotas no estilo *RPC*.

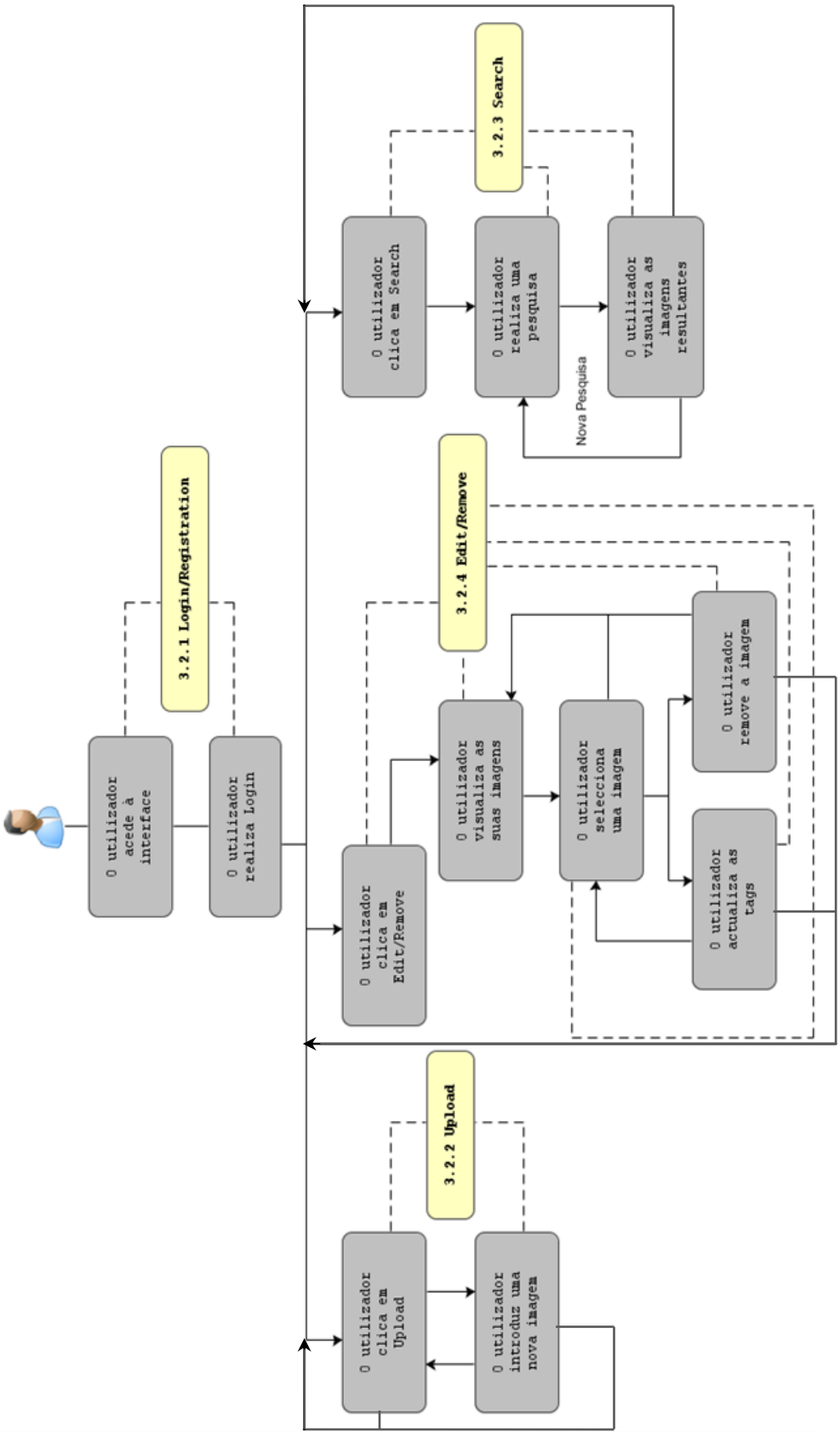

Figura 22 – Diagrama de actividades das operações permitidas pela aplicação.

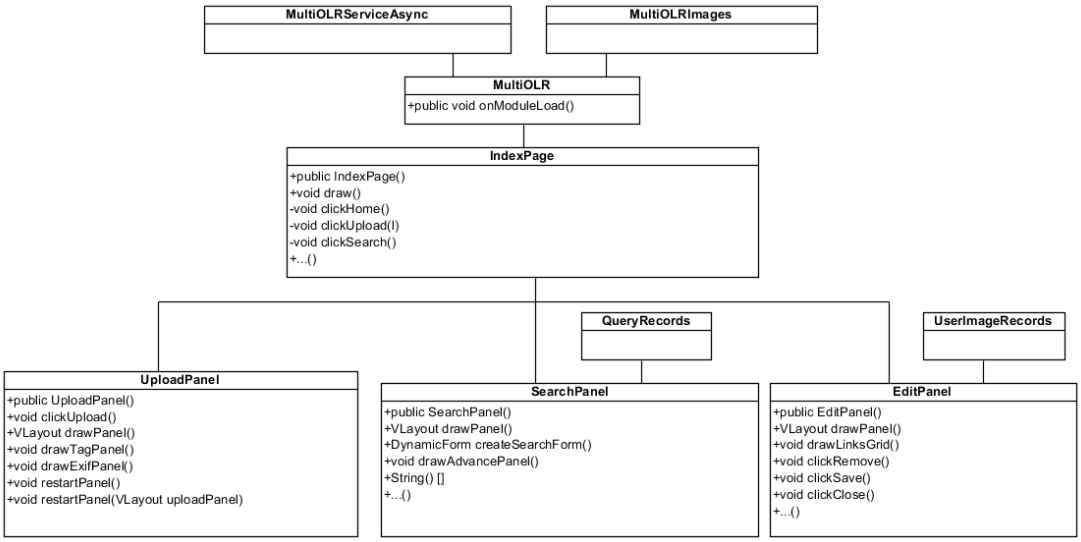

Figura 23 - Diagrama de classes da Aplicação de Navegação e Pesquisa.

conjunto de classes utilizadas para produzir os paineis correspondentes à aplicação, assim como as operações mencionadas anteriormente.

3.2.1 Login/Registration

O acesso à aplicação é realizado através de um *Web Browser*[43]. Contudo, para permitir a visualização das imagens existentes no repositório, os utilizadores necessitam de estar devidamente autenticados. Para tal, é necessário que tenham anteriormente procedido a um registo. Deste modo, quando se acede à aplicação por meio do *Web Browser*, é apresentado um menu idêntico ao exemplificado na Figura 24.

Figura 24 – Menu de *Login*.

[43] *Internet Explorer*, *Firefox*, *Opera*, *Safari* e *Google Chrome*.

Ao efectuar o login, os dados colocados no campo *username* e *password* são enviados para o servidor da interface por meio da função *loginResponse*, que ao receber os dados, invoca uma função específica no *Query Dispatcher* para a validação dos mesmos, permitindo o acesso ao utilizador (denominada por *login*). O resultado produzido por essa validação é encaminhado até ao cliente, de modo a informá-lo sobre a sua autenticação. Consoante o valor que é retornado, é apresentado um dos seguintes menus, Figura 25 e Figura 26.

Figura 25 - Exemplificação da ocorrência de inserção de dados de *Login* incorrectos.

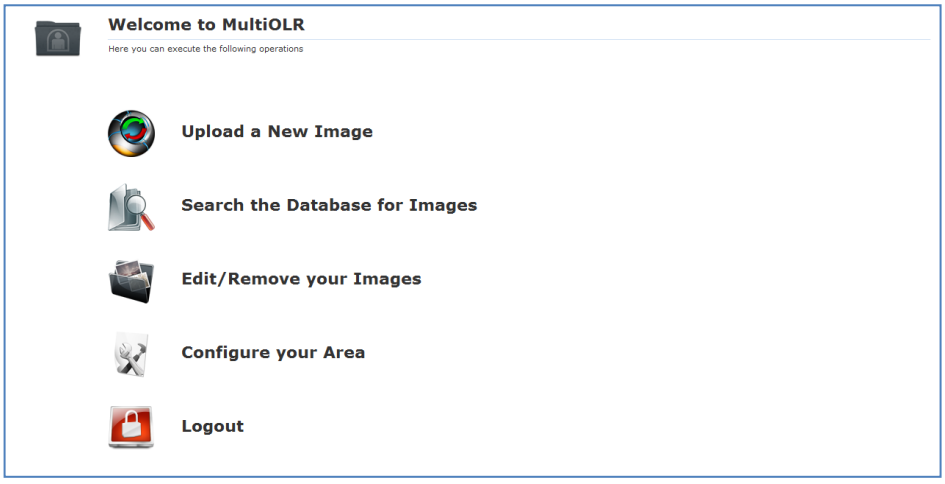

Figura 26 - Menu de apresentação após a validação dos dados de *Login*.

Ao analisarmos a Figura 26, verifica-se que o utilizador tem agora ao seu dispor um conjunto de operações que permitem a inserção, actualização, remoção e pesquisa de imagens, sendo cada uma destas operações analisadas nas secções subsequentes.

O conjunto de operações definidas para o painel de *Login* encontram-se desenvolvidas na classe *MultiOLR*, à qual depende de um conjunto de outras classes *MultiOLRServiceAsync*,

para estabelecer a comunicação com o controlador da aplicação e *MultiOLRImages*, possibilitando a utilização de imagens, nomeadamente no que diz respeito à apresentação de ícones. Por sua vez, estando os utilizadores autenticados a definição do painel representado na Figura 26 e as respectivas funcionalidades encontram-se definidas na classe *IndexPanel*, estando também dependente das classes *MultiOLRServiceAsync* e *MultiOLRImages*.

3.2.2 Upload

Seleccionando a opção de *Upload* existente no painel principal ou através do *link* existente na parte superior de cada painel, é apresentado o painel na Figura 27.

Figura 27 - Menu de Upload**.**

Analisando a Figura 27, o utilizador tem a opção de não só indicar qual a imagem que pretende introduzir, assim como associar informação textual de modo a enriquecer o conteúdo semântico da imagem em causa (por meio de inserção de *tags* e/ou de informação *Exif*[44]). Após a introdução desses dados, é desencadeado o processo de inserção da imagem no sistema, por meio do botão *Upload*. Através desta acção, é executada a chamada da função *uploadImage* existente no servidor da interface, reencaminhando de seguida esses dados para o *Query Dispatcher*, recorrendo à função denominada por *uploadImage*. Nesse serviço, é construído um documento de acordo com as especificações definidas no *schema* criado para o *serviço de Indexação de Texto* e, uma vez concluído, é executada uma *query* de inserção sobre o mesmo.

O conjunto de operações deste painel, encontram-se definidas na classe *UploadPanel*, à qual depende das classes *MultiOLRServiceAsync* e *MultiOLRImages*.

[44] *Exchangeable image file format*, é uma especificação seguida por fabricantes de câmeras digitais que gravam informações sobre as condições técnicas de captura da imagem junto ao arquivo da imagem propriamente dita na forma de metadados etiquetados.

3.2.3 Search

Seleccionando a opção de *Search* existente no painel principal ou através do *link* existente na parte superior de cada painel, é apresentado o painel na Figura 28.

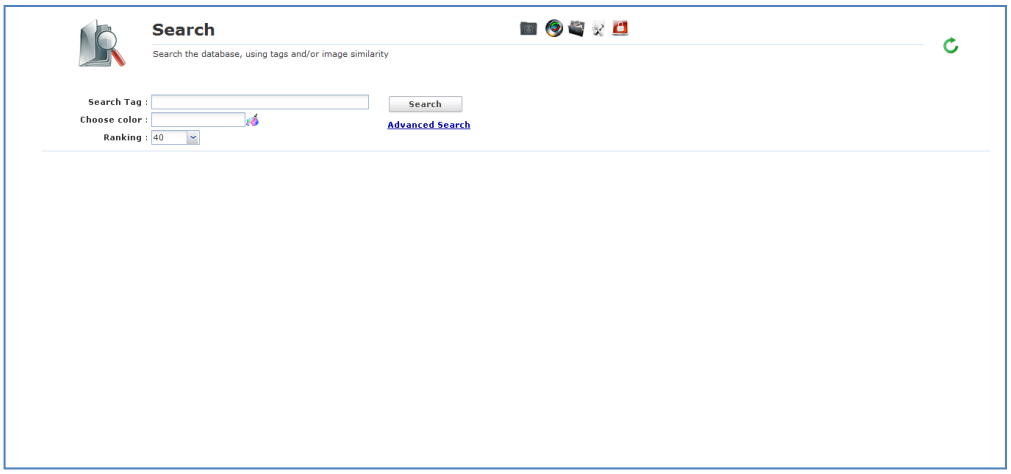

Figura 28 - Menu de *Search*.

Ao clicar no botão de *Search*, o texto existente no campo de texto (*Search Tag*) é enviado ao servidor da interface por meio da função *searchImage*. Posteriormente, essa *query* é reencaminhada para o *Query Dispatcher* por meio da execução da função *searchQuery*, para ser processada e executada sobre o *serviço de Indexação de Texto*.

Uma vez executada a *query*, as descobertas realizadas são então devolvidas ao *Controller*, de modo a identificar as imagens a serem apresentadas ao utilizador, reproduzindo algo semelhante ao indicado na Figura 29.

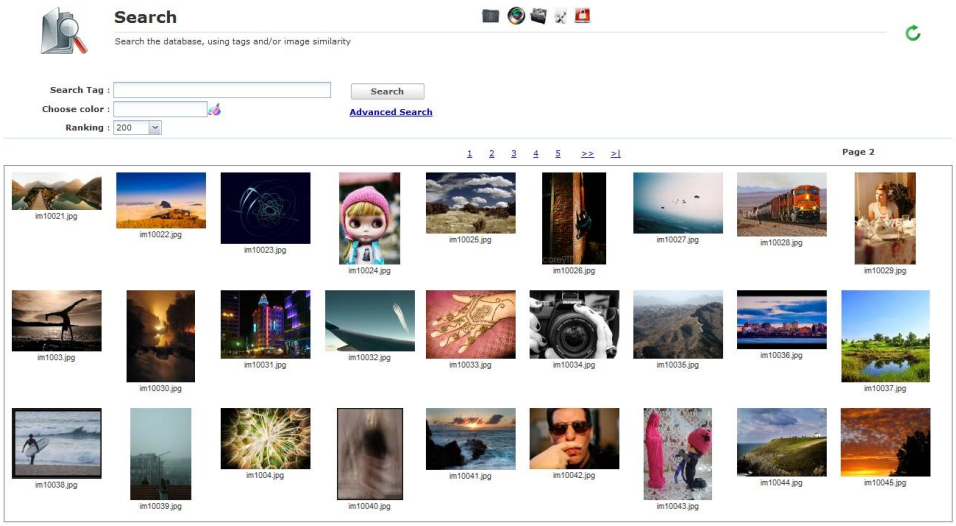

Figura 29 - Resultado obtido na sequência de uma pesquisa.

Seleccionando uma destas imagens, torna-se possível observar a informação mais

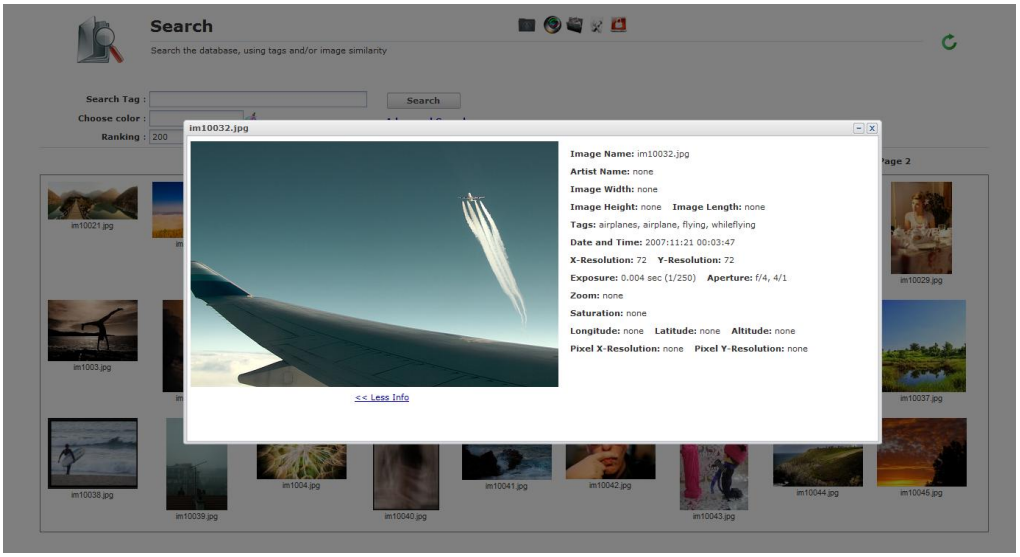

Figura 30 - Visualização do conteúdo informativo de uma imagem.

importante, associada a essa imagem – Figura 30. Este painel encontra-se definido na classe *SearchPanel*.

Para além deste modo de pesquisa por meio de informação textual, é também possível executar uma pesquisa através das características de baixo nível das imagens. Para tal, é necessário activar o painel de *Advanced Search*, onde este contém um conjunto de opções que permitem executar essas pesquisas, não só por meio dos descritores identificados nas imagens (através da selecção de um dos botões na coluna mais à esquerda), como também recorrendo a métricas para o efeito, Figura 31.

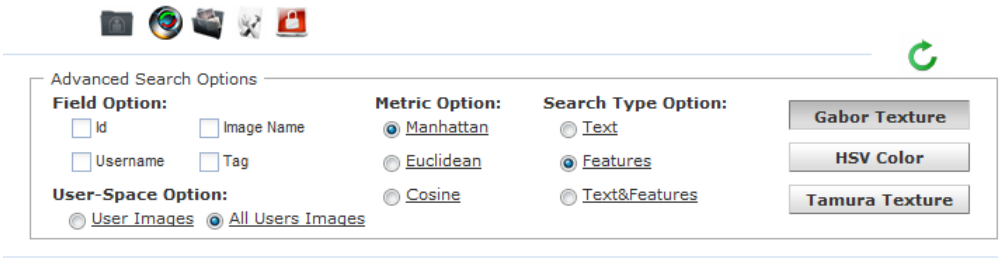

Figura 31 – Painel de realização de pesquisas avançadas.

A pesquisa por meio de um descritor é realizado escolhendo as opções da métrica (*Manhattan*, *Euclidean* e *Cosine*), do tipo de pesquisa (*Features* ou *Texto*&*Features*) e do descritor propriamente dito (*Gabor*, *HSV* e *Tamura*)[45]. Uma vez seleccionadas as opções pretendidas pelo utilizador, escolhe-se uma imagem, no conjunto de resultados obtidos na pesquisa anterior, para efectuar a pesquisa pretendida. A Figura 32 apresenta o resultado de uma

[45] A informação relativa a estes descritores encontra-se nas *secções 4.3.1.1* e *4.3.1.2*.

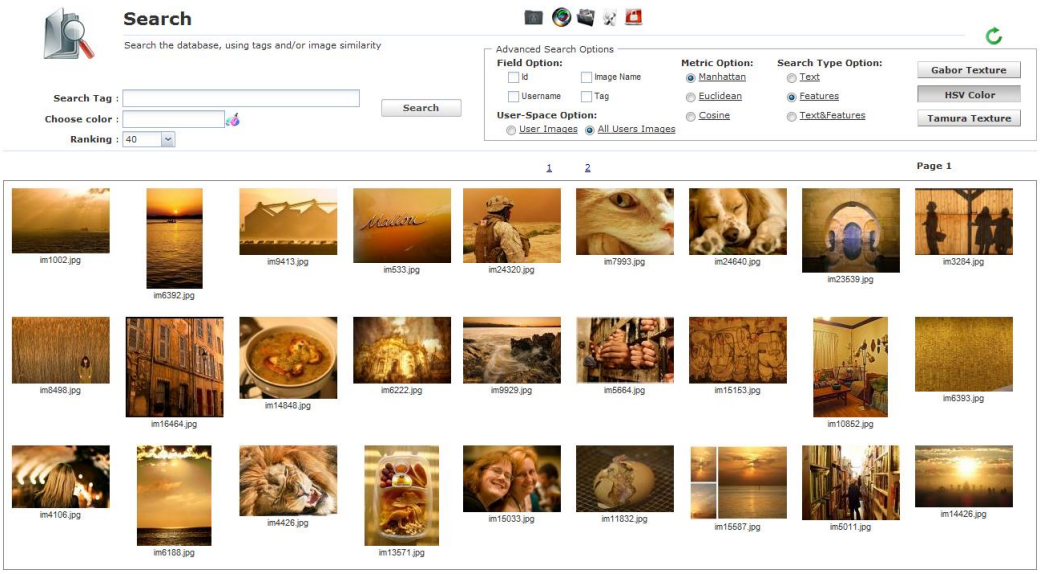

Figura 32 – Resultado da pesquisa por meio de um descritor de imagens.

pesquisa por meio da utilização de um descritor de baixo nível (*HSV*), utilizando a métrica de *Manhattan*.

A aplicação aqui desenvolvida torna ainda possível executar uma pesquisa recorrendo simultaneamente ao conteúdo textual e às características de baixo nível. Para tal, utilizam-se as opções existentes no painel *Advanced Search* e o conteúdo existente no campo de *Search Tag*.

3.2.4 Edit/Remove

Seleccionando a opção de *Edit/Remove* existente no painel principal ou através do *link* existente na parte superior de cada painel, é apresentado o painel na Figura 33.

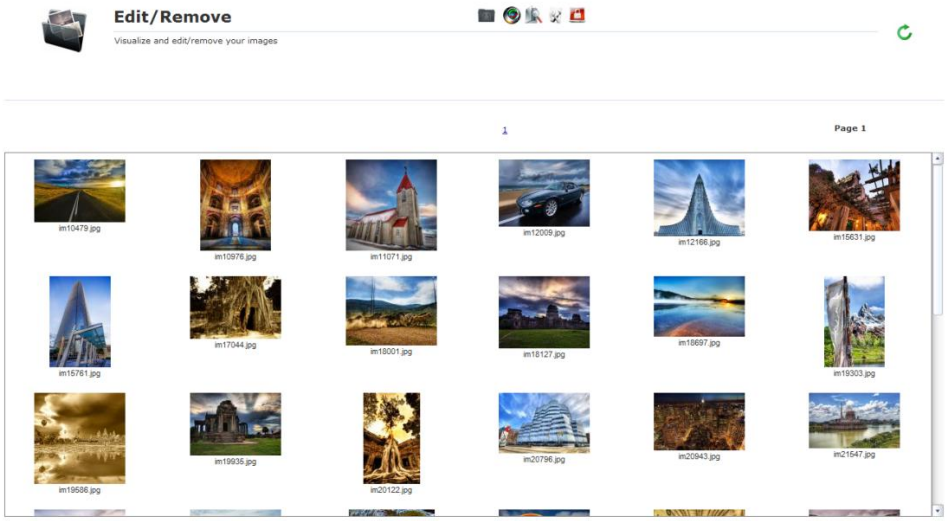

Figura 33 - Menu de *Edit/Remove*.

Neste painel são apresentadas todas as imagens que estão associadas a esse utilizador, resultantes de uma pesquisa efectuada, através de um processo muito semelhante ao descrito na *Secção 3.2.2*, na pesquisa efectuada por meio de texto. No caso em concreto, é utilizado o *username* para obter as imagens pretendidas.

Uma vez apresentas todas imagens do utilizador, é possível efectuar um conjunto de operações sobre estas, bastando para tal, seleccionar a imagem pretendida – Figura 34.

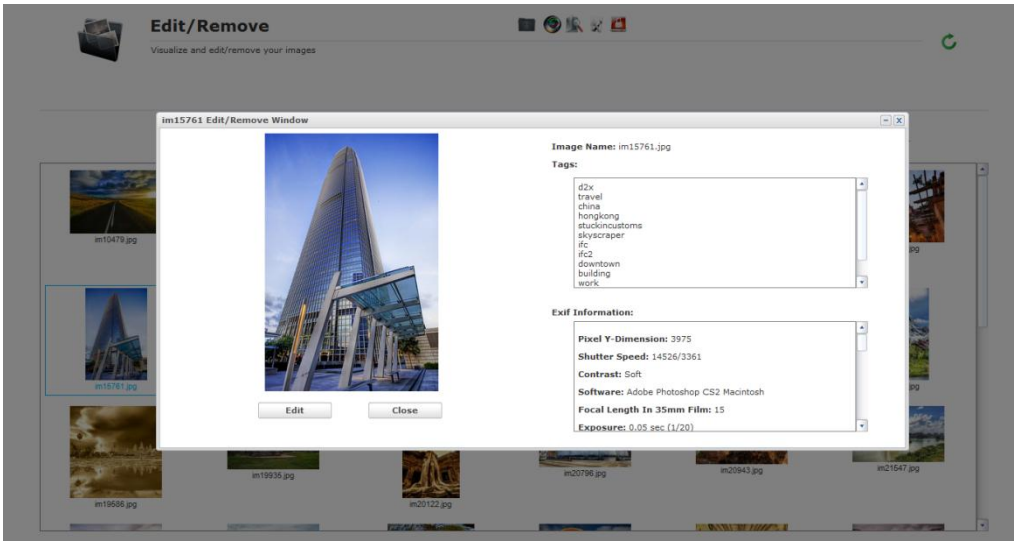

Figura 34 - Selecção de uma imagem do utilizador, no menu de *Edit/Remove*.

3.2.4.1 Actualização de uma imagem

Estando a imagem seleccionada, o utilizador pode actualizar o conteúdo correspondente às *tags*, acrescentando ou removendo-as, conforme o que pretender – Figura 35. Realizadas essas alterações, a nova informação criada é então enviada para o servidor da interface, através de uma chamada à função *editImage*, aquando da selecção do botão *Save*.

Por sua vez, o servidor chama a função *editImage* no *Query Dispatcher*, onde irá produzir um novo documento com base na informação inalterada e com o novo conteúdo de *tags*. Construído esse documento, é enviado um pedido ao *serviço de Indexação de Texto*, de modo a actualizar o conteúdo associado a essa imagem, utilizando para tal o documento então criado. Estando a informação da imagem actualizada no *serviço de Indexação de Texto*, o utilizador é informado do resultado desta operação.

3.2.4.2 Remoção de uma imagem

Estando a imagem seleccionada, o utilizador poderá proceder à sua remoção caso o pretenda efectuar. Uma vez desencadeado esse processo, por meio da função *removeImage* no servidor (seleccionando o botão *Remove*), são passados como argumentos o nome da imagem e o *username*. Após a recepção desses dados, é então chamada a função *removeImage* no *Query*

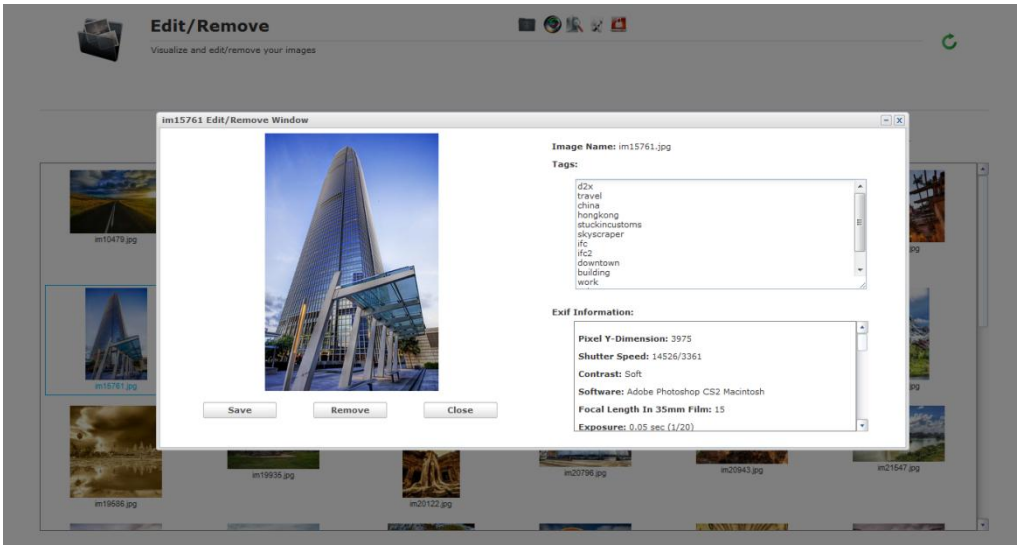

Figura 35 - Edição das *tags* de uma imagem, para um dado utilizador.

Dispatcher de modo a proceder à operação de remoção da imagem do repositório. Através destes dados é possível executar a operação de remoção no *SOLR*, utilizando para tal o identificador do documento que está associado à imagem (isto é, a concatenação do nome da imagem com o *username*). Uma vez terminada essa operação, informa-se o servidor da interface do sucedido. Caso a operação tiver sucesso, é executado um novo processo de pesquisa para obter as imagens associadas ao utilizador, de modo a actualizar o conjunto de imagens que estão a ser visualizadas.

O conjunto de operações indicadas na *secção 3.2.4* estão definidas na classe *EditPanel*.

3.3 Sumário

Através do desenvolvimento desta interface, tornou-se possível integrar os dois tipos de pesquisa identificados no Capítulo 2 e durante a análise dos sistemas *QBIC* e *Like.com*: pesquisa por texto e por semelhança visual.

Para além da operação de pesquisa, esta aplicação permite ainda inserir novo conteúdo multimédia (e.g. imagens), actualizar/remover *tags* associadas às imagens, bem como remover as imagens que foram introduzidas. Desta forma, garante-se a liberdade pretendida para o conjunto de operações aqui desenvolvidas.

4

Serviço de Indexação e Pesquisa

Como foi mencionado anteriormente aquando da definição da arquitectura *MVC* do sistema desenvolvido, este capítulo será dedicado à apresentação dos diferentes serviços de indexação e pesquisa desenvolvidos nesta dissertação. Estes serviços são utilizados pela aplicação e suportam o conjunto de operações essenciais ao funcionamento da aplicação. Os serviços em questão correspondem a um *Query Dispatcher*, um *serviço de Indexação de Texto* e um *serviço de Indexação de Multimédia*, Figura 36. Toda a comunicação entre os diferentes componentes é feita através de serviços *RESTFul* descritos na *secção 4.1.1*.

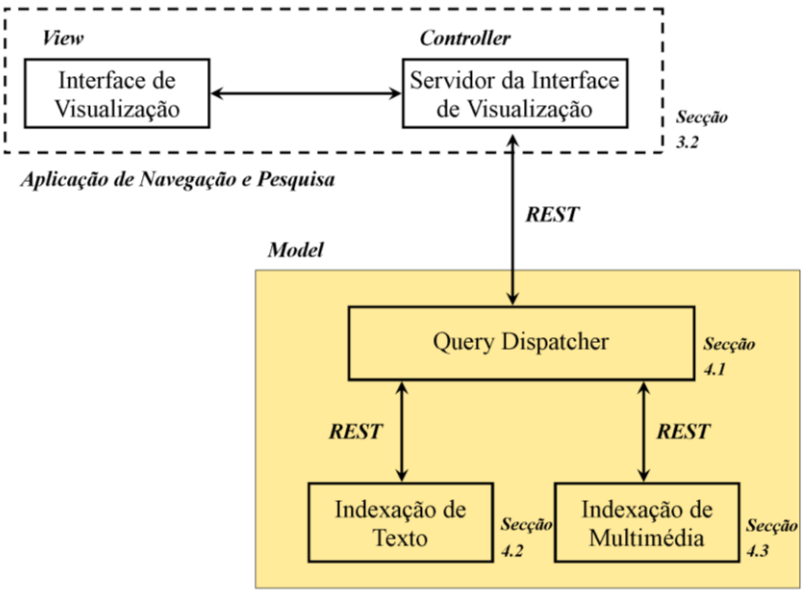

Figura 36 - Esquema da aplicação de navegação e pesquisa.

Estes serviços constituem o ponto fulcral do sistema desenvolvido uma vez que contém toda a informação associada às imagens existentes no repositório (informação textual e de

baixo nível).

Estando estes ligados à interface de visualização, estes serviços ficam em modo de espera, até que um pedido proveniente da mesma chegue ao *Query Dispatcher*.

Dependendo da operação desencadeada pelo utilizador, o *Query Dispatcher* poderá ter a necessidade de comunicar com ambos os serviços ou simplesmente com um deles, i.e., se o utilizador realizar uma pesquisa por meio da utilização de conteúdo textual, apenas o *serviço de Indexação de Texto* será utilizado neste processo para obter os resultados; caso o pedido de pesquisa incida na identificação de imagens com uma dada cor, apenas será utilizado o *serviço de Indexação de Multimédia* no processo de identificação de imagens representativas dessa mesma cor. No entanto, recorre-se ao *serviço de Indexação de Texto* para obter a informação textual associada a essas imagens que se obtém no resultado da pesquisa no *serviço de Indexação Multimédia*; finalmente, se o pedido utilizar tanto o conteúdo textual, como também uma dada cor, a pesquisa irá incidir nos dois tipos de serviços.

Ao realizar os pedidos sobre os serviços específicos, o *Query Dispatcher* fica a aguardar os resultados dessas operações. Caso utilize ambos os serviços, e se, por exemplo a operação corresponder a uma pesquisa, então o *Query Dispatcher* adquire os resultados de ambos e intersecta-os. Uma vez obtidos esses resultados e processados, são então enviados para o controlador (*Controller*), de modo a informar a *View* do sucesso da operação que foi executada. Dependendo da operação, diferentes tipos de resposta serão apresentados aos utilizadores.

De modo a obter esta interoperabilidade entre o *Query Dispatcher* e os demais serviços utilizados, foi necessário recorrer a um conjunto de tecnologias e *frameworks* adicionais, nos quais se englobam o *SOLR* e o *Restlet*[46].

4.1 Query Dispatcher (QD)

O *Query Dispatcher* (*QD*) consiste num *Web Service* desenvolvido para encaminhar os pedidos da aplicação cliente (neste caso a aplicação de pesquisa do Capítulo 3) para o(s) serviço(s) de indexação e pesquisa adequados e posteriormente devolver o resultado da pesquisa à aplicação cliente (i.e., devolver o resultado ao controlador da aplicação (*Controller*).

Devido às características apresentadas, este serviço pode provocar um *bottleneck*, uma vez que as comunicações centram-se todas neste serviço. Para minimizar os factores subjacentes, pelo facto de se tratar de um serviço central da aplicação desenvolvida, foi tido em conta várias técnicas de desenvolvimento de *Web Services*.

Do conjunto das arquitecturas existentes que possibilitam o desenvolvimento deste tipo

[46] Página oficial - http://www.restlet.org/

de serviços (e.g. *SOAP*[47] e *RESTFul*), a arquitectura *RESTFul* (Fielding 2000) ofereceu as melhores soluções para garantir estas funcionalidades.

4.1.1 Arquitectura RESTFul

A *Transferência de Estado Representacional* (*REpresentational State Transfer*) ou somente *REST* consiste numa técnica para permitir a implementação de sistemas de hipermédia de grande escala (e.g. a *World Wide Web* (*WWW*)), devido ao seu protocolo de comunicação assentar sobre *HTTP*.

Na sua definição, a arquitectura *RESTFul* está assente em 4 pressupostos:

- **Identificação dos recursos por meio de *URI*** – os serviços *RESTFul* contêm um conjunto de recursos endereçados por um *URI* específico. Desta forma, permite que um cliente possa aceder directamente um dado recurso.

- **Interface Uniforme** – Os recursos são manipulados segundo um conjunto de 4 operações (criação, leitura, actualização e remoção, ou seja, *PUT* para inserir um dado recurso, *GET* para obter o seu conteúdo, *POST* para actualizá-lo e *DELETE* para o eliminar).

- **Mensagens auto-descritivas** – Os recursos são dissociados da sua representação, para permitir o acesso aos seus conteúdos. Para tal, recorre-se a um conjunto variado de formatos (e.g. *HTML*, *XML*, texto normal, *PDF*, *JPEG*, *JSON*, etc.).

- **Interacções *Stateful* por meio de hiperligações** – todas as interacções efectuadas neste tipo de serviços, são consideradas como mensagens independentes. Desta forma, fica garantida a preservação do estado do recurso.

Por outro lado, o facto de conter estes princípios base e de tirar partido dos standards *W3C/IETF* torna os serviços *RESTFul* muito simples de implementar. Na verdade, a existência de uma variedade de linguagens de programação e de plataformas de desenvolvimento, que permitem a interacção *HTTP* cliente/servidor, torna a implementação deste tipo serviços fácil de realizar.

Um exemplo desta circunstância, recorre ao uso da linguagem de programação *Java* para o desenvolvimento de *Web Services RESTFul – framework Restlet*.

[47] *Simple Object Access Protocol*, desenvolvido por Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein em 1998, de modo a criar um protocolo de acesso a objectos.

4.1.2 Framework Restlet

O desenvolvimento do *Restlet* deveu-se à inexistência de uma *Framework* no *Java* que permitisse auxiliar os programadores no desenvolvimento de aplicações, segundo uma arquitectura *RESTFul*. Para tal, foi criada uma *API* onde os conceitos base estivessem subjacentes (e. g., *resource*, *representation*, *connector*, *content*, *media type*, *language*, etc.), ou seja, foram definidas um conjunto de classes que constituiriam a base da *framework*, possibilitando o desenvolvimento de aplicações *RESTFul* de um modo rápido e eficiente.

Não obstante da *API* definida, esta *framework* contem métodos de abstracção dos protocolos de comunicação, de modo a garantir que as mesmas interfaces possam utilizar múltiplos protocolos, sem prejudicar o desempenho da aplicação (e.g. *HTTP*, *SMTP*, etc). Tal metodologia é conseguida através de um único método *Java* (*public void handle(Request request, Response response)* pretencente à classe *Resource*), ao qual se cingem todos os aspectos que envolvem as chamadas *REST*.

Para além do método aqui descrito, existem classes que facilitam o processamento das chamadas na identificação do recurso alvo. De um modo geral, o processamento das chamadas *REST* é realizado através de duas fases:

1. A chamada é processada por meio de uma cadeia de classes *Restlet*, que permitem que esta seja filtrada (e. g., para salvaguardar questões de segurança ou para extrair variáveis do *URI*). Uma vez filtrada, torna-se possível identificar o recurso adequado a esta chamada (a classe *Finder*), ou seja, é realizada uma procura para identificar a instância do recurso, conseguida através do processamento da informação existente no *request*.

2. Estando identificada a classe referente ao recurso, o processamento da chamada é então transferido para esta. Ao processá-la, a resposta poderá tomar varias representações, com diferentes formatos e linguagens, sendo utilizado um algoritmo específico para obter a melhor representação desse recurso.

Em suma, o *Restlet* possibilita aos programadores a construção de aplicações baseadas numa arquitectura *RESTFul*, de uma forma fácil e eficiente, através de um conjunto de classes e interfaces suficientemente extensíveis e reutilizáveis.

4.1.3 Operações

Como foi referido, o *QD* implementa um conjunto de operações que possibilita a interacção com os *serviços de Indexação*, no momento do arranque do *QD* (também designada pela *API* do *QD*):

- **login** – esta função permite verificar se um dado utilizador pertence ao sistema.
- **addUser** – esta função permite inserir um utilizador na aplicação criada, verificando se o dado utilizador já pertence ao sistema. Caso o utilizador exista, informa-se a interface do sucedido que, posteriormente, informa o utilizador da aplicação.
- **removeUser** – esta função permite remover um utilizador da aplicação, onde a informação relativa ao utilizador é eliminada do sistema, mantendo-se as imagens no repositório.
- **uploadImage** – permite que se insira uma imagem no sistema, ou seja, cria um documento segundo a informação proveniente da aplicação e adiciona-o no *serviço de Indexação de Texto*.
- **searchQuery** – procura no *serviço de Indexação de Texto* ou de *Multimédia* ou em ambos, as imagens que correspondem a um dado conteúdo introduzido pelo utilizador. Para tal, o utilizador basta introduzir um conjunto de uma ou mais *tags*, para obter todas as imagens que contêm essa informação (neste caso será utilizado o *serviço de Indexação de Texto*); referir que tipo de imagens que pretende visualizar indicando o tipo de descritora utilizar na pesquisa (recorrendo ao *serviço de Indexação Multimédia*, para obter essa informação); ou utilizar ambos os dados e executar a pesquisa em ambos os serviços. Posteriormente, os resultados obtidos em ambos os serviços são intersectados.
- **getUserImageQuery** – permite que seja possível realizar uma pesquisa no *serviço de Indexação de Texto*, em busca de todas as imagens que pertencem ao utilizador em causa, indicando o ponto de inicio de apresentação de resultados.
- **editImage** – actualiza a informação contida no documento existente no *serviço de Indexação de Texto*, ou seja, cria um novo documento com base na informação proveniente da interface do utilizador com a informação do documento original, reintroduzindo-o no *serviço de Indexação de Texto* com a nova informação que foi gerada.

- **removeImage** – remove uma dada imagem dos repositórios, ou seja, dado o nome da imagem e do utilizador, é desencadeada uma *query* de remoção sobre o *serviço de Indexação Texto*, com base na concatenação destes dois atributos (nome da imagem + utilizador), como também é definida uma *query* de remoção sobre o *serviço de Indexação Multimédia*, recorrendo simplesmente ao nome da imagem.

4.1.4 Encaminhamento de Operações

A comunicação com os vários serviços criados ocorre no momento em que o *QD* arranca. Para tal, é utilizado um *URI* para cada serviço, indicando o endereço da máquina onde este se localiza e a porta associada ao serviço. Estabelecidas as ligações, o *QD* fica a partir deste momento disponível para realizar as operações descritas anteriormente.

O encaminhamento de operações para o *Indexador de Texto* ou *Multimédia* existe apenas para operações de pesquisas uma vez que o trabalho realizado nesta dissertação centrou-se na análise do comportamento de estruturas de indexação de dados de alta dimensão. Contudo, as outras operações (inserção, actualização e remoção) estão a funcionar unicamente para o *serviço de Indexação de Texto*.

A diferenciação no processo de pesquisa é conseguida através de um valor passado como parâmetro na função de pesquisa (*searchQuery*), funcionando como uma *flag*. Dependo do valor em causa, este vai indicar se a pesquisa irá ser processada num dos serviços ou em ambos.

De modo a realizar a operação pretendida, são utilizados *URI*s específicos para cada um dos serviços. No caso do *serviço de Indexação de Texto*, o *URL* utilizado no processo de pesquisa descrito por parâmetros separados pelo caracter "&" e segundo o seguinte formato:

```
http://.../select?indent=on&version=2.2&q=im1&fq=&start=0&row
s=10&fl=*%2Cscore&qt=standard&wt=standard&debugQuery=on&expla
inOther=&hl=on&hl.fl=Image_Name
```

Para realizar uma pesquisa neste tipo de serviço não é necessário recorrer a todos estes parâmetros. No caso em concreto, é possível definir os parâmetros que pretendemos utilizar, como o ponto inicial da pesquisa, o número de resultados a retornar, e a/s própria/s *tag*/s, facilitando este processo.

No que diz respeito ao *serviço de Indexação de Multimédia*, este inclui 4 processos de pesquisa, aos quais 3 deles permitem realizar uma pesquisa sobre o tipo de descritor em causa, ao passo que o 4º executa uma pesquisa sobre todos os descritores que são utilizados pelo repositório.

Independentemente do tipo de pesquisa em causa, o *URL* utilizado neste processo apresenta os mesmos parâmetros em todas elas, tomando, como exemplo, o seguinte formato:

```
http://.../select?search=color&q=im1.jpg&metric=0&rank=40&sta
rt=0&rows=40&structures=all&queryRange=100
```

4.1.5 Pesquisas Multimodais

Sendo um dos objectivos da aplicação desenvolvida permitir a integração dos dois tipos de pesquisa, tornou-se lógico conjugar os dois serviços de forma a suportar pesquisas multimodais. Ao receber os resultados de ambos, o *QD* intersecta-os através dos rankings das imagens comuns aos dois rankings. Posteriormente, retorna os resultados deste processamento para a interface (*Controller*), de modo a permitir a visualização dos resultados.

O processamento desta operação é realizado na função definida na *API* do *QD* - *searchQuery*.

4.2 Serviço de Indexação de Texto

Este serviço está assente sobre um servidor de texto denominado por *SOLR*, cujas suas características foram analisadas no Capítulo 2. Nesta secção iremos descrever a configuração do *SOLR* para suportar as diferentes pesquisas por texto. Detalhes sobre o funcionamento do *SOLR* estão presentes no Capítulo 2.

As alterações efectuadas neste servidor de texto incidiram no ficheiro *schema* que carateriza o formato dos dados e quais dados são indexados por documento. Definiram-se campos que englobassem toda a informação descrita nos ficheiros *Exif*, essencial para o processo de visualização/apresentação das imagens na aplicação que foi desenvolvida.

4.2.1 Schema

Como foi referido, foram feitas alterações no *schema* ao nível dos *fields*. Foram introduzidos um conjunto de campos aos quais permitissem englobar toda a informação existente nos ficheiros *Exif*. Para tal, foram definidos *826* campos, sendo *822* respeitantes à informação *Exif* das imagens.

Para além de se indicar os campos a utilizar nos documentos a serem indexados, cada um deles tem de estar dotado de um conjunto de propriedades, de forma a potenciar o processo de pesquisa e de possibilitar um conjunto de alternativas aos utilizadores na criação/actualização dos documentos neste repositório. Na Tabela 2 é possível observar uma pequena amostra dos campos inseridos no *schema* e as suas respectivas propriedades.

As linhas identificadas a azul, correspondem aos campos que todos os documentos do repositório partilham em comum, sendo os restantes tuplos específicos da imagem. Um modo de ser possível observar um documento no *SOLR* é através da execução de uma pesquisa num Web Browser. O resultado obtido, por meio de uma pesquisa será a apresentação de todos os documentos, com os seus respectivos campos, Figura 37.

Tabela 2 – Apresentação de algum dos campos definidos no ficheiro *schema* do *SOLR*, com as suas respectivas propriedades.

name	type	Indexed	stored	required	multiValued
id	string	true	true	true	false
Owner_Id	text	true	true	false	false
Image_Name	textgen	true	true	false	false
tag	textgen	true	true	false	true
Final_Height	int	true	true	false	false
Panorama	text	true	true	false	false
Urgency	int	true	true	false	false
Contrast	text	true	true	false	true
Headline	text	true	true	false	false
Software	text	true	true	false	true
Flash_Mode	int	true	true	false	false
…					

Figura 37 – Resultado de uma pesquisa por meio de um *Web Browser*, onde se apresenta o conjunto de campos existentes no documento.

4.3 Serviço de Indexação de Multimédia

Como foi referido no decurso desta dissertação, a utilização de descritores de baixo nível no processo de inserção, remoção e pesquisa é algo complexo e muita das vezes utiliza-se o conteúdo textual para ultrapassar essas dificuldades.

O objectivo deste serviço é o de permitir a execução de pesquisas eficientes sobre descritores de baixo nível e, ao mesmo tempo, transpor algumas das características do *SOLR* para o mesmo (nomeadamente permitir a execução de pesquisas rápidas e eficientes).

A dificuldade imposta neste tipo de dados consiste na sua elevada variância, na medida em que o número de imagens no repositório e a dimensão dos seus descritores ser muito elevada, influenciando negativamente o tempo de construção dos índices, assim como o tempo necessário para realizar uma pesquisa. Desta forma, é necessário identificar a melhor estrutura de indexação para o tipo de dados que estamos a trabalhar e, ao mesmo tempo, garantir a maior precisão dos dados aquando de uma pesquisa.

Em suma é necessário conjugar todos estes factores e garantir, em tempo útil, respostas aos pedidos que são realizados, sendo a análise desta dissertação centrada nos problemas mencionados anteriormente, Figura 38.

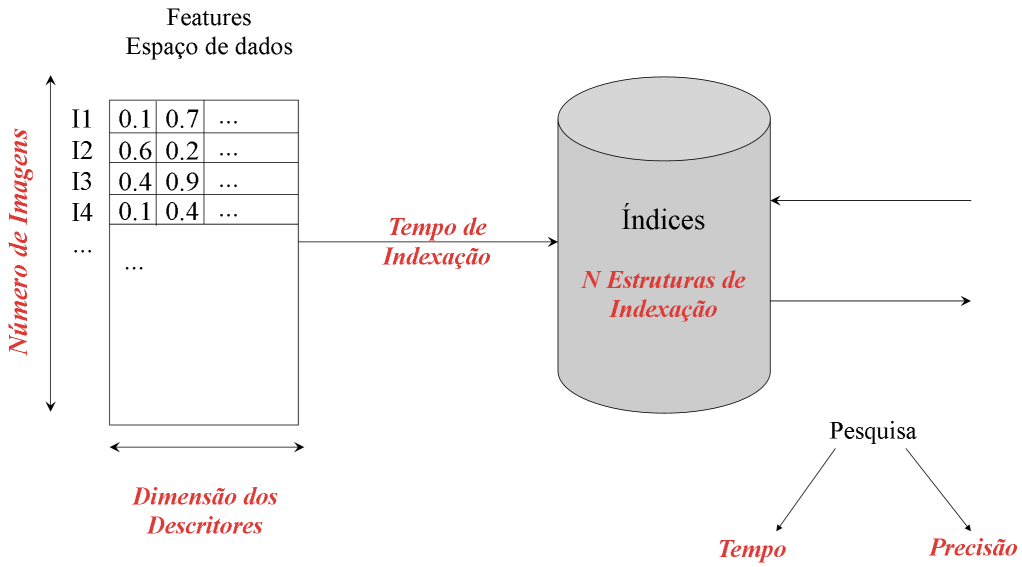

Figura 38 – Identificação dos principais problemas subjacentes nos índices com descritores de baixo nível.

Tendo como base estes objectivos, o *serviço de Indexação de Multimédia* está assente sobre a *framework Restlet*, permitindo assim recriar as funcionalidades existentes no *SOLR*. O modo de construção e de funcionamento do *SOLR* aproxima-se em muito da arquitectura *RESTFul*, uma vez que o tipo de operações que este permite a muito assemelha-se com os aspectos deste tipo de arquitectura. Assim recorrendo ao *Restlet*, é possível recriar essas

funcionalidades.

4.3.1 Análise de Multimédia

Estando a trabalhar com descritores de baixo nível, os conteúdos multimédia (nomeadamente as imagens) podem produzir um conjunto variado de descritores. No caso em concreto, foram utilizados apenas 3 tipos de descritores de baixo nível (*Gabor filters*, *Tamura filters* - Espaço de Textura - e *HSV Color Histograms*), aos quais foram determinados através de um programa externo, não sendo integrado nesta dissertação, por sair fora do objectivo desta.

4.3.1.1 Descritores de Textura

A textura apresenta um papel importante no que se refere à identificação de similaridade de imagens, observando-se um conjunto diversificado de técnicas capazes de extrair características subjacentes nas imagens. Umas das técnicas que foram desenvolvidas assentam na utilização de filtros, como são os casos do *Gabor* e do *Tamura*.

4.3.1.1.1 Gabor

Esta técnica de obtenção de características/descritores de imagens é das mais utilizadas, uma vez que os resultados que são produzidos superam, em muito, outras técnicas de extracção de descritores de baixo nível baseados em textura (Zhang, Wong et al. 2000).

Em traços gerais, esta técnica está assente num grupo de *wavelets*, em que cada *wavelet* captura a energia numa frequência e direcção específica. Expandindo o sinal através desta propriedade, torna-se possível a descrição localizada da frequência e desta forma capturar as características locais do sinal. Assim, é possível obter os descritores da textura através da distribuição dos diversos grupos.

Por outro lado, esta técnica contém propriedades que permite melhorar o seu desempenho da análise sobre as texturas, no que se refere à orientação quer na escala da amostra (frequência), de modo a aproximar-se na noção da frequência espacial da visão humana.

4.3.1.1.2 Tamura

O desenvolvimento da técnica *Tamura* (Tamura, Mori et al. 1978) teve como base as características da percepção visual humana no que diz respeito à densidade, contraste, direccionalidade, delimitação, simetria e irregularidade. Deste modo, foram necessários criar 6 tipos de descritores para produzir esse efeito. Contudo, identificou-se que 3 dessas propriedades são particularmente importantes (densidade, contraste e direccionalidade), uma vez que correlacionam-se mais directamente com a percepção humana, através dos seguintes factos:

- **Densidade** – indica o tamanho dos elementos presentes na textura, ou seja, quanto maior for o valor do tamanho dos elementos, maior será a densidade. Por exemplo, se existirem duas texturas diferentes, uma com elevada densidade e outra com densidade reduzida, apenas a elevada será tida em conta. Em traços gerais o cálculo do valor da densidade consiste em utilizar operadores de vários tamanhos, de acordo com os tipos de densidades existentes, na imagem a tratar.

- **Contraste** – refere-se à qualidade da imagem em causa e pode ser afectado segundo 4 factores: alcance dinâmico dos tons de cinzento; a polarização da distribuição do preto e do branco no histograma de tons de cinzento; acentuação dos limites; e o período de padrões repetidos.

- **Direccionalidade** – este conceito pretende identificar a presença de orientação na textura, ou seja se duas texturas diferem na orientação das suas texturas, estas são consideradas por apresentarem a mesma direccionalidade. Assim, para calcular a direccionalidade horizontal e vertical são utilizadas matrizes de 3×3.

Para possibilitar a utilização destas propriedades é necessário definir um valor a cada pixel, para o descritor em causa. Estando na posse destes valores é possível utilizá-los de duas formas. Em primeiro lugar, considerá-los como uma imagem *RGB* e guardá-la como tal e, em segundo, é criado um histograma de 3 dimensões a partir desses valores.

4.3.1.2 Descritores de Cor: Histogramas de HSV

A cor constituí num ponto muito importante na extracção de informação de uma imagem e recorrendo à utilização de histogramas de cor para realizar comparações tornou-se numa técnica bastante popular na indexação de imagens e de vídeos. Tal deve-se, à fácil obtenção desses índices, uma vez que são eficazes e apresentam um baixo poder de processamento para os obter, o que trás vantagens significativas em sistema que utilizam este tipo de indexação.

A utilização de histogramas *HSV* consiste num método que permite obter a informação necessária para realizar a indexação, por meio da cor. Através destes histogramas torna-se possível eliminar possíveis ruídos contidos nas imagens, uma vez que o processamento da imagem é realizado independentemente dos canais de cor e, por outro lado, permite que a navegação no espaço de cores seja intuitiva.

4.3.2 Operações de Indexação

No momento em que se inicia este serviço, é carregada toda a informação dos descritores de baixo nível que foi gerado especificamente para o repositório multimédia. Nesse processo,

o serviço vai verificar a presença dos dados e caso os dados não estejam presentes, o serviço irá proceder á construção dos índices, através de um conjunto de estruturas implementadas para o conjunto de imagens existentes. Como referido anteriormente, o tipo de descritores utilizados correspondem ao *Gabor*, *Tamura* e *HSV Color Histograms*, onde a relação que se estabelece entre as várias imagens, é conseguida através de algumas funções de semelhança estudadas no Capítulo 2 (*Manhattan*, *Euclidiana* e *Co-seno*).

4.3.3 Operações de Pesquisa

Este serviço irá estar em comunicação com o *QD*, permitindo que sempre que se realizem *queries* que necessitem a utilização de descritores de baixo nível, este possa realizar pesquisas sobre o repositório gerado e retornar esses resultados ao *QD*.

A pesquisa processa-se ao nível de um dos descritores ou em todos eles, definindo um conjunto de parâmetros necessários para obter a pesquisa em concreto (esses parâmetros podem ser observados na Figura 39).

Através destes parâmetros, torna-se possível verificar o comportamento dos índices para este tipo de dados. Esta análise recai, principalmente, na verificação das imagens que são obtidas na pesquisa, com as que são produzidas numa pesquisa linear.

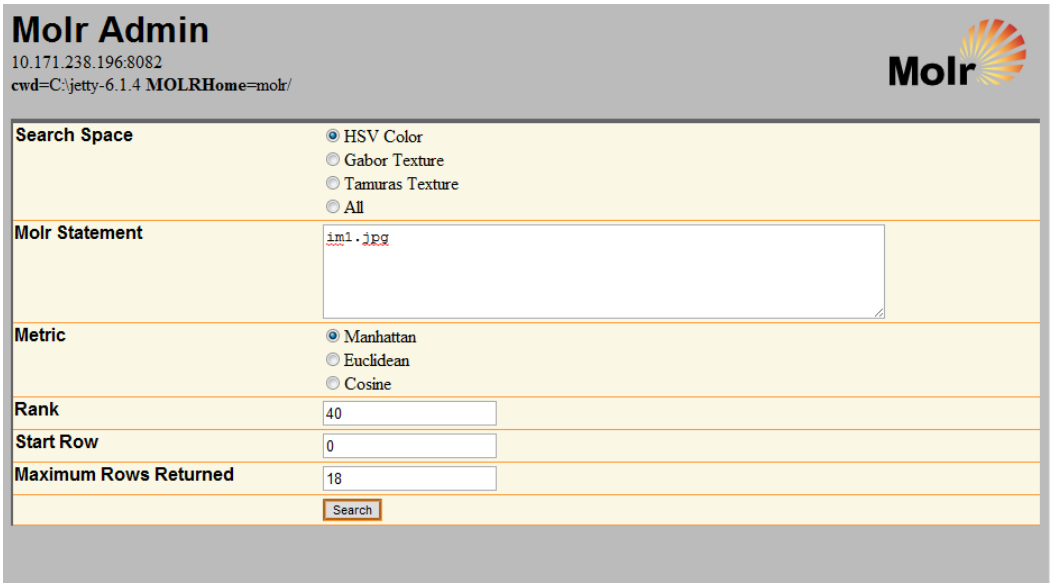

Figura 39 – Interface que permite a elaboração de *queries*.

4.3.4 Interface de Administração

Existindo o objectivo de ambos os serviços de indexação se aproximarem um do outro, em termos de funcionalidades permitidas e eficiência, a criação de uma interface de administração *Web* para o *serviço de Indexação de Multimédia* tornou-se um processo natural de aproximação dos dois serviços, Figura 40.

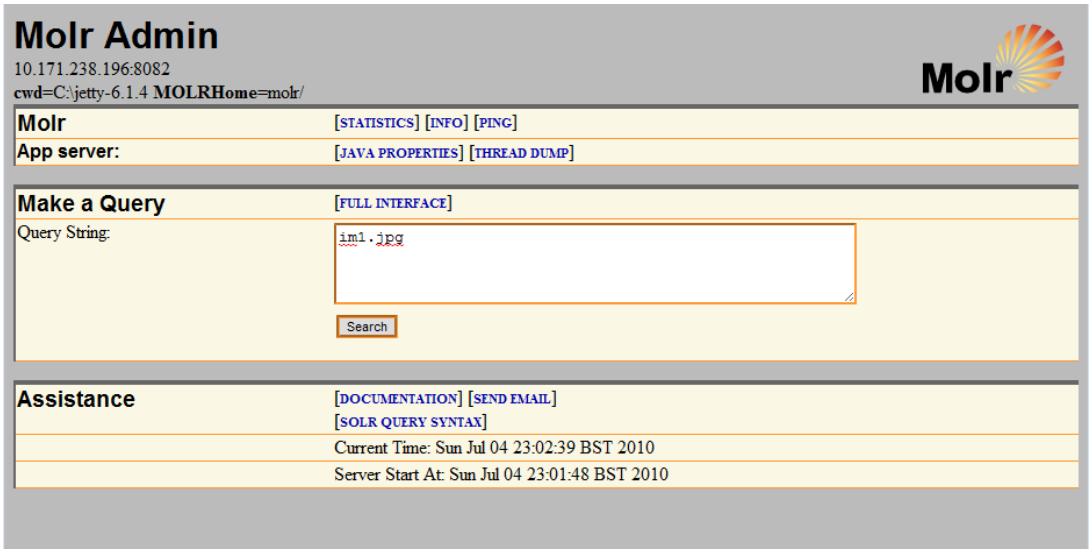

Figura 40 - Interface de administração do serviço de *Indexação de Multimédia*.

Por meio de um *layout* muito semelhante ao utilizado no *serviço de Indexação de Texto*, permite que qualquer utilizador/administrador do repositório obtenha uma facilidade de interacção, uma vez que conhece o modo de funcionamento das operações que estão disponíveis.

Através desta interface, é possível observar o estado do repositório, ou seja, verificar o número de imagens existentes no repositório, o número de pesquisas realizadas, a documentação de apoio para o conjunto de funcionalidades e classes criadas para este serviço, assim como permitir a realização de pesquisas sobre os tipos de descritores utilizados, por meio de uma interface exemplificada na Figura 40.

Ao realizar-se uma pesquisa por meio desta interface, o resultado por defeito vem em formato *XML*, indicando a *query* introduzida pelo utilizador, o tempo requerido para efectuar a pesquisa em causa e os resultados que produziu, indicando o grau de semelhança existente entre a imagem que originou a pesquisa e o nome da imagem que se obteve como resultado. Na Figura 41, apresenta um resultado de uma pesquisa neste tipo de serviço.

Contudo, a aplicação desenvolvida não fica restrita à utilização de *XML* como retorno de respostas a *queries* efectuadas sobre este tipo de serviço. Do mesmo modo que acontece no *serviço de Indexação de Texto*, os dados que são produzidos numa pesquisa podem ser reproduzidos segundo um diferente formato. No caso em concreto, o novo formato disponível corresponde ao *JSON*.

De modo a tirar partido deste novo formato, foi definido um pacote no *QD* (*molr*) ao qual engloba um conjunto de classes que estabelecem a comunicação com este serviço e efectuam *queries* sobre o mesmo, obtendo as respostas em formato *JSON* (processo esse, muito semelhante ao que é utilizado no *serviço de Indexação de Texto*).

```
−<response>
  −<query>
     <image>images/im1.jpg</image>
     <metric>Manhattan</metric>
   </query>
  −<result>
    −<M-Tree numFound="40" time="98.05567 miliseconds">
      −<doc>
         <Filename>images/im1.jpg</Filename>
         <Rank>0.0</Rank>
       </doc>
      −<doc>
         <Filename>images/im15051.jpg</Filename>
         <Rank>5.122608</Rank>
       </doc>
      −<doc>
         <Filename>images/im5047.jpg</Filename>
         <Rank>5.128023</Rank>
       </doc>
      −<doc>
         <Filename>images/im4217.jpg</Filename>
         <Rank>5.448476</Rank>
       </doc>
      −<doc>
         <Filename>images/im8625.jpg</Filename>
         <Rank>5.4818277</Rank>
```


Figura 41 – Resultado obtido numa pesquisa efectuada neste serviço de Indexação.

Por último, é de salientar ainda que, a interface recorre à utilização de *resources*, ou seja, a construção do conteúdo *Web* para visualização nos *Web Browsers*, consiste na introdução de conteúdo *HTML* no *resource* correspondente. Assim, para visualizar o conteúdo *HTML* associado ao *resource*, basta simplesmente indicar o endereço *URL* do mesmo, obtendo os resultados na Figura 39 e Figura 40.

4.4 Sumário

De um modo geral, a utilização da *framework Restlet* veio favorecer em muito o desenvolvimento do *serviço de Indexação Multimédia*, na medida em que possibilitou recriar a aproximação desejada com o *serviço de Indexação de Texto*.

Por outro lado, conseguiu-se a compatibilidade desejada entre os descritores de baixo nível e os de alto nível, requerendo por vezes o acesso a ambos os serviços para visualização dos dados na interface criada para aplicação.

Por último, de referir o papel integrador que o *Query Dispatcher* detém neste sistema, uma vez que este controla todos os acessos aos diversos serviços, ficando toda a carga de processamento dos dados provenientes desses serviços nele próprio, deixando a organização dos dados a apresentar ao utilizador, remetida para interface da aplicação.

5

Análise de Métodos de Indexação de Multimédia

A última parte desta dissertação incide no estudo do comportamento de algumas das estruturas referidas no Capítulo 2. Iremos testar diversas características das estruturas de indexação, nomeadamente os tempos de construção dos índices, os tempos de pesquisa e precisão dos resultados, para um conjunto alargado de imagens – Figura 42. Os resultados que advêm desta análise irão ser comparados ao método de referência, que no caso em concreto consiste num índice não estruturado, cujos dados estão dispostos linearmente.

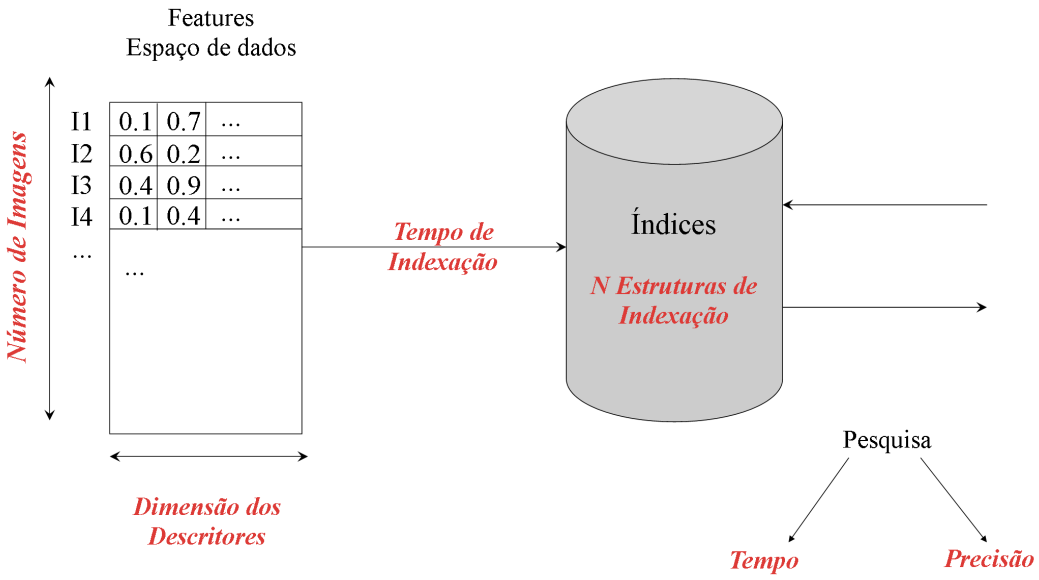

Figura 42 – Identificação dos principais pontos de análise em índices com descritores de baixo nível.

5.1 Estruturas de Indexação Implementadas

Tendo em conta o tipo de dados utilizados e estudo efectuado sobre as diversas estruturas específicas para indexação de conteúdo multimédia (Capítulo 2), considerei que as estruturas mais aconselháveis correspondem à: *VP-Tree*, *GH-Tree*, *GNAT* (neste caso são utilizadas duas *GNATs*, uma com *3 pivots* e outra com *7 pivots*, permitindo verificar a eficiência da estrutura, à medida que o número de *pivots* aumenta), *SA-Tree* e *M-Tree*. A selecção destas estruturas depreende-se da representação espacial de cada imagem. Através dos descritores gerados, é possível inferir que cada imagem é representada por um ponto no espaço e, por essa razão, a utilização tanto da *R-Tree*, como da *SS-Tree* (e respectivas variantes) não podem ser tidas em conta, uma vez que cada imagem necessitaria de delimitar uma dada região ou área (e.g. uma zona geográfica).

No que se refere à *VP-Tree*, *GH-Tree*, *GNAT*, a problemática da construção dos seus índices prende-se na identificação dos *pivots* ou, por outras palavras, identificar o conjunto de imagens mais relevantes do repositório que permitem definir uma estruturação eficiente (árvore) sobre este tipo de dados. Para tal, a construção destas estruturas seguiram os princípios identificados no Capítulo 2 (*Secções 2.4.1*, *2.4.2* e *2.4.3*).

No que diz respeito à *VP-Tree*, a sua implementação adoptou a variante proposta por (Yianilos 1993), descrita na *secção 2.4.1*, no qual são escolhidas aleatoriamente imagens para formarem um pequeno grupo, onde será seleccionado o *pivot* do índice (para efeitos de avaliação foram escolhidas 20 imagens). Estando na posse deste conjunto de imagens, seleccionou-se a melhor imagem recorrendo à utilização de uma métrica específica (*Manhattan*, *Euclideana* e *Coseno*), calculando o grau de semelhança entre imagens desse conjunto. O *pivot* é determinado pela imagem que apresenta a melhor média de distâncias, sendo o raio de actuação obtido pela distância máxima, resultante do cálculo do grau de semelhança entre os diversos descritores do conjunto. Uma vez determinado o *pivot*, o processo de construção da árvore verifica se a imagem a introduzir está contida (ou não) na área de actuação do *pivot*.

Relativamente à *GH-Tree* e *GNATs*, o processo de obtenção dos *pivots* foi conseguido através da escolha aleatória de imagens no repositório, onde o afastamento entre pivots foi determinado recorrendo aos valores das distâncias. No caso em concreto, a distância mínima é definida entre o último *pivot* determinado e o novo *pivot* a introduzir (o valor foi calculado segundo a métrica de *Manhattan*, sendo o esse valor igual a *20*). Estando os *pivots* determinados, o processo de construção da árvore respeita o descrito no Capítulo 2 (*secções 2.4.2* e *2.4.3*), ou seja, cada nó da árvore contém o mesmo número de elementos correspondentes ao número de *pivots* e a inserção dependerá do valor da distância em relação aos *pivots*.

Obstantes deste processo de construção encontram-se a *SA-Tree* e a *M-Tree*.

No que se refere à *SA-Tree* (*secção 2.4.4*), o processo de construção dos índices é executado por meio de um conjunto de passos, de modo a minimizar os problemas identificados pelo algoritmo de determinação dos melhores caminhos. Para tal, a solução encontrada consistiu em percorrer todas as imagens existentes no repositório e determinar o conjunto de *50 imagens* que fossem próximas à imagem que estamos a analisar (designado por conjunto de vizinhos). Uma vez determinados os conjuntos de vizinhos de cada imagem, procede-se a um processo de filtragem que garante que cada conjunto vizinho produzido contém todas as imagens que são verdadeiramente mais próximas e evitando a formação de ciclos durante o processo de pesquisa (ou seja, que o processo de filtragem produza o *Diagrama de Delaunay*).

Finalmente, no que se refere à construção da *M-Tree* (*secção 2.4.5*), o ponto problemático incide no número de imagens que cada nó poderá conter. Para o conjunto de imagens escolhido, foi necessário verificar o número mínimo de entradas que cada nó pode conter, recorrendo-se a construções sucessivas do índice, verificando sempre o número de imagens resultantes na raiz. Deste modo, garantiu-se que o número de imagens não excedia o número de entradas da raiz, uma vez que esta árvore é composta por 3 níveis.

5.2 Tipos de dados considerados

No estudo aqui realizado foram consideradas duas situações. Em primeiro lugar, foi efectuada uma análise do comportamento das estruturas face a diferentes tipos de descritores e, em segundo lugar, analisar a influência do aumento progressivo do número de imagens no repositório, com o aumento progressivo da dimensão dos descritores (análise em larga escala para diferentes dimensões e imagens).

Para tal, foram definidos 3 tipos de descritores (*Gabor*, *Tamura* e *HSV* – ver Capítulo 4 nas *secções 4.3.1.1* e *4.3.1.2*) – aos quais vão incidir sobre 2 conjuntos de imagens (um conjunto com 25.000 imagens e outro com 250.000 de imagens).

No que se refere ao conjunto de 25.000 imagens, foram produzidos os 3 tipos de descritores: (i) um descritor referente ao espaço de textura *Gabor*, ao qual cada imagem existente no repositório é definida segundo um descritor com 48 dimensões, (ii) um descritor referente ao espaço de cor *HSV*, ao qual cada imagem é definida segundo um descritor com 432 dimensões e, (iii) um descritor referente ao espaço de textura *Tamura*, ao qual cada imagem é definida por um descritor com 27 dimensões. Assim, através destes descritores, torna-se possível avaliar o comportamento das estruturas face a existência de descritores definidos em espaços procura diferentes.

No que diz respeito ao conjunto de 250.000 de imagens, foram obtidos descritores

correspondentes ao espaço de cor (*HSV Color Histogram*), cuja variação incide no número de dimensões para representar as componentes *H*, *S* e *V* (no caso em concreto foram determinados 4 descritores). Ao recorrer-se a um conjunto considerável de imagens, permite que seja possível avaliar o comportamento das estruturas implementadas na presença de repositórios com um elevado número de imagens e dimensões. Deste modo, torna-se possível determinar quais as estruturas melhor adaptadas para espaço de procura em causa.

As imagens utilizadas, com as quais permitiram obter estes descritores, foram retiradas da base de dados do Flickr e sujeitas a um processo de extracção de características utilizando *Gabor filters*, *Tamura filters* e *HSV Color Histogram*.

5.3 Resultados obtidos e Discussão

Como foi mencionado, a análise do comportamento das estruturas implementadas, irá ter em conta conjuntos de imagens de dimensão considerada, identificando os descritores resultantes desses conjuntos. Por outro lado, de acordo com as métricas utilizadas (*Manhattan*, *Euclideana* e *Co-seno*), será necessário verificar o tempo de indexação da informação de baixo nível, como também observar o comportamento das estruturas no processo de pesquisa, avaliando-as segundo a rapidez na obtenção de resultados e na precisão dos mesmos (*Coeficiente de Pearson*) – Figura 42.

5.3.1 Coeficiente de Pearson

O *Coeficiente de Pearson*, consiste numa medida que quantifica o grau de correlação entre dois rankings, ou seja, verifica o grau de semelhança existente entre os resultados produzidos por dois índices distintos. Este coeficiente, também conhecido por *Coeficiente de Spearman para correlação de rankings*, foi desenvolvido por *Charles Spearman* e é definido segundo a seguinte expressão:

$$\rho = 1 - \frac{6\sum {d_i}^2}{n(n^2-1)},$$

onde $d_i = x_i - y_i$ corresponde à diferença entre o valor do *rank* x e o *rank* y na posição i e n ao tamanho dos dois rankings.

Uma exemplificação deste processo encontra-se definida no esquema apresentado na Figura 43.

5.3.2 Condições de Teste

A análise efectuada ao comportamento das estruturas resultou num conjunto de testes efectuados sobre uma máquina com as seguintes características:

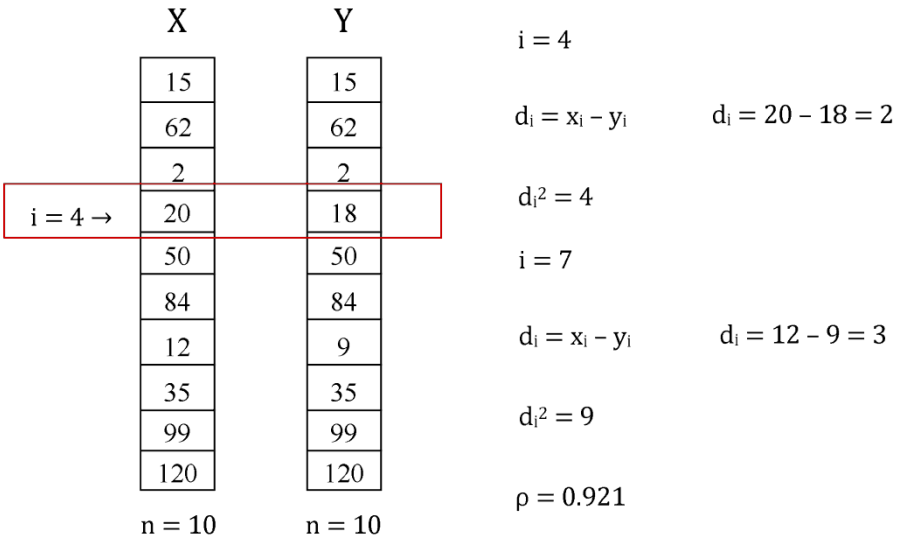

Figura 43 – Exemplificação da aplicação do *Coeficiente de Pearson*.

- PC notebook HP Pavilion dv9890ep
- Processador: Intel(R) Core(TM)2 Duo CPU T9300 @2.50GHz 2.50GHz
- Memória: 4 GB DRAM
- Sistema Operativo: Windows 7 Professional 64-bits

Visto tratar-se de uma máquina sujeita a escalonamento de processos, derivado do Sistema de Operação, foi necessário garantir o máximo isolamento possível para minimizar o erro na exactidão dos resultados. Para tal, a solução encontrada correspondeu à interrupção de processos e limitar a memória atribuída à máquina virtual onde os testes foram executados.

Uma vez efectuadas estas alterações, os testes consistiram na realização de um conjunto de 100 pesquisas sobre as estruturas (uma vez que as *queries* utilizadas foram sempre diferentes de estrutura para estrutura, o valor aqui indicado permite minimizar o favorecimento que uma dada *query* pode exercer sobre uma dada estrutura), utilizando o algoritmo de *Nearest Neighbour* para obter os resultados (objectivo consiste em obter as imagens mais próximas a uma dada *query*). Para tal, as *queries* utilizadas recorreram a imagens do repositório (escolhidas aleatoriamente), uma vez que estas já se encontram processadas. Através destas pesquisas obteve-se os valores médios dos tempos de pesquisa de cada estrutura, como também a precisão dos resultados gerados.

Para efeitos da análise efectuada sobre a precisão dos resultados, o *ranking* da pesquisa linear será considerado como o correcto, sendo os gráficos gerados como uma representação

da correlação dos *rankings* das estruturas face ao *ranking* do índice não estruturado.

5.3.3 Análise do comportamento para o conjunto de 25.000 imagens

Como foi referido anteriormente, a análise irá centrar-se na observação dos resultados gerados na construção dos índices (tempo de construção dos índices e o espaço ocupado em disco), nos tempos e precisão das pesquisas nas diversas estruturas implementadas. Assim, iniciemos a nossa análise no comportamento das estruturas na fase de construção dos seus índices.

Tabela 3 – Tempo de construção dos índices para o espaço de textura *Gabor* e *Tamura*, juntamente com o espaço de cor *HSV*.

	Espaço de Textura (*Gabor*)	**Espaço de Cor (*HSV*)**	**Espaço de Textura (*Tamura*)**
Linear	n.a.	n.a.	n.a.
SA-Tree	1127 segundos	2750 segundos	1021 segundos
M-Tree	4 segundos	7 segundos	5 segundos
VP-Tree	4.5 segundos	6 segundos	4 segundos
GH-Tree	21 segundos	18.5 segundos	12.5 segundos
GNAT-3	17 segundos	16.5 segundos	17 segundos
GNAT-7	20 segundos	17 segundos	18 segundos

5.3.3.1 Complexidade na Indexação

A Tabela 3 resume os tempos obtidos na construção dos diferentes índices para o conjunto de descritores definidos (*Gabor*, *HSV* e *Tamura*).

Perante estes resultados verifica-se que o processo de construção do índice da *SA-Tree* é que apresenta um tempo de construção bastante elevado, face às demais estruturas. A razão subjacente nos resultados aqui apresentados, prende-se no algoritmo utilizado para determinar os conjuntos vizinhos das imagens, uma vez que é necessário percorrer todas as imagens do repositório para encontrar esse conjunto. Uma vez determinados esses conjuntos, é necessário proceder-se a uma filtragem dos mesmos de modo garantir que as imagens contidas nesses conjuntos são realmente as mais semelhantes (obtendo-se os melhores caminhos e evitando a ocorrência de ciclos durante a fase de pesquisa). Posto isto, o processo aqui especificado apresenta uma complexidade temporal $O((n \times m) \times l)$, onde n corresponde ao número de imagens do repositório, $m = n - 1$ o número de imagens percorridas para obter o conjunto de vizinhos e, l o número de imagens a percorrer para eliminar os falsos caminhos (através da comparação das distâncias).

Contrapondo a *SA-Tree*, encontram-se as restantes estruturas. Como é possível observar

na Tabela 3, as estruturas que se destacam são a *VP-Tree* e a *M-Tree*.

No que se refere à *VP-Tree*, os valores aqui apresentados resultam do facto da estrutura ser constituída por apenas 1 *pivot* e, o processo de indexação das imagens, apenas verifica se uma dada imagem irá pertencer (ou não) ao conjunto de imagem que esse *pivot* consegue abranger. Deste modo, a indexação irá apresentar uma complexidade temporal $O(n)$.
No caso da *M-Tree*, apesar da complexidade do seu algoritmo de inserção no que diz respeito à ocorrência de situações de saturação (*overflow*) dos nós e/ou folhas (o conteúdo dos nós e/ou folhas é dividido, criando-se novos nós e folhas com base na informação que foi gerada a partir dessa divisão), esta estrutura apresenta resultados bastante satisfatórios. Uma razão para o sucedido encontra-se no número de elementos máximo que cada nó ou folha poderá conter (esse valor foi obtido por meio de experimentações sucessivas, de modo a evitar que o número de imagens na raiz seja superior ao que esta pode conter). Desta forma, o valor atribuído ao número de entradas de cada nó permite inserções sem recorrer a sucessivas divisões do conteúdo dos nós ou folhas da árvore.

Relativamente às restantes estruturas implementadas, apesar de apresentarem tempos satisfatórios, a construção dos índices assenta na obtenção dos *pivots* segundo um processo de selecção aleatório de imagens, sujeito a um conjunto de restrições. Desta forma, torna-se natural que, por vezes, o tempo necessário para construir os índices seja superior aos aqui apresentados na Tabela 3.

Para além dos tempos aqui apresentados, é possível ainda aferir sobre o espaço que cada índice ocupa em disco – Tabela 4. Tal procedimento é conseguido através de uma análise sobre o comportamento das estruturas durante a fase de construção e da disposição do conteúdo nessas estruturas.

Tabela 4 – Tamanho dos índices guardados em disco, respectivo a uma métrica (KB - Kilobytes).

	Espaço de Textura (*Gabor*)	**Espaço de Cor (*HSV*)**	**Espaço de Textura (*Tamura*)**
Linear	4.688	42.188	2.637
SA-Tree	10.073	10.073	10.073
M-Tree	5.836	43.336	3.785
VP-Tree	245	245	245
GH-Tree	6.837	44.337	4.786
GNAT-3	6.837	44.337	4.786
GNAT-7	6.837	44.337	4.786

Toda a informação das estruturas é guardada em ficheiros. Dependendo da estrutura em causa e das ligações necessárias para esta operar correctamente, o tamanho dos ficheiros irá variar. Na maioria das estruturas escolhidas, a informação contida nesses ficheiros está sob a forma de objectos, de modo a preservar as ligações entre os vários nós e facilitar o carregamento da informação em memória. Devido a esse factor, os tamanhos desses ficheiros irão ser superiores aos utilizados pelo índice não estruturado.

No entanto existem excepções. Tanto a *SA-Tree* como a *VP-Tree*, guardam apenas os nomes das imagens e as respectivas distâncias. No caso da *SA-Tree* para cada imagem contem o conjunto de imagens semelhantes, com as respectivas distâncias. Relativamente à *VP-Tree*, é guardado o *pivot* da estrutura e os dois conjunto de imagens e respectivas distâncias (i.e., o conjunto abrangido pelo *pivot* e o conjunto que não é compreendido por esse mesmo *pivot*).

Em traços gerais, todas as estruturas que guardam a sua informação sob a forma de objectos, não se diferenciam muito do índice não estruturado, em termos de espaço que é ocupado em disco (embora estes apenas contêm a informação relativa aos descritores de baixo nível).

Estando efectuada a análise da complexidade de indexação, o próximo passo consiste na verificação do comportamento das diferentes estruturas durante a pesquisa. Nesta análise, irão ser verificados os comportamentos dos diferentes descritores (recorrendo a diferentes métricas – ver *secções 2.2.1.2*, *2.2.1.3* e *2.2.1.5*), observando o tempo necessário para efectuar uma pesquisa (Tempo de Pesquisa), como também a precisão dos resultados obtidos (*Coeficiente de Pearson*).

5.3.3.2 Pesquisa no Espaço de Cor HSV (432 dimensões)

Os resultados apresentados em cada um dos gráficos resultam de valores médios obtidos nas pesquisas efectuadas sobre o repositório de imagens (100 pesquisas).

5.3.3.2.1 Tempo de Pesquisa

Analisando os resultados obtidos para os diferentes valores de *ranking* (Figura 44, Figura 45 e Figura 46), verifica-se, tal como esperado, que a utilização de estruturas de dados para indexação de conteúdo de baixo nível permitiu diminuir o tempo de pesquisa sobre um repositório de imagens, quando comparado com um índice não estruturado, onde a informação se encontra disposta linearmente.

Embora apresentem melhorias significativas, existem estruturas que se destacam mais do que outras. Um desses casos corresponde à *SA-Tree*. Esta estrutura apresenta valores de tempos de pesquisa relativamente baixos, independentemente da métrica utilizada e dos comprimentos de *ranking* utilizados. Tal resultado deve-se ao facto das distâncias entre as imagens terem sido previamente determinadas no momento da construção da árvore. Desta

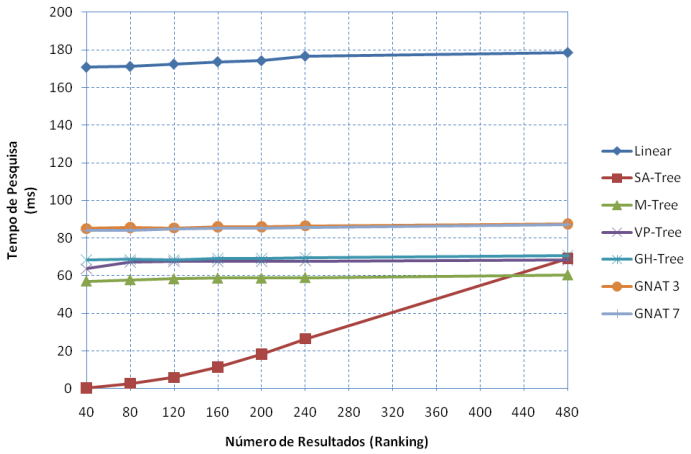

Figura 44 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica de *Manhattan*.

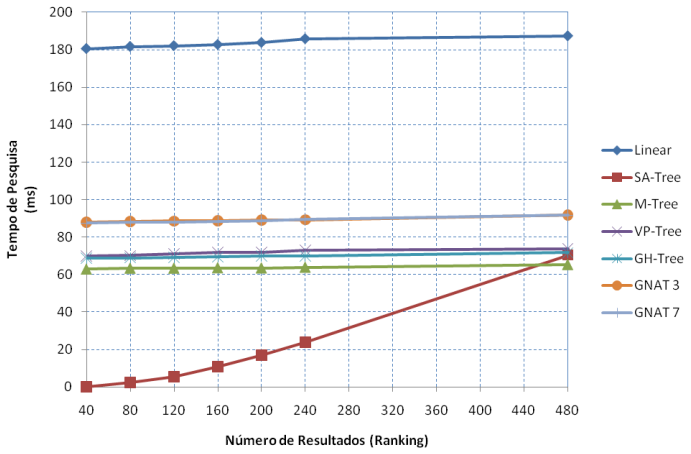

Figura 45 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica *Euclidiana*.

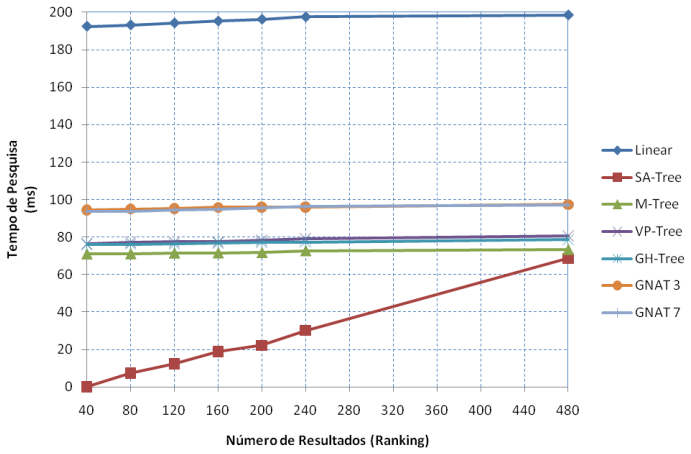

Figura 46 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica do *Co-seno*.

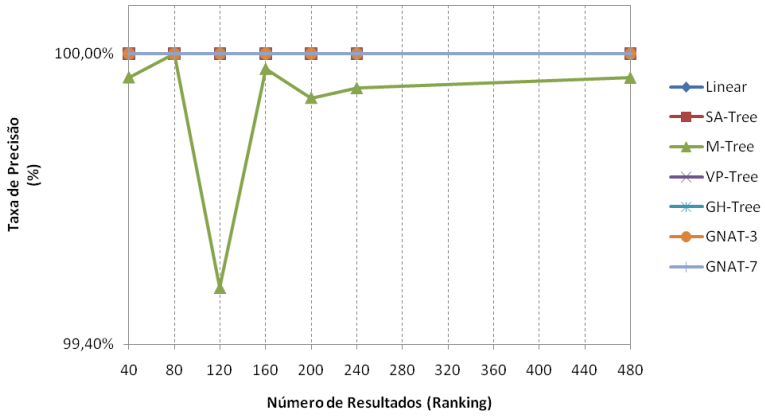

Figura 47 – Precisão dos resultados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica de *Manhattan* sobre espaço de cor *HSV* (432 dimensões).

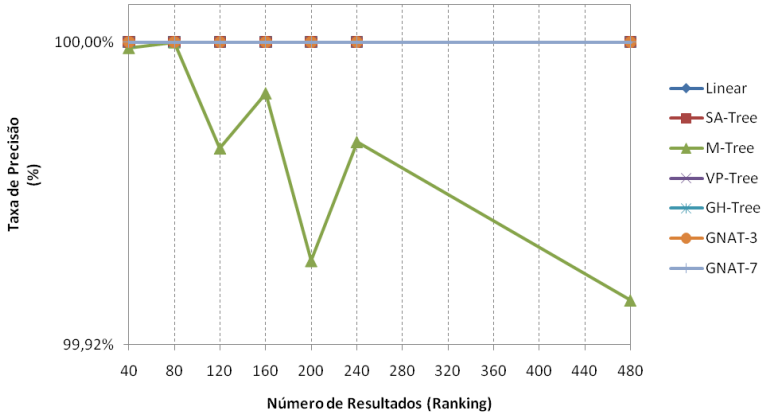

Figura 48 – Precisão dos resultados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica *Euclidiana* sobre espaço de cor *HSV* (432 dimensões).

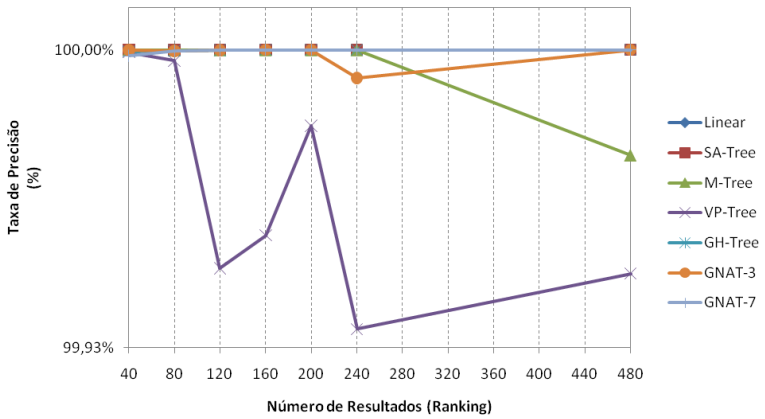

Figura 49 – Precisão dos resultados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica do *Co-seno* sobre espaço de cor *HSV* (432 dimensões).

forma, o tempo que requerido para efectuar esses cálculos é eliminado da equação.

Apesar da *SA-Tree* exibir maior relevo, outras estruturas que apresentam igualmente bons resultados. O comportamento demonstrado nos gráficos é bastante linear, com poucas oscilações, à medida que o comprimento do *ranking* aumenta. Tais resultados são indicadores de duas circunstâncias. Em primeiro lugar, o conjunto de *pivots* escolhidos para a construção das várias estruturas foi o mais indicado, na medida em que conseguem distribuir o espaço eficientemente. Por outro lado, esta distribuição reflecte-se nas pesquisas, ao inferir sobre as *queires* raios de actuação inferiores, garantindo obtenção de resultados idênticos aos gerados pelo índice não estruturado.

Não obstante dos bons indicadores demonstrados, a *M-Tree* é a estrutura que sobressai das demais (independentemente do tipo de métrica utilizada). Como é possível observar, a *M-Tree* produz valores bastante satisfatórios para o espaço que estamos analisar, sendo ligeiramente melhor ou igual que a própria *SA-Tree* quando o comprimento do *ranking* pretendido é elevado (480). Isto deve-se ao facto do seu índice assemelhar-se a uma *B-Tree* (as imagens estão agrupadas segundo o grau de proximidade com o *pivot*), traduzindo-se em bons desempenhos na pesquisa quando se pretende um conjunto de resultados elevado.

Finalmente, é de salientar o comportamento das *GNATs* e da *GH-Tree*. Embora na sua construção utilizarem o mesmo princípio base na determinação dos *pivots*, os resultados aqui apresentados não traduzem o comportamento semelhante que deviam de demonstrar. Na verdade, a razão deve-se à escolha dos *pivots* ser realizada de modo aleatório, não permitindo que se obtenha, na maioria dos casos, imagens que melhorem significativamente a distribuição do espaço (tornando-se natural o surgimento de situações semelhantes às identificadas nos gráficos).

5.3.3.2.2 Precisão dos Resultados

Observando os gráficos gerados (Figura 47, Figura 48 e Figura 49), verifica-se que, independentemente da métrica utilizada, a precisão obtida apresenta uma ligeira degradação dos resultados obtidos nas várias pesquisas efectuadas (mais concretamente na *M-Tree*, *VP-Tree* e *GNAT 3*).

Tais resultados derivam do facto, de pelo menos uma imagem encontrar-se numa posição diferente à indicada pela pesquisa realizada linearmente, traduzindo-se assim num pequeno factor de ruído que se encontra subjacente nos resultados produzidos.

No caso da *VP-Tree* e *GNAT 3* (*Co-seno*), a explicação para o ocorrido encontra-se no alcance atribuído às *queries* não ser suficientemente grande para a obter todas as imagens correspondentes, quando comparado com uma pesquisa efectuada sobre o índice não estruturado. Por outro lado, permite concluir-se que os *pivots* utilizados não corresponderam às melhores opções para os tipos de métricas utilizadas, resultante do processo de obtenção

dos *pivots* ser exclusivamente aleatório.

Relativamente à *M-Tree*, o facto de não identificar uma dada imagem na posição indicada, resulta do efeito dessa imagem estar associada a um *pivot* diferente ao utilizado pela pesquisa, isto é, a imagem em causa encontrar-se mais próxima do *pivot* onde está inserida, do que aquele que está a ser utilizado na pesquisa e, deste modo, não poderá ser incluída no resultado final. Assim, é possível concluir que o raio atribuído às *queries* não é suficientemente abrangente para pesquisar sobre a folha em causa, sendo para tal necessário aumentar esse valor, garantindo, assim, o alcance dos resultados pretendidos.

5.3.3.3 Pesquisa no Espaço de Textura Gabor (48 dimensões)

Do mesmo modo que sucedeu na análise anteriormente realizada, os resultados que irão ser apresentados, nos diversos gráficos, resultam de valores médios obtidos dos tempos de pesquisa e dos valores do *Coeficientes de Pearson*.

5.3.3.3.1 Tempo de Pesquisa

Analisando os gráficos apresentados (Figura 50, Figura 51 e Figura 52), conclui-se que existem poucas diferenças entre estes resultados e os obtidos sobre o espaço de cor *HSV*. No entanto, existem algumas questões que necessitam uma análise mais cuidada.

Em primeiro lugar e como seria de esperar, o tempo de pesquisa na *SA-Tree* não é influenciado pelo tipo de descritor ou métrica utilizada, uma vez que as distâncias foram previamente determinadas na fase de indexação.

Não obstante desta verificação, a *M-Tree* continua apresentar o mesmo comportamento apesar da alteração do descritor, traduzindo a eficiência que esta estrutura demonstra face a diferentes tipos de descritores. Contudo, a diferença encontra-se no facto dos tempos de pesquisa serem inferiores, resultante da dimensão dos descritores.

Por outro lado, de referir uma alteração nos comportamentos de algumas estruturas, nomeadamente as *GNATs* e a *GH-Tree*. Como foi referido durante análise do sobre o espaço de cor, tanto a *GNAT* como a *GH-Tree* tinham o mesmo princípio de construção dos seus índices e por esse facto deveriam de apresentar o mesmo comportamento. Como é possível observar essa situação está a ocorrer, devendo-se sobretudo ao tipo de descritor utilizado. Sendo o *Gabor* um descritor pertencente ao espaço de textura, o valor das distâncias entre as imagens vai ser bastante maior do que estivéssemos a trabalhar, por exemplo, sobre o espaço de cores, uma vez que dificilmente as imagens apresentam o mesmo tipo de textura (logo o grau de semelhança será menor). Assim, é necessário recorrer a *queries* com um raio de actuação elevado para garantir a obtenção de todos os resultados aos igualmente gerados por uma pesquisa sobre o índice não estruturado. Deste modo, torna-se natural que os valores aqui apresentados sejam superiores às restantes estruturas e que se aproximem dos resultados

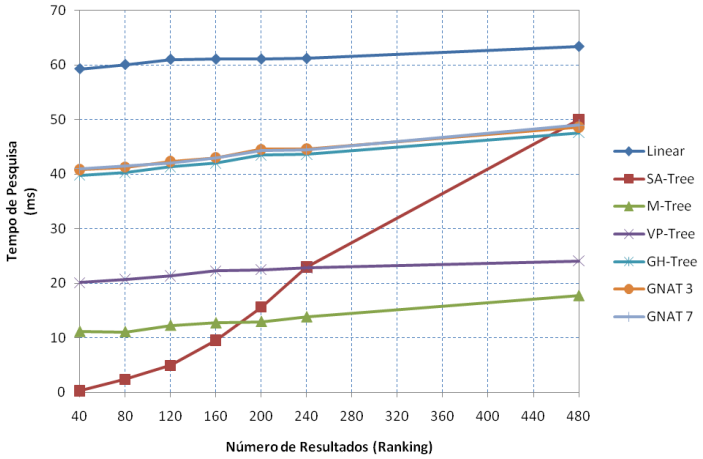

Figura 50 – Tempos de pesquisa obtidos sobre o espaço de textura *Gabor* (48 dimensões), recorrendo à métrica de *Manhattan*.

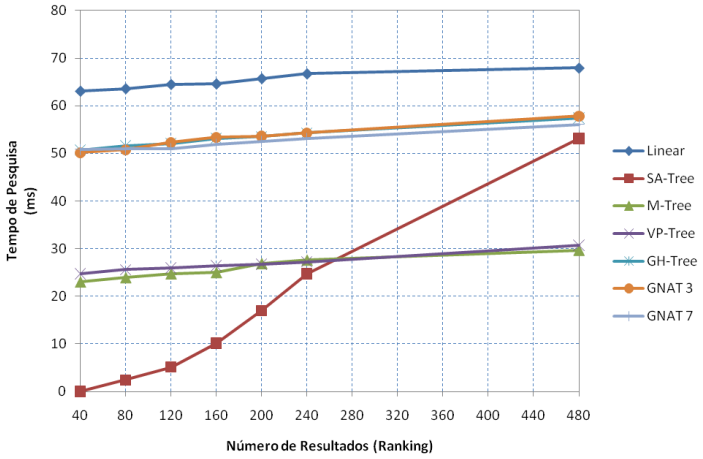

Figura 51 – Tempos de pesquisa obtidos sobre o espaço de textura *Gabor* (48 dimensões), recorrendo à métrica *Euclidiana*.

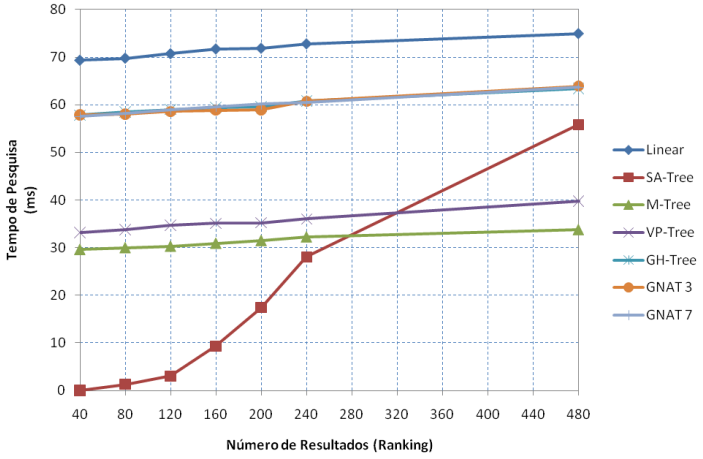

Figura 52 – Tempos de pesquisa obtidos sobre o espaço de textura *Gabor* (48 dimensões), utilizando a métrica do *Co-seno*.

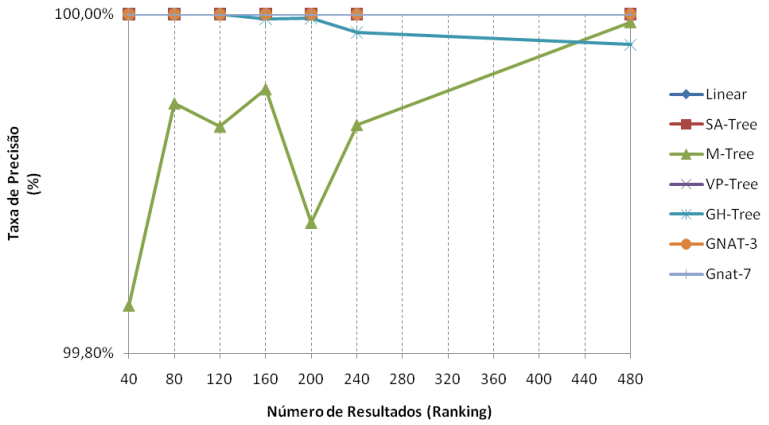

Figura 53 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica de *Manhattan* sobre espaço de textura *Gabor* (48 dimensões).

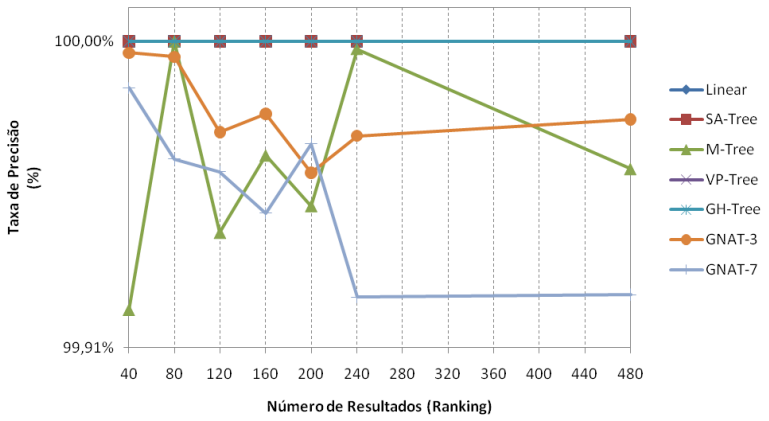

Figura 54 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica *Euclidiana* sobre espaço de textura *Gabor* (48 dimensões).

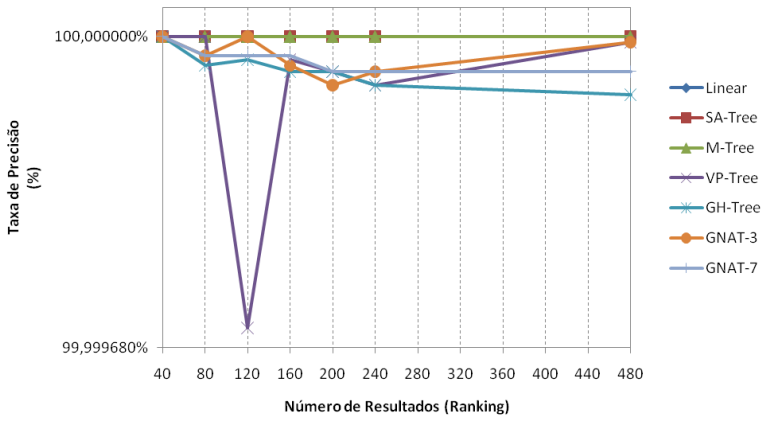

Figura 55 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica do *Co-seno* sobre espaço de textura *Gabor* (48 dimensões).

obtidos pelo índice não estruturado.

Finalmente, de salientar o comportamento da *VP-Tree*. Contrapondo ao que tinha ocorrido na análise realizada sobre o descritor *HSV*, a *VP-Tree* apresenta melhores resultados (tempos de pesquisa mais baixos) sobre o descritor *Gabor*. Para tal, este resultado deve-se à qualidade do *pivot* escolhido permitir distribuir eficazmente o espaço, tornando-se possível aferir a localização da pesquisa sobre o índice e obtendo-se pesquisas mais eficazes.

5.3.3.3.2 Precisão dos Resultados

Observando os gráficos gerados (Figura 53, Figura 54 e Figura 55) e tal como sucedeu na pesquisa sobre o espaço de cor *HSV*, verifica-se que a precisão obtida apresenta, em algunscasos, uma ligeira degradação: e.g., *M-Tree* (métricas de *Manhattan* e *Euclidiana*), *GH-Tree* (métricas de *Manhattan* e do *Co-seno*), *VP-Tree* (métrica do *Co-seno*) e *GNATs* (métrica *Euclidiana* e do *Co-seno*). Contudo, a degradação verificada é mais acentuada neste espaço.

As razões para o sucedido são idênticas às descritas no espaço de cor *HSV*, nomeadamente no que se refere à selecção dos *pivots* não ser a mais correcta e/ou a localização da imagem, no resultado final da pesquisa, encontrar-se numa posição diferente à indicada pelo o *ranking* da pesquisa linear. Deste modo, é possível inferir a existência de ruído neste espaço, impossibilitando a obtenção de todas as imagens de um modo coerente e exacto ao que é produzido pela pesquisa do índice não estruturado (assim, torna-se natural a visualização destes resultados nos gráficos aqui apresentados).

5.3.3.4 Pesquisa no Espaço de Textura Tamura (27 dimensões)

Finalmente iremos verificar o comportamento das estruturas segundo os pressupostos utilizados nas análises anteriores, mas agora para o descritor em causa.

5.3.3.4.1 Tempo de Pesquisa

Analisando os gráficos obtidos (Figura 56, Figura 57 e Figura 58), é possível verificar uma melhoria significativa dos resultados atingidos pelas estruturas face à pesquisa executada sobre o índice não estruturado, nomeadamente da *GNATs* e *GH-Tree* (comparando com o espaço de textura anterior).

Apesar do espaço que está a ser analisado corresponder a espaço de textura e das diferenças entre as imagens serem mais significativas, os resultados obtidos pelas estruturas não representam essa complexidade da pesquisa que seria de esperar. Tais factores que contribuem para o sucedido resulta na qualidade dos *pivots* gerados, uma vez que estes dispersam o espaço de modo eficiente e a dimensão dos descritores ser bastante mais reduzida do que em relação aos demais descritores analisados.

No que se refere às restantes estruturas (*SA-Tree*, *M-Tree* e *VP-Tree*), os seus comportamentos não sofreram grandes alterações face ao que foi analisado anteriormente

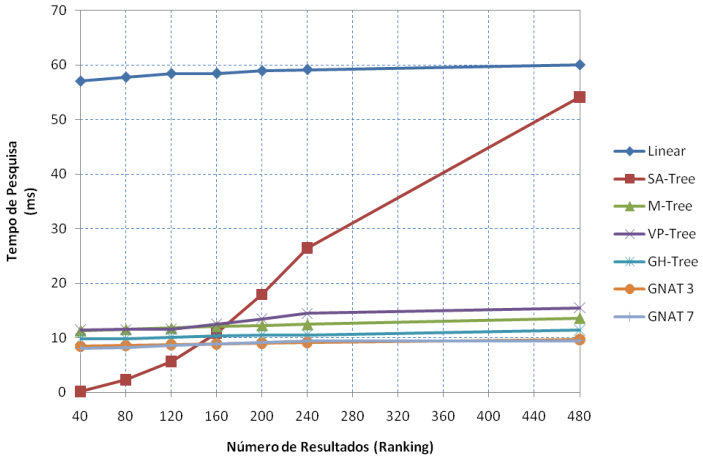

Figura 56 – Tempos de pesquisa obtidos sobre o espaço de textura *Tamura* (27 dimensões), utilizando a métrica de *Manhattan*.

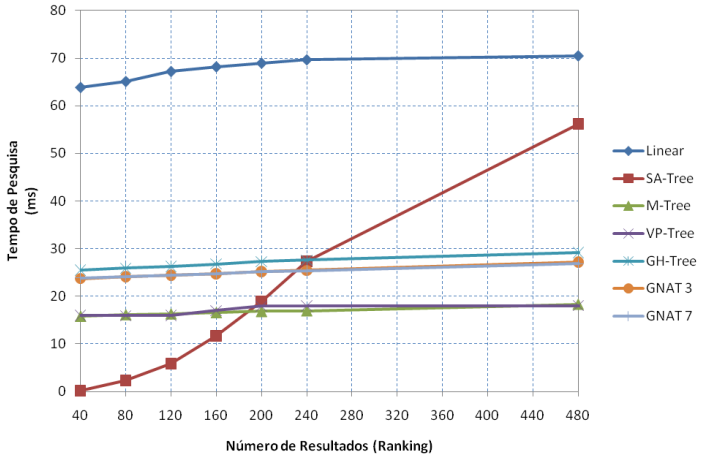

Figura 57 – Tempos de pesquisa obtidos sobre o espaço de textura *Tamura* (27 dimensões), utilizando a métrica *Euclidiana*.

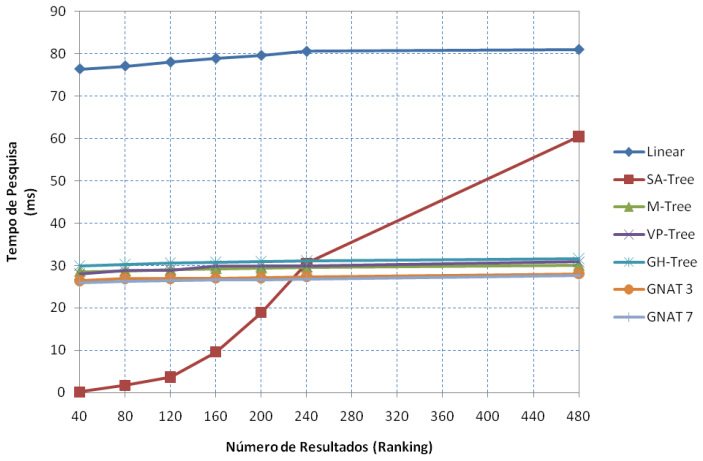

Figura 58 – Tempos de pesquisa obtidos sobre o espaço de textura *Tamura* (27 dimensões), utilizando a métrica do *Co-seno*.

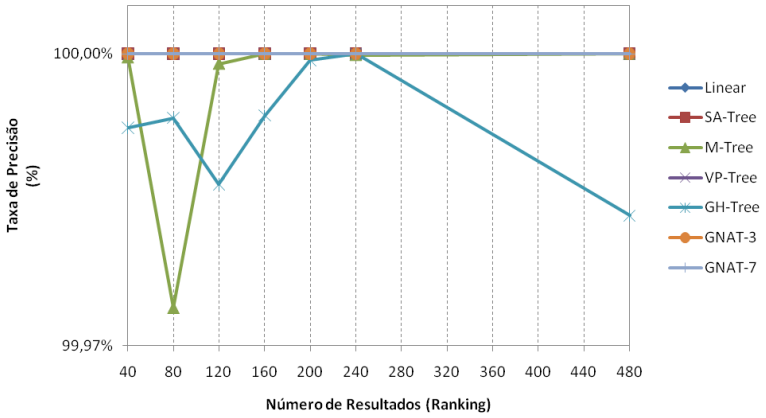

Figura 59 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica de *Manhattan* sobre espaço de textura *Tamura* (27 dimensões).

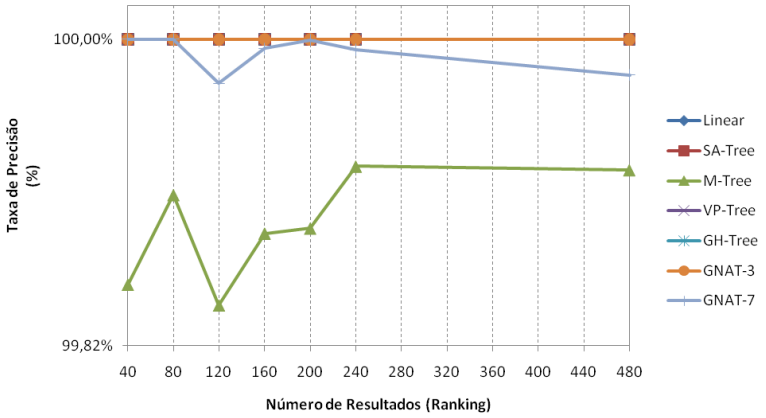

Figura 60 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica E*uclidiana* sobre espaço de textura *Tamura* (27 dimensões).

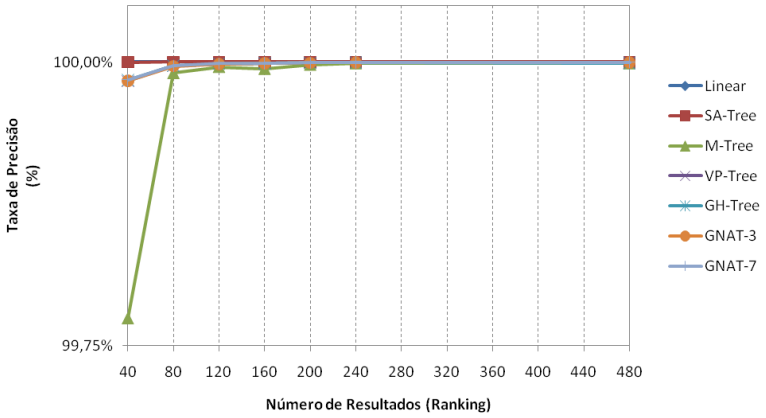

Figura 61 – Precisão dos dados obtidos em cada uma das estruturas comparativamente à procura linear, utilizando a métrica do *Co-seno* sobre espaço de textura *Tamura* (27 dimensões).

(*HSV* e *Gabor*), estando a diferença nos valores de tempos obtidos.

5.3.3.4.2 Precisão dos Resultados

Observando os gráficos gerados (Figura 59, Figura 60 e Figura 61) e comparando-os com os obtidos nos restantes descritores estudados, continua-se a verificar uma ligeira degradação na precisão. Deste modo, as conclusões alcançadas nas análises anteriormente efectuadas adequam-se para este espaço, às quais resultam na identificação da existência de ruído no espaço e que impossibilita a obtenção de uma precisão de 100% nos resultados que são produzidos pelas pesquisas.

5.3.3.5 Avaliação Global dos Resultados

Observando os resultados produzidos pelas diversas experiências efectuadas sobre este conjunto de imagens (25.000 imagens), permite-se concluir que as várias estruturas implementadas garantiram, na maioria dos casos, ganhos significativos em termos de eficácia e eficiência no processo de pesquisa, face ao índice não estruturado.

Tal como foi analisado, tanto a *SA-Tree*, como a *M-Tree* e a *VP-Tree* foram as estruturas que melhor se adaptaram aos diferentes descritores analisados, permitindo aferir a possível utilização destas estruturas, numa aplicação que congregue a gestão de multimédia nas suas funcionalidades. Contudo, existiram excepções ao que seria esperado (caso das *GNATs* e da *GH-Tree*).

Uma das possíveis explicações para o sucedido prende-se na máquina onde foram elaborados estes testes. Devido ao seu processamento e memória disponível, esta pode ser a causa provável de se obter resultados relativamente baixos para a pesquisa linear e, de certo modo, influenciar negativamente os resultados gerados pelas estruturas. Uma outra hipótese assenta no número imagens utilizadas não ser suficientemente grande para garantir os resultados pretendidos (maior eficiência das estruturas, face ao índice não estruturado). Deste modo, torna-se necessário realizar uma análise sobre um conjunto de imagens mais alargado e verificar o comportamento das estruturas face ao índice não estruturado.

5.3.4 Análise em larga escala para diferentes dimensões e imagens

Como foi mencionado anteriormente, os resultados produzidos através da utilização de um conjunto de 25.000 imagens, não permitiu obter uma conclusão assertiva do comportamento das estruturas utilizadas, face ao índice cujos dados estão dispostos linearmente. Deste modo, foi necessário introduzir um novo conjunto contendo 250.000 de imagens, ao qual permitisse verificar o comportamento desejado e identificar quais as estruturas melhor adaptadas. Assim, a análise irá centrar-se na observação do comportamento das estruturas face a variações no número de imagens no repositório, ao

mesmo tempo que se executam a variações das dimensões dos descritores, sobre o espaço de cor *HSV*.

Tal como sucedeu no estudo efectuado sobre o conjunto de 25.000 imagens, a análise deste conjunto de imagens irá utilizar a mesma máquina descrita na *secção 5.3.2*.

De referir ainda, que análise realizada aos diversos espaços consideram a realização de 100 pesquisas a partir de imagens do repositório, escolhidas aleatoriamente. Como o estudo incide numa análise de larga escala quer na variação das dimensões dos descritores, quer no número de imagens no repositório, o comprimento de *ranking* que foi tido em consideração corresponde ao valor 200. Desta forma, torna-se possível aferir com bastante exactidão o comportamento das diversas estruturas sobre o espaço procura escolhido, considerando um número elevado de resultados obtidos numa pesquisa.

5.3.4.1 Complexidade na Indexação

A análise aqui apresentada permite observar o comportamento das estruturas durante processo de indexação, avaliando segundo o aumento do número de imagens a analisar e tipo de descritor utilizado – Tabela 5, Tabela 6, Tabela 7 e Tabela 8.

Perante os resultados apresentados, nas diversas tabelas geradas, verifica-se que à medida que o número de imagens aumenta, maior será o tempo requerido para construir os índices.

Analisando pormenorizadamente, observa-se que o processo de construção da *SA-Tree* é bastante pesado (situação também identificada na Tabela 3, *secção 5.3.3.1*). Desta forma, o custo na identificação das melhores imagens, para constituir o conjunto de vizinhos, trás custos substancialmente elevados na construção do índice. Sendo os conjuntos de vizinhos obtidos através da utilização das métricas propostas, o método de construção do índice permite facilitar o processo de pesquisa, pois para além da identificação da imagem que está

Tabela 5 – Tempo de construção dos índices (descritor *HSV* de 54 dimensões), segundo o número de imagens contidas no repositório (tempo em segundos).

	50.000	**100.000**	**150.000**	**200.000**	**250.000**
Linear	n.a.	n.a.	n.a.	n.a.	n.a.
SA-Tree	4365	18368	43080	-	-
M-Tree	7	15	24	35	41
VP-Tree	6	8	12	20	27
GH-Tree	18	40	65	88	110
GNAT-3	18	40.5	67	90	112
GNAT-7	18.5	41	69	92	115

Tabela 6 – Tempo de construção dos índices (descritor *HSV* de 108 dimensões), segundo o número de imagens identificado (tempo em segundos).

	50.000	**100.000**	**150.000**	**200.000**	**250.000**
Linear	n.a.	n.a.	n.a.	n.a.	n.a.
SA-Tree	5479	24138	-	-	-
M-Tree	9	28	36	41	49
VP-Tree	7.5	11	19	27	36
GH-Tree	21	46	71.5	94	122
GNAT-3	22.5	48.5	73	99	127
GNAT-7	23	50	74.5	101	130

Tabela 7 – Tempo de construção dos índices (descritor *HSV* de 216 dimensões), segundo o número de imagens identificado (tempo em segundos).

	50.000	**100.000**	**150.000**	**200.000**	**250.000**
Linear	n.a.	n.a.	n.a.	n.a.	n.a.
SA-Tree	6845	30750	-	-	-
M-Tree	14	35	47	56	62
VP-Tree	12	20	34	48	56
GH-Tree	30	62	87	115	149
GNAT-3	34	65	90.5	119	153
GNAT-7	36.5	68.5	93	121	157

Tabela 8 - Tempo de construção dos índices (descritor *HSV* de 432 dimensões), segundo o número de imagens identificado (tempo em segundos).

	50.000	**100.000**	**150.000**	**200.000**	**250.000**
Linear	n.a.	n.a.	n.a.	n.a.	n.a.
SA-Tree	8022	-	-	-	-
M-Tree	20	49	62	78	91
VP-Tree	20	43	57	69	81
GH-Tree	48	79	108	132	170
GNAT-3	51.5	85	111.5	135	174
GNAT-7	55	87.5	114	137	179

associada, encontra-se também o valor da distância dessa imagem vizinha à imagem principal. Logo, torna-se natural a visualização dos valores como os descritos nas diversas tabelas.

Um outro ponto relativo à *SA-Tree* encontra-se nos valores que não são apresentados nas tabelas relativos aos tempos de construção dos índices. Isso fica a dever-se ao hardware utilizado não permitir aferir esses tempos, devido a limitações de memória durante a execução do programa de teste. No entanto, possibilitou concluir que o algoritmo utilizado é bastante complexo e que trás bastantes desvantagens, caso se pretenda construir uma aplicação de gestão de multimédia, cujo número de conteúdos multimédia é bastante elevado.

No que se refere às restantes estruturas, a análise elaborada na *secção 5.3.3.1* contínua a verificar-se.

No entanto, a análise sobre a complexidade de indexação não pode simplesmente terminar com o estudo relativo ao tempo de construção dos índices de cada estrutura.
Para tal, é necessário ter em consideração o espaço que cada índice ocupa em disco e de que forma o seu tamanho influencia o desempenho de uma aplicação de gestão de multimédia. Para o efeito, foram criados um conjunto de gráficos que permitem avaliar a evolução do tamanho dos ficheiros, à medida que o número de imagens no repositório aumenta – Figura 62, Figura 63, Figura 64, Figura 65, Figura 66 e Figura 67.

Observando atentamente cada um dos gráficos mencionados, verifica-se duas situações distintas. Em primeiro lugar, tanto os índices da *SA-Tree* como os índices da *VP-Tree* ocupam menos espaço em disco que as restantes estruturas e a seu tamanho não é influenciado com o aumento da dimensão dos descritores. Tal situação resulta da informação contida nesses ficheiros, apenas definir o nome das imagens, o conjunto de imagens que estão associadas e as respectivas distâncias (permitindo que o processo de pesquisa seja bastante rápido e eficiente).

Relativamente às restantes estruturas, verifica-se que o tamanho dos ficheiros que contêm a informação dos índices é influenciado, grandemente, com o aumento das imagens no repositório e com aumento progressivo das dimensões dos descritores. Isto resulta do facto,da informação contida nos ficheiros compreender os objectos das classes de cada estrutura. Assim, por meio deste tipo de serialização, o carregamento da informação é bastante maisrápido, embora apresente uma pequena desvantagem. Ao carregar os objectos contidos nos ficheiros, é necessário acrescentar um pouco de memória à máquina virtual, de modo a carregar o índice em memória e, posteriormente, efectuar pesquisas sobre os mesmos.

Analisada a complexidade no processo de indexação de conteúdo multimédia, no que se refere ao aumento do número de imagens e aplicando variações sucessivas na dimensão dos descritores, torna-se fundamental analisar o impacto que este conjunto imagens exerce sobre

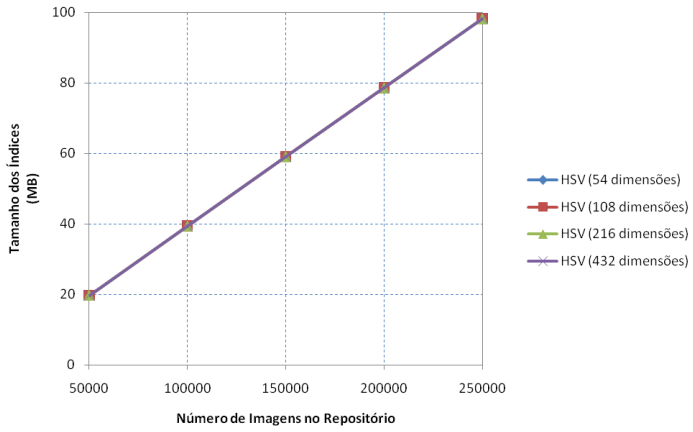

Figura 62 – Evolução dos índices da *SA-Tree*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

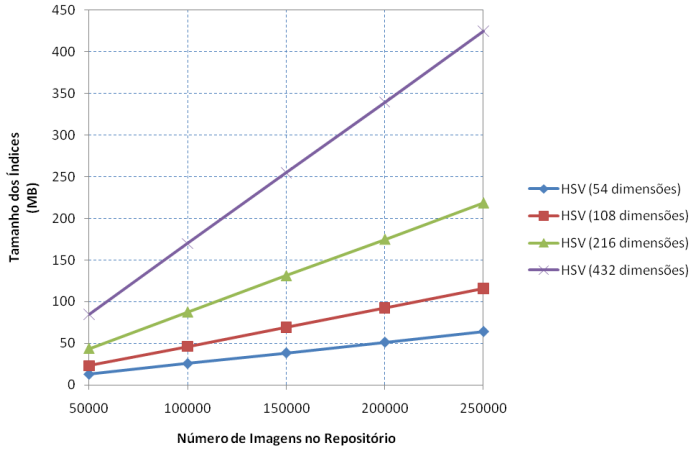

Figura 63 - Evolução dos índices da *M-Tree*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

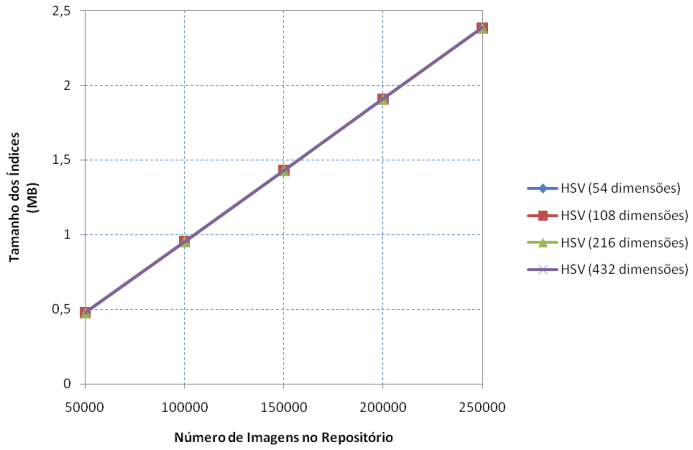

Figura 64 - Evolução dos índices da *VP-Tree*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

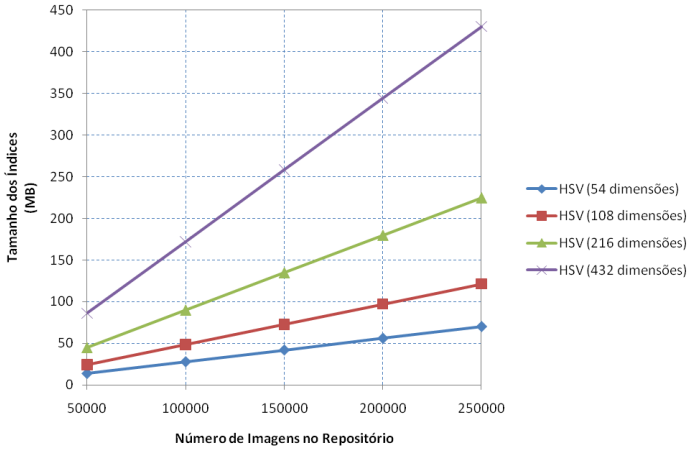

Figura 65 - Evolução dos índices da *GH-Tree*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

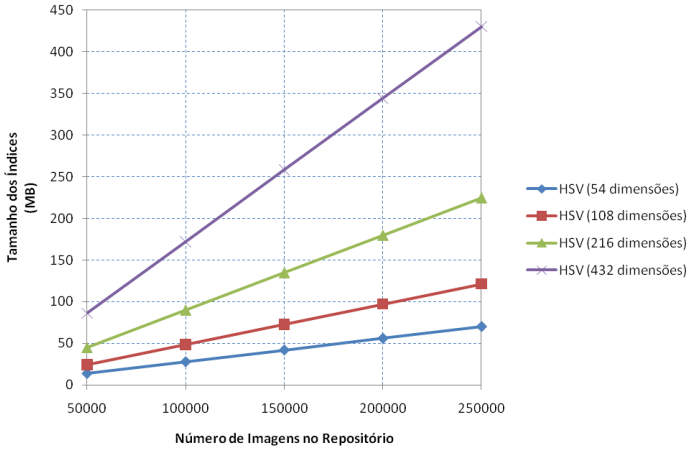

Figura 66 - Evolução dos índices da *GNAT-3*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

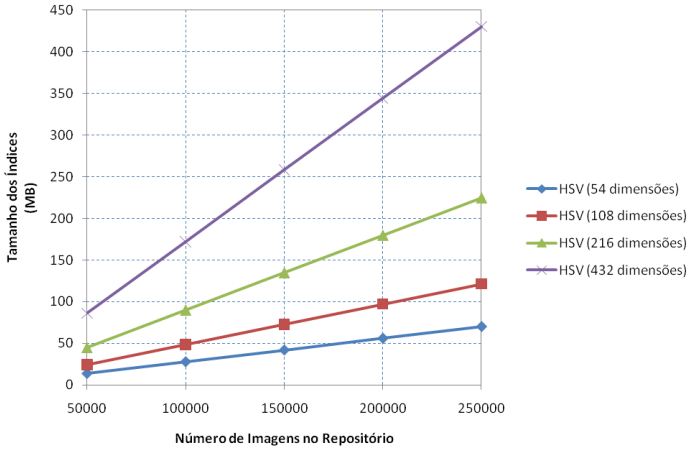

Figura 67 - Evolução dos índices da *GNAT-7*, à medida que o número de imagens no repositório e a dimensão dos descritores aumenta.

as pesquisas e os ganhos obtidos, relativamente ao índice não estruturado.

5.3.4.2 Pesquisa no Espaço de Cor HSV (54 dimensões)

A análise deste espaço, incide na verificação do tempo de pesquisa sobre o conjunto de imagens no repositório e a verificação da precisão dos dados obtidos através do *Coeficientes de Pearson*.

5.3.4.2.1 Tempo de Pesquisa

Analisando os gráficos obtidos (Figura 68, Figura 69 e Figura 70), verifica-se a existência de uma degradação do tempo pesquisa de algumas das estruturas, à medida que o número de imagens no repositório aumenta. Como é possível observar, as estruturas que se destacam, por apresentarem tal comportamento, correspondem à *GH-Tree* e às *GNATs*. Isto resulta do facto, destas estruturas obterem os seus *pivots* por meio de uma selecção aleatória de imagens, como um método de dispersar o espaço de forma eficiente. Contudo, pelos resultados produzidos, as imagens seleccionadas não corresponderam às melhores soluções, para o tipo de espaço em causa. Para tal, é necessário reconstruir os índices de modo a tentar solucionar o problema e garantir melhores tempos de pesquisa. No entanto, se o problema assenta na selecção dos *pivots*, o aumento do número de imagens no repositório dificultará ainda mais este processo, visto que a probabilidade de se obter as melhores imagens diminui drasticamente.

Embora a *GH-Tree* com as *GNATs* não apresentem uma eficiência significativa, face ao índice não estruturado, existem outras estruturas de dados que apresentam um comportamento bastante mais positivo. Uma dessas estruturas corresponde à *SA-Tree*, uma vez que independentemente do tamanho do repositório e da métrica utilizada, esta é capaz de gerar tempos de pesquisa muito baixos. Segundo o que foi identificado anteriormente (ver *secção 5.3.3*), este comportamento resulta, em grande medida, das distâncias entre imagens terem sido previamente calculadas, durante a construção do índice. Deste modo, o processo de pesquisa apenas necessita obter os valores de distâncias para determinar os resultados gerados pela mesma.

Apesar de as estruturas aqui analisadas se destacarem de forma mais ou menos positiva, não reflectem a totalidade dos resultados apresentados nos gráficos. Para tal, é necessário salientar o comportamento tanto da *VP-Tree*, como da *M-Tree*. Embora apresentem uma ligeira degradação do tempo de pesquisa, para o tipo de espaço e para o número de imagens utilizadas, estas estruturas conseguem apresentar resultados bastante satisfatórios, aos quais permite concluir que ambas as estruturas conseguem dispersar o espaço de modo eficiente, apesar da complexidade inerente à construção dos seus índices (*secção* 5.3.4.1).

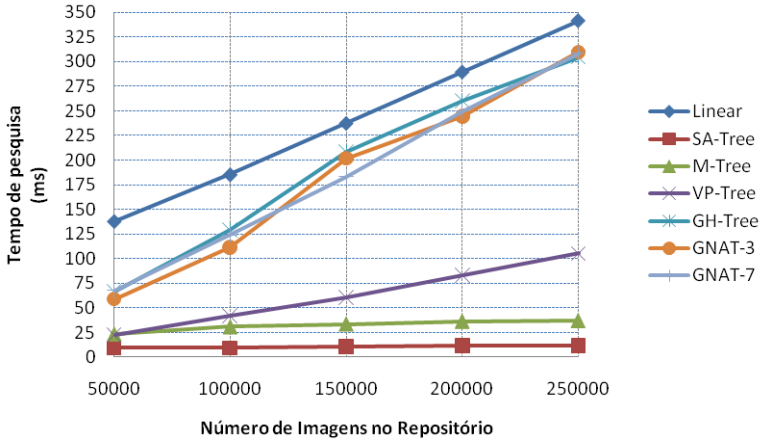

Figura 68 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (54 dimensões), recorrendo à métrica de *Manhattan*.

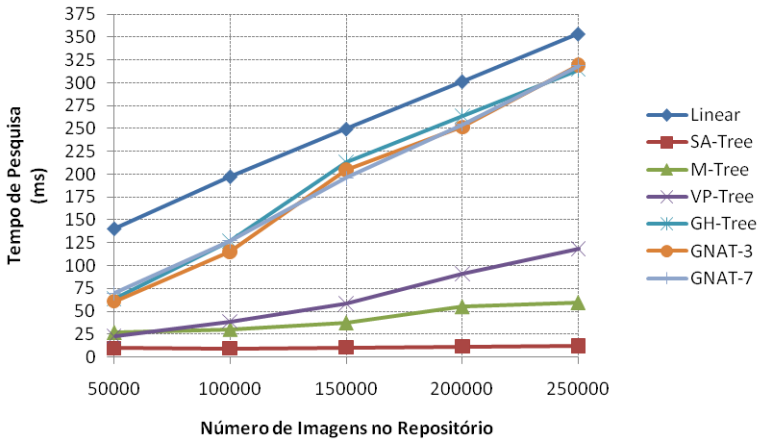

Figura 69 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (54 dimensões), recorrendo à métrica *Euclidiana*.

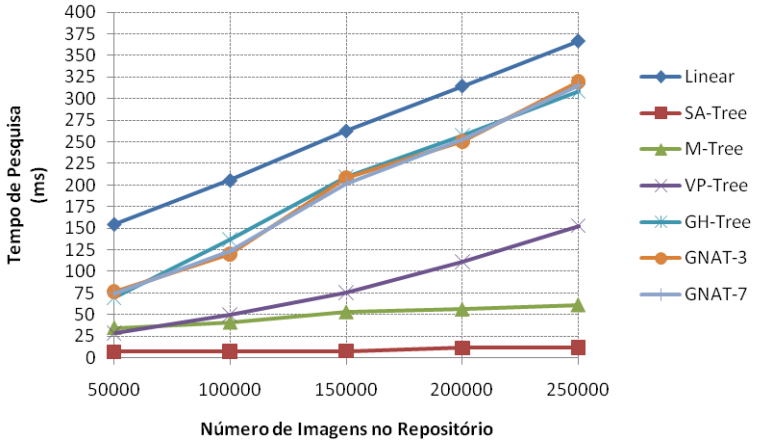

Figura 70 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (54 dimensões), recorrendo à métrica do *Co-seno*.

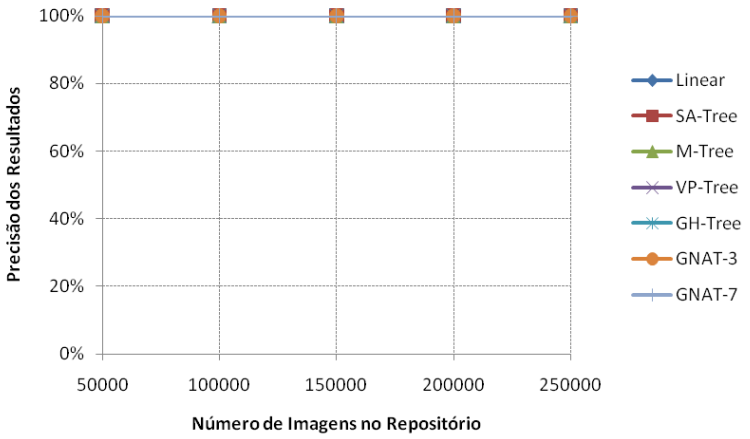

Figura 71 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Manhattan* sobre espaço de cor *HSV* (54 dimensões).

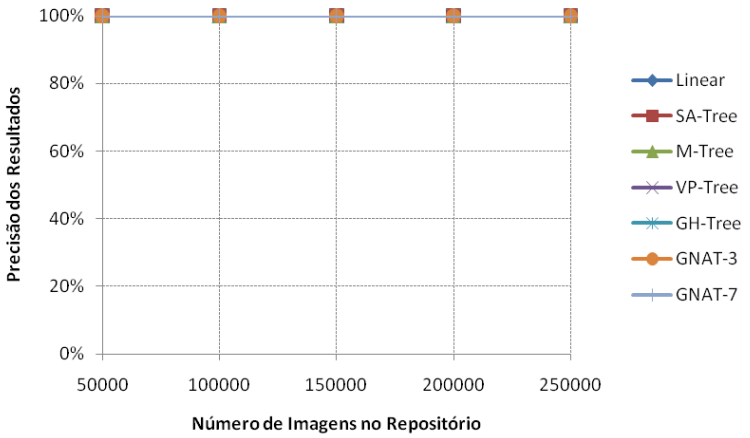

Figura 72 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Euclidiana* sobre espaço de cor *HSV* (54 dimensões).

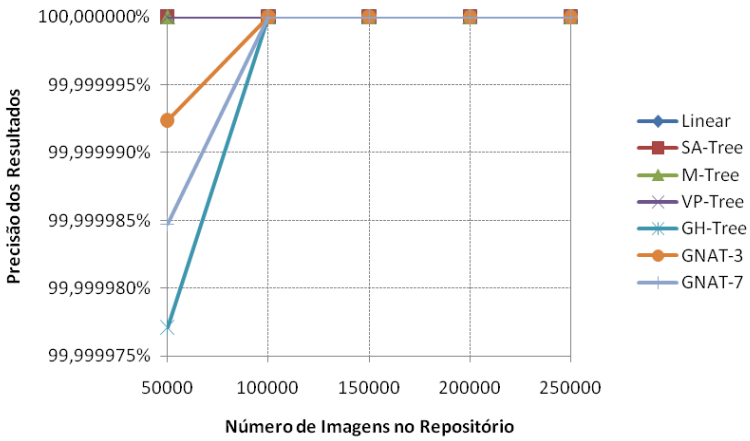

Figura 73 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Co-seno* sobre espaço de cor *HSV* (54 dimensões).

5.3.4.2.2 Precisão dos Resultados.

Os resultados gerados e apresentados nos gráficos (Figura 71, Figura 72 e Figura 73), permitem verificar que apesar dos tempos de pesquisa em algumas das estruturas serem muito elevados, estas conseguem produzir os resultados idênticos ao que são produzidos por uma pesquisa linear. Contudo, é de notar uma ligeira degradação dos resultados (Figura 73), resultante de um pequeno ruído existente no espaço, derivado de vários factores (factores esses, descritos nas análises realizadas anteriormente).

5.3.4.3 Pesquisa no Espaço de Cor HSV (108, 216 e 432 dimensões)

Analisando o comportamento das estruturas considerando os valores de dimensões aqui indicados, muito pouco é acrescentado ao que foi observado no mesmo tipo de espaço, recorrendo a descritores de dimensões mais reduzidas (54 dimensões) – ver *Anexo 2*, *Anexo 3* e *Anexo 4*. As únicas diferenças substanciais aquando do aumento das dimensões num dado espaço são: o aumento dos tempos de pesquisa quer linearmente, quer ao nível das estruturas, sem haver grandes alterações na disposição indicada nos gráficos; e ao nível da precisão dos dados, com o surgimento de pequenas situações de degradação resultantes da existência de algum ruído que possa existir no espaço.

Por outro lado, é de ainda salientar que o comportamento da *SA-Tree* não sofre qualquer tipo de alteração, uma vez que o valor das distâncias entre imagens já se encontra calculado, sendo apenas é necessário percorrer as imagens e retirar o valor de distância associado, obtendo-se o *ranking* correspondente.

5.3.4.4 Avaliação Global dos Resultados

Observando o comportamento das estruturas no espaço considerado, em conjunto com o número de imagens e dimensões dos descritores consideradas, verifica-se o grande desempenho da *SA-Tree* no processo de pesquisa. Contudo, a sua eficiência acarreta consequências na construção do índice. Como foi identificado, todas as imagens necessitam de conhecer os seus nós vizinhos de modo a construir o grafo sem ciclos (*Diagrama de Delaunay*) e, desta forma, é necessário percorrer todo o repositório. Devido a esse facto e como vimos nas várias tabelas (Tabela 5, Tabela 6, Tabela 7 e Tabela 8), torna-se impraticável utilizar esta estrutura quando o número de imagens no repositório aumenta, na medida em que acarreta consequências significativas da complexidade temporal na construção dos índices (apresenta um crescimento exponencial), constituindo-se no *Calcanhar de Aquiles* desta estrutura.

Por outro lado, é de salientar o comportamento da *M-Tree* e da *VP-Tree*. Embora apresentem diferentes complexidades na construção dos seus índices, estas demonstram um bom desempenho à medida que o número de imagens aumenta, indicativo da dispersão

eficaz e equilibrada do espaço sem que prejudique o tempo de pesquisa e a precisão dos resultados produzidos. Assim, estas estruturas constituem numa mais-valia para uma possível utilização numa aplicação de gestão de conteúdos multimédia.

5.4 Sumário

Em síntese, das várias estruturas aqui analisadas sobre os diversos espaços de pesquisa, verificando o número de imagens no repositório e diferentes valores de dimensões dos descritores, é possível inferir que tanto a *M-Tree*, como a *SA-Tree* e a *VP-Tree* correspondem nas melhores soluções a serem utilizadas na aplicação desenvolvida nesta dissertação. Contudo, devido ao facto da existência de uma complexidade temporal bastante elevada, por parte da *SA-Tree* (Tabela 3, Tabela 5, Tabela 6, Tabela 7 e Tabela 8), a sua aplicação num sistema de larga escala de gestão de conteúdos multimédia, torna-se inviável.

Do conjunto das 3 estruturas referidas, a *M-Tree* constitui na melhor solução. Ao apresentar um carácter dinâmico na construção dos índices, não necessita de os reconstruir na totalidade de modo a manter a eficiência desejada, sempre que uma nova imagem é inserida no repositório. Por outro lado, ao contrario das restantes estruturas, não necessita de identificar previamente os seus *pivots* de forma a dispersar correctamente o espaço (situação essa, que acarreta alguns problemas, uma vez que é possível obter *pivots* de fraca qualidade devido ao processo de selecção ser aleatório).

6

Conclusões e Trabalho Futuro

Esta dissertação descreveu um sistema para indexar e pesquisar informação textual e multimédia, tendo sido analisado o comportamento das estruturas de indexação para um conjunto de 250.000 imagens. Este estudo permitiu apurar quais são os melhores métodos para solucionar os problemas que advêm de pesquisas em espaços de alta dimensão. A título de exemplo foi igualmente desenvolvida uma aplicação de pesquisa multimédia como demonstração das tecnologias implementadas. Posto isto, iremos agora discutir quais são os principais contributos que resultam do trabalho realizado nesta dissertação, assim como propostas de melhoramentos ou extensões que ajudem a aperfeiçoar o sistema de gestão de multimédia implementado.

6.1 Contribuições

A principal contribuição desta dissertação foi o desenvolvimento do serviço de indexação e pesquisa de multimédia, de um modo rápido e eficiente. Para tal, desenvolveu-se um conjunto de serviços de indexação (de texto e de imagens) que permitissem a realização das pesquisas pretendidas e que apresentassem o mesmo grau de desempenho. Estes serviços tentam ser os mais próximos possíveis um do outro, embora trabalhem sobre conteúdos bastante diferentes. No entanto, os seus desempenhos e funcionalidades são bastante próximos, às quais podem ser observadas na interface de administração de cada serviço (muito contribui-o a utilização da *framework Restlet* no desenvolvimento do serviço de indexação de multimédia – ver Capítulo 4).

Por último, uma contribuição fundamental consiste na análise rigorosa e precisa sobre as estruturas implementadas, ao nível dos diversos espaços de procura considerados. Através deste estudo, foi possível identificar os problemas inerentes à utilização deste tipo de descritores e verificar quais as estruturas melhor adaptadas para o tipo de dados considerado.

6.2 Conclusões

O objectivo principal desta dissertação consistia no desenvolvimento de um sistema que permitisse a pesquisa de conteúdos multimédia. Através da utilização de várias *frameworks* de desenvolvimento (*GWT*, *SOLR* e *Restlet*), foi possível alcançar esse objectivo com um sistema real, possibilitando, não só, a realização de pesquisas textuais, como também efectuar pesquisas por exemplo (seleccionando uma imagem do repositório).

Por outro lado, a fiabilidade e a eficácia da aplicação foi conseguida recorrendo à selecção da estrutura melhor adaptada ao tipo de dados utilizados. Para tal, esta escolha resulta da análise pormenorizada do comportamento das estruturas (tempo de construção dos índices, espaço ocupado pelos índices em disco, tempo de pesquisa e precisão dos resultados obtidos nas pesquisas), face ao número de imagens no repositório, dimensão dos descritores e métricas utilizadas – ver as *secções 5.3.3* e *5.3.4*. A conjugação destes factores torna a solução proposta bastante robusta, uma vez que compreende todos os pontos fundamentais, caso se pretenda transpor este sistema para o domínio público.

6.3 Trabalho Futuro

Embora o objectivo proposto tenha sido alcançado, muito pode ainda ser realizado para aprofundar o trabalho efectuado nesta dissertação. Deste modo, as possíveis áreas de investigação que puderão contribuir na evolução do estudo realizado são:

- Análise da problemática relativa à inserção e remoção de imagens do repositório, estando o índice definido – actualização dos índices.
- Propor de novas soluções que melhorarem os processos de construção dos índices, referentes às estruturas analisadas.
- Integração de um sistema de cálculo de *features* na aplicação desenvolvida.
- Extensão do serviço de indexação de multimédia para permitir a utilização de outro tipo de conteúdos (e.g., vídeos).
- Extensão da aplicação de gestão de multimédia para possibilitar pesquisa por exemplos, recorrendo à elaboração de esboços produzidos pelos utilizadores.
- Propor uma solução de descentralização da informação existente no serviço de multimédia, garantindo a tolerância a falhas e que sustente eficiência e rapidez pretendida em aplicações de gestão de conteúdos multimédia.

Referências

Beckmann, N., H.-P. Kriegel, et al. (1990). "The R*-tree: an efficient and robust access method for points and rectangles." SIGMOD Rec. 19(2): 322-331.

Berchtold, S., D. A. Keim, et al. (1996). The X-tree: An Index Structure for High-Dimensional Data. Proceedings of the 22th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.

Böhm, C., S. Berchtold, et al. (2001). "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases." ACM Comput. Surv. 33(3): 322-373.

Brin, S. (1995). Near Neighbor Search in Large Metric Spaces. Proceedings of the 21th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.

Burkhard, W. A. and R. M. Keller (1973). "Some approaches to best-match file searching." Commun. ACM 16(4): 230-236.

C. Traina, J., A. Traina, et al. (2002). "Fast Indexing and Visualization of Metric Data Sets using Slim-Trees." IEEE Trans. on Knowl. and Data Eng. 14(2): 244-260.

Chávez, E., G. Navarro, et al. (2001). "Searching in metric spaces." ACM Comput. Surv. 33(3): 273-321.

Ciaccia, P. and M. Patella (2002). "Searching in metric spaces with user-defined and approximate distances." ACM Trans. Database Syst. 27(4): 398-437.

Ciaccia, P., M. Patella, et al. (1997). M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.

Dehne, F. and H. Noltemeier (1987). "Vorono trees and clustering problems." Inf. Syst. 12(2): 171-175.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures, University of California, Irvine: 162.

Flickner, M., H. Sawhney, et al. (1995). Query by image and video content: the QBIC system.

Fukunaga, K. and P. M. Narendra (1975). "A Branch and Bound Algorithm for Computing k-Nearest Neighbors." Computers, IEEE Transactions on C-24(7): 750-753.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. Proceedings of the 1984 ACM SIGMOD international conference on Management of data. Boston, Massachusetts, ACM.

Hjaltason, G. R. and H. Samet (2000). Incremental similarity search in multimedia databases. Computer Science Department TR-4199, Univ. Maryland, College Park, Md., Nov. .

Hjaltason, G. R. and H. Samet (2003). "Index-driven similarity search in metric spaces (Survey Article)." ACM Trans. Database Syst. 28(4): 517-580.

Howarth, P. and S. M. Rüger (2005). Fractional distance measures for content-based image retrieval. European Conference on Information Retrieval. Santiago de Compostela, Spain.

Kalantari, I. and G. McDonald (1983). "A Data Structure and an Algorithm for the Nearest Point Problem." IEEE Trans. Softw. Eng. 9(5): 631-634.

Katayama, N. and S. i. Satoh (1997). The SR-tree: an index structure for high-dimensional nearest neighbor queries. Proceedings of the 1997 ACM SIGMOD international conference on Management of data. Tucson, Arizona, United States, ACM.

Merkwirth, C., U. Parlitz, et al. (2000). "Fast nearest-neighbor searching for nonlinear signal processing." Physical Review E 62(2): 2089.

Navarro, G. (1999). Searching in Metric Spaces by Spatial Approximation. Proceedings of the String Processing and Information Retrieval Symposium \& International Workshop on Groupware, IEEE Computer Society.

Navarro, G. (2002). "Searching in metric spaces by spatial approximation." The VLDB Journal 11(1): 28-46.

Navarro, G. and N. Reyes (2002). Fully Dynamic Spatial Approximation Trees. Proceedings of the 9th International Symposium on String Processing and Information Retrieval, Springer-Verlag.

Sellis, T. K., N. Roussopoulos, et al. (1987). The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. Proceedings of the 13th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc.

Stonebraker, M., T. Sellis, et al. (1986). An Analysis of Rule Indexing Implementations in Data Base Systems, EECS Department, University of California, Berkeley.

Tamura, H., S. Mori, et al. (1978). "Textural Features Corresponding to Visual Perception." Systems, Man and Cybernetics, IEEE Transactions on 8(6): 460-473.

Town, C. and D. Sinclair (2001). Ontological Que y Language for Content Based Image Retrieval. Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'01), IEEE Computer Society.

Uhlmann, J. K. (1991). "Satisfying general proximity/similarity queries with metric trees." Information Processing Letters 40 4: 175-179.

White, D. A. and R. Jain (1996). Similarity Indexing with the SS-tree. Proceedings of the Twelfth International Conference on Data Engineering, IEEE Computer Society.

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. Austin, Texas, United States, Society for Industrial and Applied Mathematics.

Zhang, D., A. Wong, et al. (2000). "Content-based Image Retrieval Using Gabor Texture Features." IEEE Transactions PAMI: 13-15.

Anexo 1
Estruturas de Indexação sobre o Espaço Particionado

Nesta secção da dissertação irão ser incluídas algumas das estruturas analisadas durante a fase de preparação. Estas constituem em variantes das estruturas escolhidas, não acrescentando muito relevo para o trabalho realizado. No entanto, apresentam a relevância necessária, uma vez que permitiram definir quais as estruturas de maior interesse, consoante as características apresentadas.

Bisector Trees e MB-Trees

Bisector Trees ou BST's

A *BST* ou *Bisector Tree* (Kalantari and McDonald 1983), resultam de um melhoramento das *GH-Trees*, por meio da inclusão da distância máxima a um objecto em cada *pivot* da sub-árvore (*covering balls*[48]).

O objectivo, é permitir melhorar a velocidade das pesquisas, uma vez que elimina os objectos cujas as *covering balls* estão mais afastadas da *query*, do que em relação ao objecto vizinho mais próximo ou por estarem fora do alcance da *query*. Podemos verificar que, à medida que as *covering balls* aumentam, elimina-se menos objectos na pesquisa. Por outro lado, o algoritmo *search hierarchy* é descendente, uma vez que as *covering balls* diminuem a sua dimensão à medida que se desce na árvore, eliminando um maior número de caminhos. Contudo, os raios das *covering balls* dos filhos não são necessariamente mais pequenos do que os dos seus antecessores. A Figura 74 exemplifica essa situação.

[48] Consistem no alcance máximo, que um dado *pivot* consegue abranger todos os objectos que lhe estão associados.

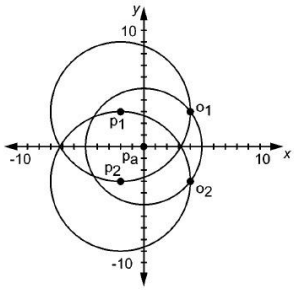

Figura 74 - (Hjaltason and Samet 2003).

Analisando a figura em concreto, verificamos que o raio das *covering balls* em torno dos *pivots* p_1 e p_2 no elemento e_a é maior do que no raio das *covering balls* em torno do *pivot* p_a no elemento antecessor e de e_a.

(Dehne and Noltemeier 1987), identificaram este tipo de comportamento e classificaram os filhos da *BST* como "*excêntricos*". No entanto, este comportamento constitui uma desvantagem, uma vez que ao agregarem um maior número de objectos, dificulta o processo de eliminação de caminhos à medida que se desce na árvore. Devido a este facto, foi proposto uma modificação das *BST's*, onde um dos dois *pivots* de cada nó (excluindo as folhas e a raiz), pode ser herdado do seu nó antecessor (o critério de escolha seria o pivot mais próximo dos objectos da sub-árvore). Assim, tornam-se necessários poucos *pivots*, melhorando o número de distâncias a calcular durante a pesquisa. Contudo, piora o custo de particionamento e aumenta a profundidade da árvore (e. g., caso a decomposição terminar quando cada folha da árvore só contiver um objecto, para além de guardar o raio das *covering balls* em torno dos *pivots*), resultando nas árvores *MB-Trees* ou *Monotonuos Bisector Tree*.

MB-Trees

O intuito destas árvores é armazenar métricas de *Minkonwski* e de dados pontuais, podendo por vezes serem usadas para métricas arbitrárias. Outras extensões permitem guardar objectos mais complexos (linhas e polígonos).

Na figura que se segue, podemos ver a representação deste tipo de árvores:

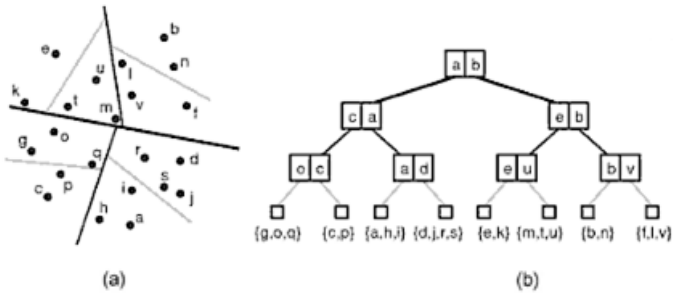

Figura 75 - Exemplo de uma representação de uma *MB-Tree*, para um dado conjunto de pontos: Representação espacial (a); Representação em árvore (b) (Hjaltason and Samet 2003).

Neste exemplo, o particionamento produzido pela *MB-Tree* é idêntico quer em termos de representação espacial, quer em árvore (tem o mesmo número de níveis), ao que é definido para as *GH-Trees*. Ao analisarmos a árvore, verifica-se que cada nó é constituído por dois pontos e, à medida que descemos na árvore, esses pontos repetem-se no nível seguinte. No entanto, por questões de optimização de espaço, a informação dos nós é guardada nas folhas, ficando os nós com apontadores para esses pontos.

Existem muitas configurações possíveis, uma vez que existem múltiplas possibilidades de escolher os *pivots* a cada passo da decomposição. A construção destas árvores segue a estratégia utilizada nas *GH-Trees* (cada folha tem o mesmo número de objectos). Por outro lado, se fosse levada a decomposição do espaço ao máximo, levaria com que as folhas apenas pudessem conter um objecto, fazendo com que a *MB-Tree* tivesse maior profundidade do que a *GH-Tree*, ou seja, para *N* elementos, a *MB-Tree* totalmente decomposta iria necessitar de $N-1$ nós, enquanto que, a *GH-Tree* apenas necessitaria $N/2$ nós (cada nó contém dois objectos).

Ao partilhar o *pivot* do antecessor, retira a excentricidade dos filhos reflectindo-se positivamente no raio das *covering balls*, uma vez que a distância da *query* a um elemento tem de ser maior ou igual à distância da *query* ao seu nó antecessor (formando uma hierarquia confinada).

Este tipo de hierarquia é impossível de existir quando os filhos da árvore são excêntricos, mas também poderá ser impossível acontecer quando não existe excentricidade. Na figura que se segue (Figura 76), explica essa situação:

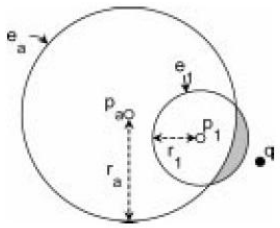

Figura 76 - Exemplo ao qual a partilha do antecessor não favorece a pesquisa (Hjaltason and Samet 2003).

Como podemos verificar, a *covering ball* da e_1 não está totalmente contida em e_a.

(Merkwirth, Parlitz et al. 2000), ignora o conceito de excentricidade e acentua o uso da hierarquia confinada, colocando o limite inferior na distância máxima que existe entre a *query* e o filho e a distância da *query* ao pai. Para tal, é utilizada uma *BST*.

Foi demonstrado que na procura do vizinho mais próximo da *query*, por meio do algoritmo *best-first* e o facto das *covering balls* dos filhos não conterem todos os objectos que não se encontram nas *covering balls* dos seus antecessores, significa que a distância ao pai tem de ser tomada em conta no cálculo da distância da *query* ao elemento, o que não irá resultar

num maior número de objectos ou nós a eliminar durante a pesquisa.

O único requisito necessário são de as distâncias, dos elementos que não são objectos, terem de conter, no seu limite inferior, as distâncias dos objectos.

Outros métodos relacionados com o Generalized Hyperplane Partitioning

A *GH-Tree* e o *GNAT* são considerados como casos especiais de uma classe generalizada de *métodos de clustering hierárquico* (Burkhard and Keller 1973; Fukunaga and Narendra 1975).

(Burkhard and Keller 1973) também descreveram um *método de clustering*, onde cada cluster é designado por "*clique*". Trata-se de um conjunto de objectos onde a distância maior, entre dois objectos desse conjunto, não é superior a um determinado valor *D*. Esta propriedade reduz o número de distâncias necessárias a calcular e permite maior eliminação durante a pesquisa, em detrimento do custo de pré-processamento.

O modo como as hierarquias são descritas, implica que a construção seja feita de cima para baixo, podendo também ser realizada de baixo para cima. No caso de a construção ser de baixo para cima, o algoritmo assume que todos os objectos foram processados por meio de um *método de clustering*, obtendo-se k clusters com k centros. Estes clusters são processados aplicando o método de *clustering hierárquico* (ex: *R-Tree*, *R*-Tree*, *SS-Tree*, *balltree* e *sphere tree*, estruturas às quais, algumas delas serão analisadas mais tarde).

R*-Tree

A *R*-Tree* (Beckmann, Kriegel et al. 1990) consiste numa extensão da *R-Tree*, tem como base optimização de pequenos volumes nas regiões de páginas criadas. Os objectivos para garantir essa optimização consiste em: **1)** minimizar a sobreposição entre regiões de páginas; **2)** minimizar a superfície das regiões de páginas; **3)** minimizar o volume que é abrangido pelos nós internos; **4)** maximizar o espaço utilizado.

Na inserção, a única alteração a efectuar dá-se ao nível do terceiro caso definido na *R-Tree*, ou seja, quando um ponto não pertence a nenhuma das regiões da página, é necessário identificar se a página filho é do *tipo data* ou do *tipo directoria*. Caso seja do *tipo data*, então é escolhida a região que minimiza o alargamento durante o processo de sobreposição (em caso de empate considera-se também o volume total). Caso o nó filho seja uma *directoria*, então

considera-se a região que irá produzir menor expansão (em caso de dúvida, o volume decide).

Utilizando o *Algoritmo de Greene*, a heurística de divisão é constituída por várias fases. A primeira consiste em definir a tamanho da divisão segundo vários critérios:

- Para cada dimensão/componente, os objectos são ordenados de acordo com o seu limite inferior e superior.
- Determina-se um número de particionamentos, como um grau de controlo assimétrico.
- Para cada dimensão, as áreas de superfície das *MBR* de todos os particionamentos, é somada e o valor mais pequeno determina a dimensão a dividir.

Na segunda fase, o plano de divisão é determinado minimizando a sobreposição entre as regiões de páginas (caso não se consiga determiná-lo, escolhe-se o plano que apresente menos espaço livre).

As divisões podem ser evitadas, caso seja aplicado um processo de *forced reinsert*. Quando ocorre um *overflow*[49], é definida uma percentagem de objectos que se encontram mais distantes em relação ao centro da região, excluindo-os desse nó. Uma vez excluídos, a região é adaptada e pode-se proceder a inserção desses objectos. Desta forma, garante-se ganhos significativos no espaço utilizado e melhora o particionamento qualitativamente, uma vez que corrige as situações como foram descritas na *R-Tree*. Contudo, este tipo de algoritmo prejudica as directorias, pois é necessário carregar todo o índice de modo a processar as *queries*, tornando este processo ineficiente.

A heurística utilizada na *R*-Tree* permite que se realizem divisões, optimizando as regiões de páginas com uma superfície pequena e, deste modo, torna as pesquisas de *range query* e *nearest-neighbor queries* mais eficientes.

R+-Tree

A *R+-Tree* (Stonebraker, Sellis et al. 1986; Sellis, Roussopoulos et al. 1987) é uma outra variante da *R-Tree* que pretende eliminar a sobreposição de regiões. Para tal, o algoritmo de

[49] Ocorre quando uma região de página está totalmente preenchida e pretende-se inserir um objecto nessa região.

divisão é modificado por meio de uma estratégia *forced-split*. As páginas do filho que dificultam a divisão livre, na ocorrência de sobreposição, são divididas ao meio na posição mais indicada para o efeito. Contudo, estas divisões poderão ser propagadas até que seja atingido o nível da página de dados, aumentando exponencialmente este número de nível para nível. No limite, existirá uma página por ponto, diminuindo a utilização do espaço em todo o índice.

X-Tree

A *X-Tree* (Berchtold, Keim et al. 1996) é uma variante da *R*-Tree*, melhorando o comportamento da *R*-Tree* ao nível de dois conceitos:

- A divisão livre, em caso de sobreposição, está dependente do histórico das divisões.
- Os *super nós* têm uma maior capacidade de páginas.

Se o histórico das divisões de páginas é registado numa estrutura semelhante a uma *R-Tree*, o resultado obtido será uma árvore binária. O processo inicia-se com uma única página que abrange, praticamente, a totalidade do espaço de dados. Na ocorrência de *overflow*, o índice divide a página em duas. Por sua vez, caso ocorra um novo *overflow*, então divide-se a página em outras duas e assim sucessivamente. Esse histórico pode ser visto como uma árvore binária, onde o número de divisões corresponderá aos nós e as páginas de dados como folhas.

A figura que se segue (Figura 77), pretende representar esse processo:

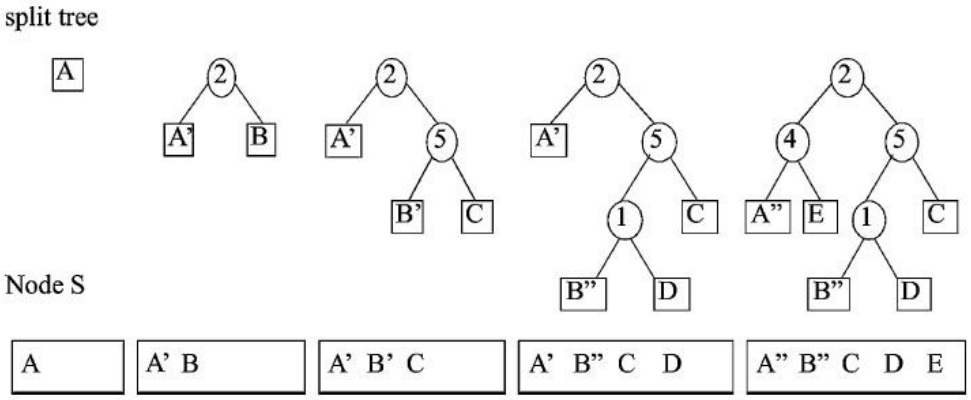

Figura 77 - Exemplo de um histórico de divisão (Böhm, Berchtold et al. 2001).

Caso ocorra um *overflow* na directoria, é necessário escolher um eixo de divisão de modo a permitir dividir, em dois, o conjunto de páginas. Neste exemplo, a melhor solução será a utilização da *dimensão 2* para a divisão, uma vez que abrange todo o espaço de dados.

A *X-Tree* utiliza este tipo de observação para realizar a divisão. Para tal, serve-se da dimensão de divisão da raiz do nó específico da árvore de divisão seleccionada, garantindo, assim, que a directoria esteja livre de sobreposição. Contudo, a divisão pode originar árvores de divisão desequilibradas. Nesse caso, é preferível não realizar a divisão, uma vez que possibilita a criação de nós com praticamente sem informação e outros perto da situação de *overflow*. Por outro lado, piora a taxa de utilização da directoria.

Nesta situação, a *X-Tree* cria um super nó, contendo uma directoria maior do que a do nó original. Assim quanto maior for a dimensão do espaço (dimensionalidade), maior será o número de super nós.

Neste tipo de espaços, a *X-Tree* tem de percorrer um elevado número de nós e, para tal, a procura linear torna-se um processo bastante mais eficiente. Contudo, é impossível fornecer valores concretos, visto que é necessário analisar vários factores (e.g. o número de itens, dimensionalidade, distribuição e o tipo de *query*) que contêm um papel preponderante no desempenho da estrutura de indexação.

SR-Tree

A *SR-Tree* resulta da combinação de outras duas estruturas estudadas anteriormente (*R*-Tree* e *SS-Tree*), (Katayama and Satoh 1997). A ideia que está por detrás desta estrutura consiste em tirar partido da utilização da intersecção dos rectângulos com as esferas, de modo a definir a região de página. Tal como na *R*-Tree*, os rectângulos correspondem às *MBR's* de todos os pontos guardados na sub-árvore. Relativamente às esferas, estas são constituídas por esferas mínimas (*SS-Tree*) onde estão contidos os objectos em torno dos respectivos *pontos centroid*.

A Figura 78 apresenta uma exemplificação desta estrutura no espaço.

A *SR-Tree* apresenta uma descrição bastante mais complexa que todas as outras estruturas anteriormente estudadas. Ela é definida por $2d$ floats para a descrever as *MBR's* e $d + 1$ floats para as esferas.

A principal motivação, segundo (White and Jain 1996), está na utilização das esferas para permite que se realizarem pesquisas *nearest-neighbor query* e *range query* de forma eficaz, utilizando apenas a métrica L_2. Contudo, apresenta algumas desvantagens, no que diz respeito à sua manutenção e à elevada sobreposição na divisão. Assim, analisando estas

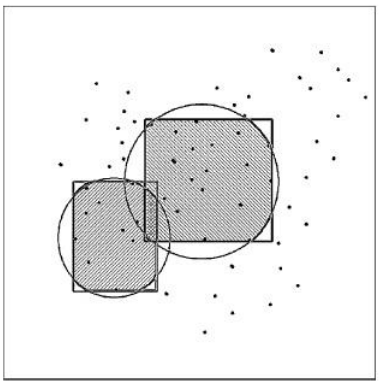

Figura 78 - Representação espacial de uma *SR-Tree*
(Böhm, Berchtold et al. 2001).

desvantagens, a combinação das duas estruturas permite ultrapassar estes problemas.

Foi definida uma função de cálculo da distância entre uma *query* e uma região, cuja expressão é dada por:

$$MINDIST(q,R) = max\big(MINDIST(q,R.MBR), MINDIST(q,R.Sphere)\big).$$

Contudo esta expressão não corresponde à distância mínima em relação ao sólido de intersecção, como podemos observar no esquema que se segue:

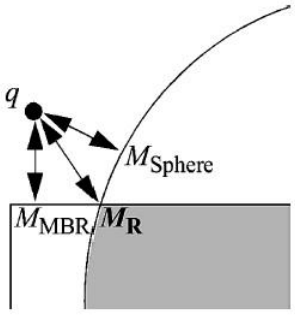

Figura 79 - Representação incorrecta do MINDIST na *SR-Tree*
(Böhm, Berchtold et al. 2001).

Quer a distância ao *MBR* quer à esfera, ambas são menores do que em relação ao sólido de intersecção. Contudo a $\mathrm{MINDIST}(q,R)$ corresponde ao limite inferior da função de distância correcta, garantindo que as pesquisas de *range* e *nearest-neighbor queries* produzam bons resultados. No entanto, a eficiência piora se a função de distância estiver incorrecta.

A MAXDIST pode ser definida por meio do mínimo das funções MAXDIST, aplicado ao *MBR* e à esfera. Relativamente à MAXMINDIST, só é possível aplicar a *MBR*, uma vez que não é possível de aplicar ao nível das esferas.

Na inserção e na divisão, o algoritmo utilizado é semelhante ao da *SS-Tree*, com umas ligeiras modificações, mais concretamente, as *MBR's* têm de ser continuamente actualizadas sempre que haja uma inserção ou uma divisão dos nós.

Anexo 2

Resultados dos Testes Efectuados sobre o Espaço de Cor HSV (108 dimensões)

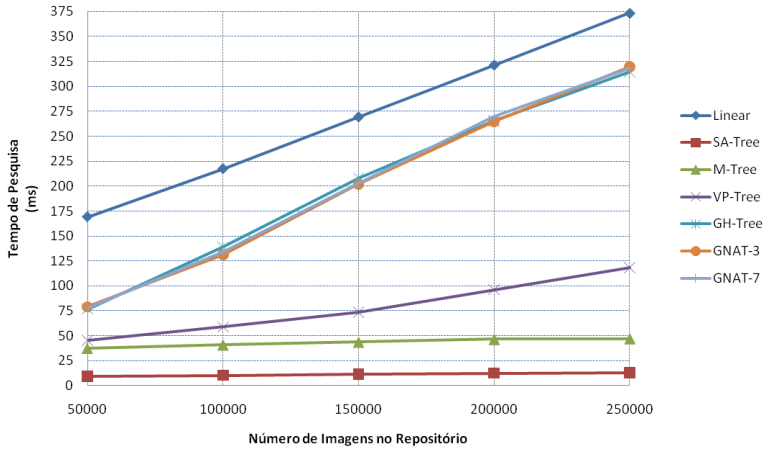

Figura 80 - Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (108 dimensões), recorrendo à métrica de *Manhattan*.

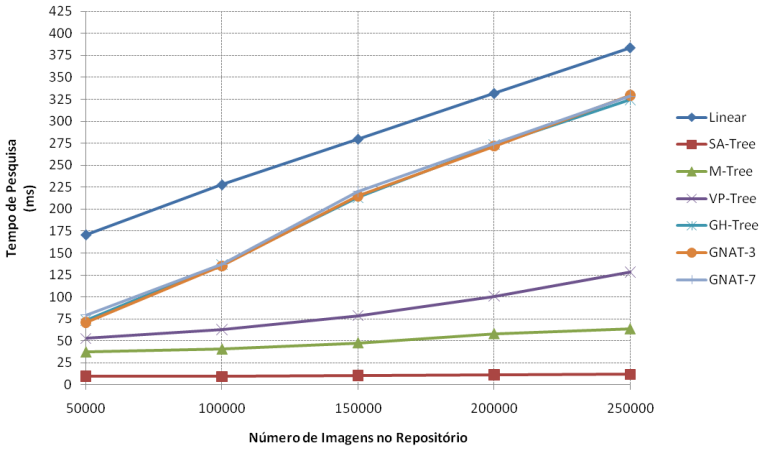

Figura 81 - Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (108 dimensões), recorrendo à métrica *Euclidiana*.

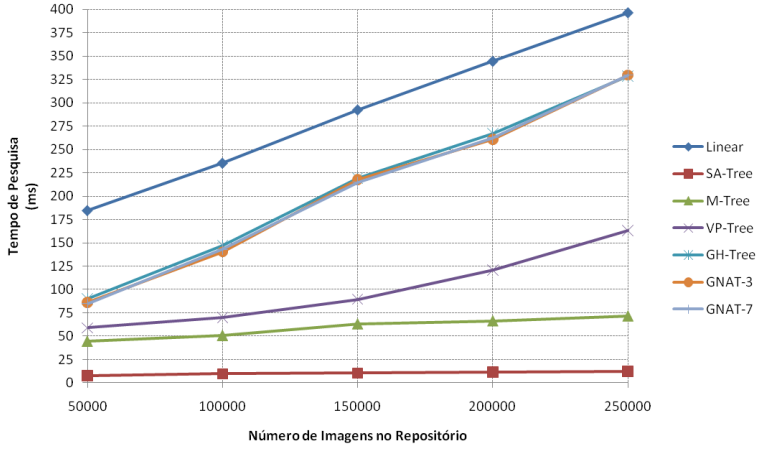

Figura 82 - Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (108 dimensões), recorrendo à métrica do *Co-seno*.

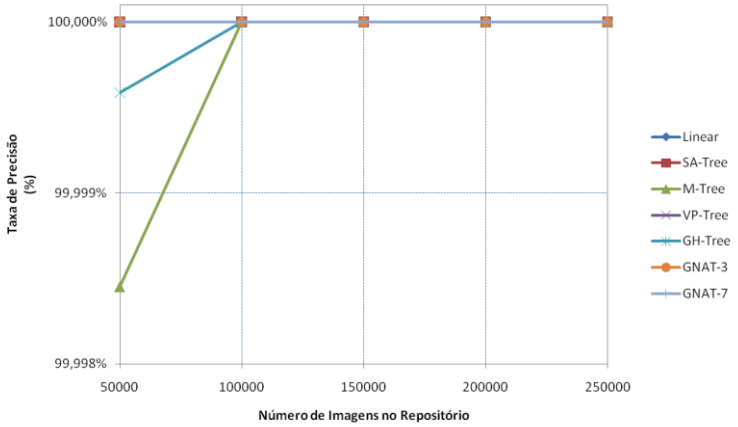

Figura 83 - Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Manhattan* sobre espaço de cor *HSV* (108 dimensões).

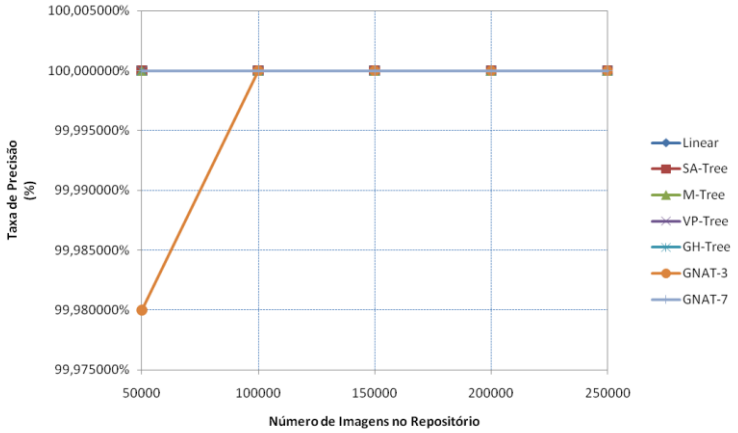

Figura 84 - Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Euclidiana* sobre espaço de cor *HSV* (108 dimensões).

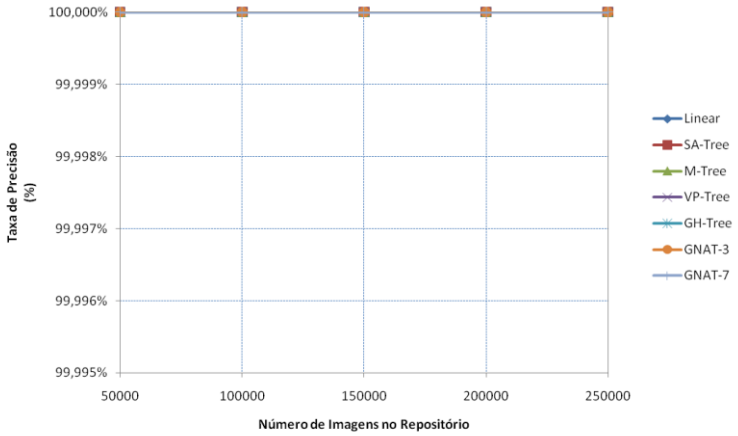

Figura 85 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Co-seno* sobre espaço de cor *HSV* (108 dimensões).

Anexo 3

Resultados dos Testes Efectuados sobre o Espaço de Cor HSV (216 dimensões)

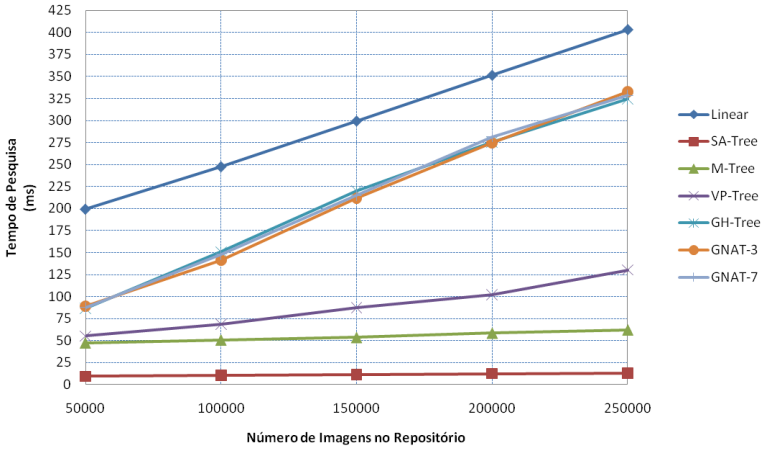

Figura 86 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (216 dimensões), recorrendo à métrica de *Manhattan*.

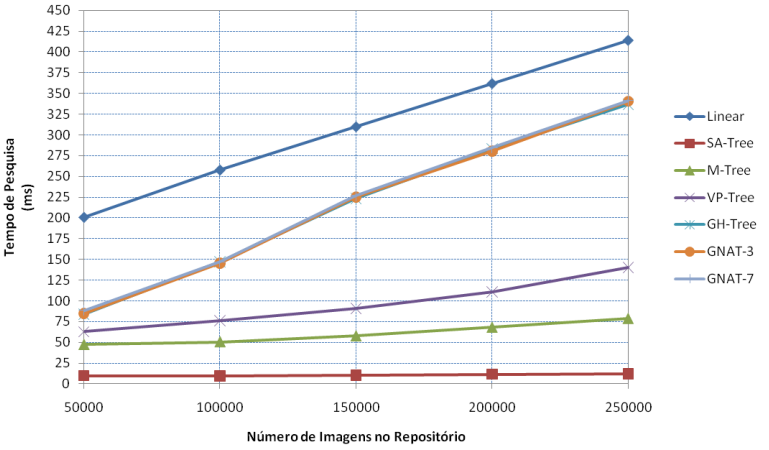

Figura 87 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (216 dimensões), recorrendo à métrica *Euclidiana*.

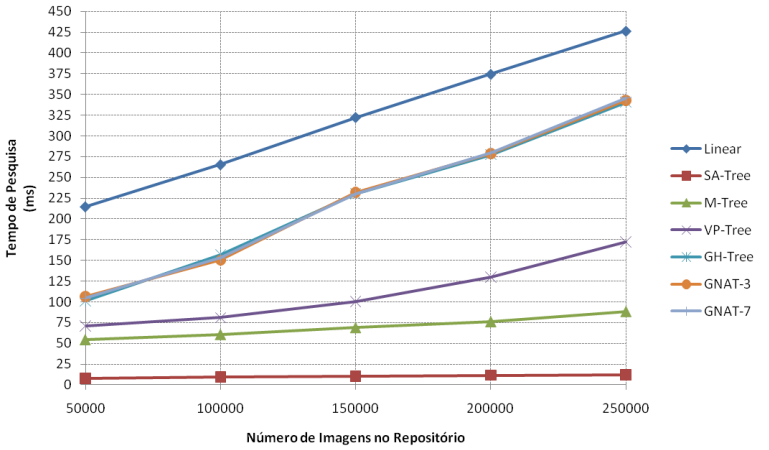

Figura 88 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (216 dimensões), recorrendo à métrica do *Co-seno*.

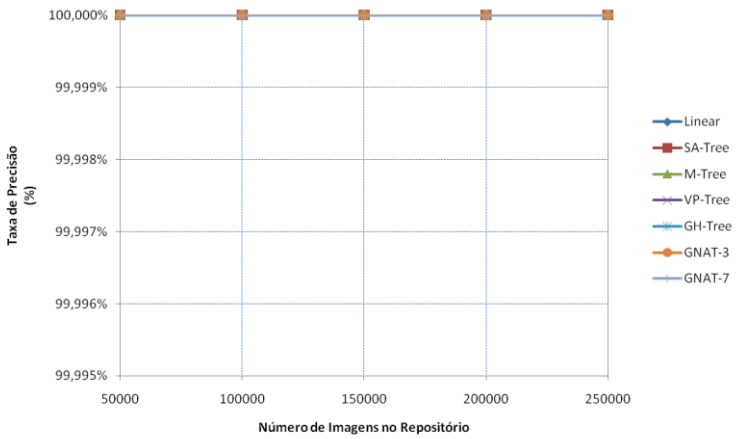

Figura 89 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Manhattan* sobre espaço de cor *HSV* (216 dimensões).

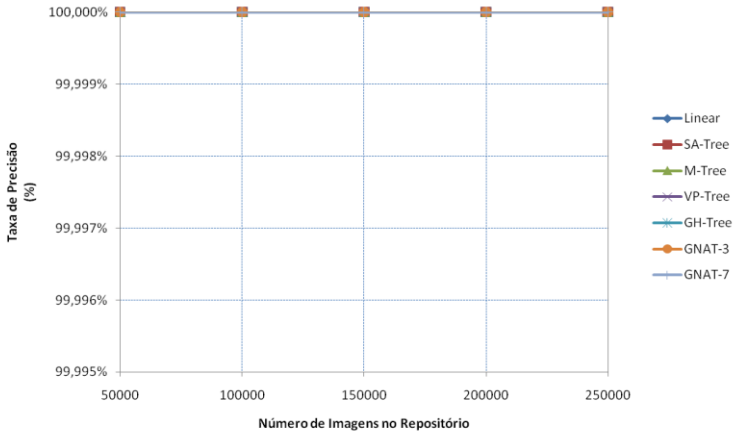

Figura 90 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Euclidiana* sobre espaço de cor *HSV* (216 dimensões).

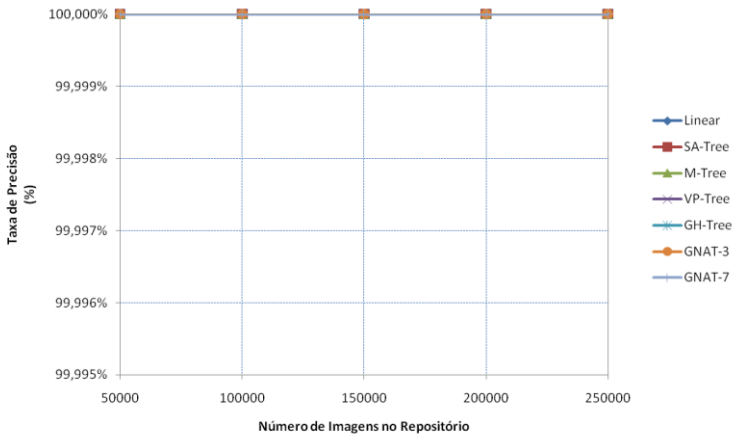

Figura 91 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Co-seno* sobre espaço de cor *HSV* (108 dimensões).

Anexo 4

Resultados dos Testes Efectuados sobre o Espaço de Cor HSV (432 dimensões)

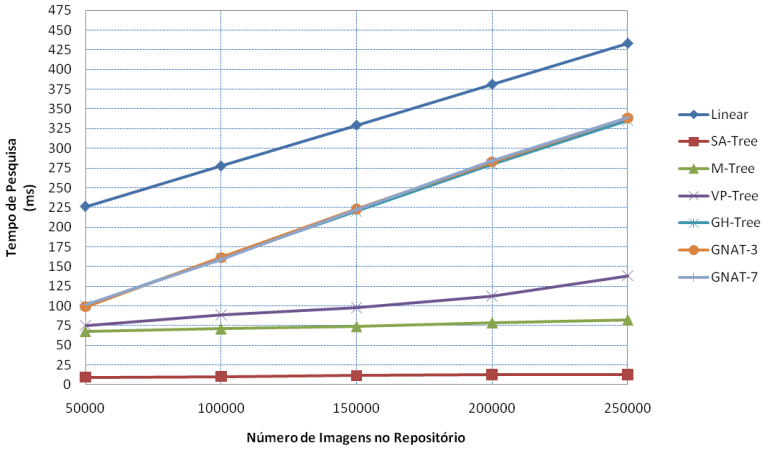

Figura 92 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica de *Manhattan*.

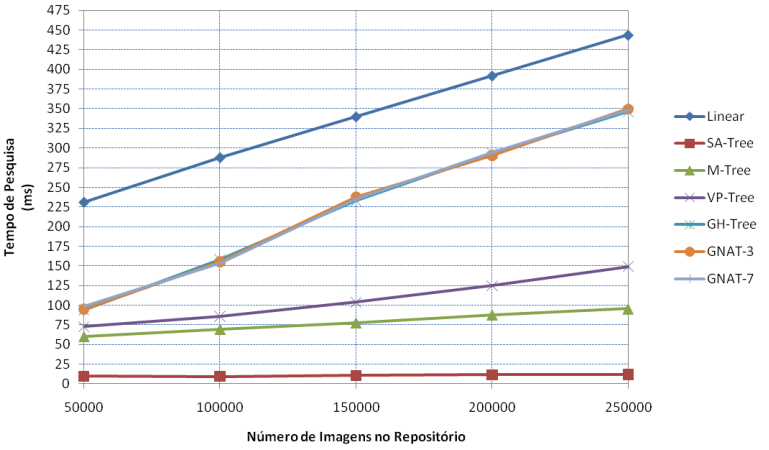

Figura 93 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica *Euclidiana*.

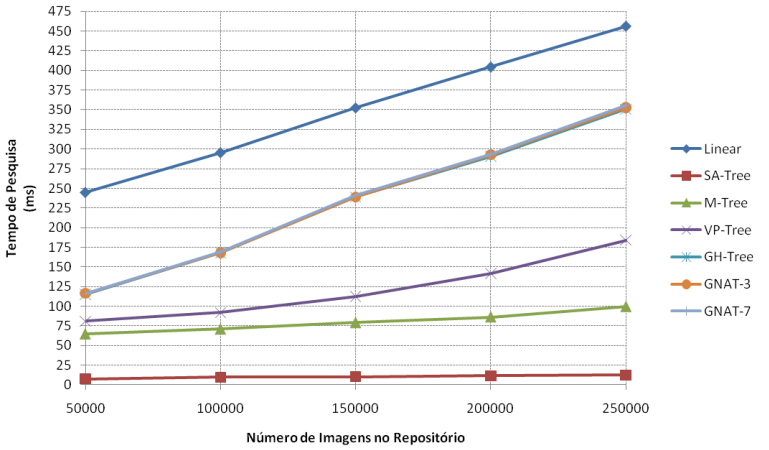

Figura 94 – Tempos de pesquisa obtidos sobre o espaço de cor *HSV* (432 dimensões), recorrendo à métrica do *Co-seno*.

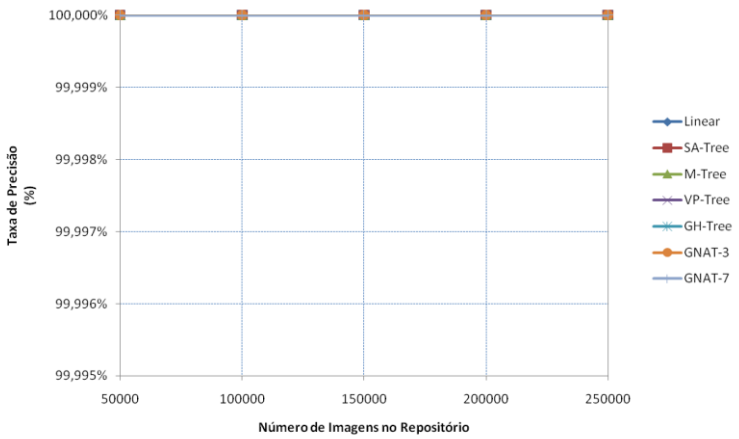

Figura 95 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Manhattan* sobre espaço de cor *HSV* (432 dimensões).

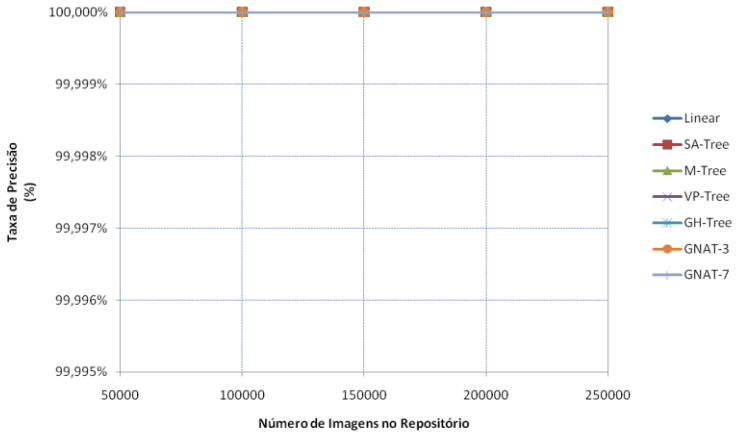

Figura 96 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Euclidiana* sobre espaço de cor *HSV* (432 dimensões).

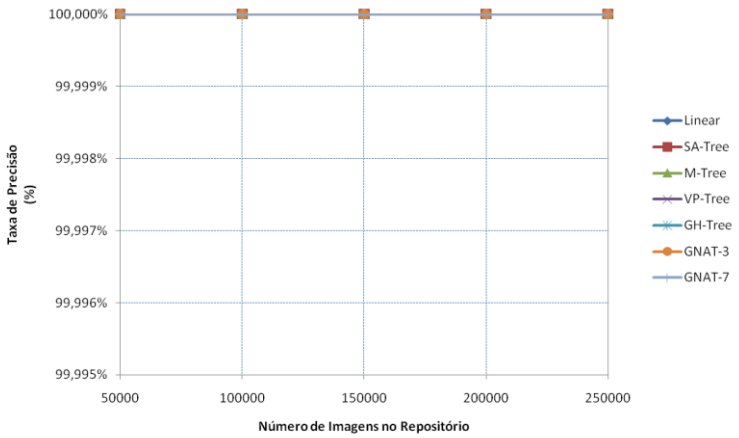

Figura 97 – Precisão dos resultados obtidos de cada uma das estruturas, utilizando a métrica de *Co-seno* sobre espaço de cor *HSV* (432 dimensões).

Glossário

A

AJAX – é um acrônimo para Asynchronous Javascript And XML e consiste no uso metodológico de tecnologias como Javascript e XML, providas pelos Web Browsers, para tornar páginas Web mais interativas, utilizando-se de solicitações assíncronas de informações.

H

HTTP - Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto) é um protocolo de comunicação utilizado em sistemas de informação de hipermedia distribuídos e colaborativos. O seu uso na obtenção de recursos interligados levou ao estabelecimento da World Wide Web.

J

Java RMI – Java Remote Method Invocation, é uma interface de programação que permite a execução de chamadas remotas no estilo RPC.

JPEG - o JPEG ou JPG é um método amplamente utilizado para comprimir imagens fotográficas. O grau de redução pode ser ajustado, permitindo escolher o tamanho de armazenamento, sem comprometer significativamente com a qualidade da imagem.

JSON - um acrônimo para "JavaScript Object Notation", é um formato leve para transferência de dados computacionais. *JSON* é um subconjunto da notação de objeto de JavaScript, mas sua utilização não requer Javascript.

O formato JSON foi originalmente criado por Douglas Crockford e é descrito no RFC 4627. O media-type oficial do JSON é `application/json` e a extensão é `.json`.

P

PDF - Portable Document Format é um formato de ficheiro, desenvolvido pela Adobe Systems em 1993, para representar documentos de maneira independente da aplicação, do hardware e do sistema operativo utilizados para criá-los. Um ficheiro PDF pode descrever documentos que contenham texto, gráficos e imagens num formato independente de dispositivo e resolução.

S

SMTP - Simple Mail Transfer Protocol é o protocolo padrão para envio de e-mails através da Internet. Consiste num protocolo relativamente simples, baseado em texto, onde um ou vários destinatários de uma mensagem são especificados (e, na maioria dos casos, validados). Posteriormente, a mensagem é enviada.

X

XML - eXtensible Markup Language é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.

Trata-se de um subtipo de SGML (Standard Generalized Markup Language) capaz de descrever diversos tipos de dados. A ideia principal do seu desenvolvimento consistiu em facilitar a partilha de informações através da Internet.

XSLT - eXtensible Stylesheet Language for Transformation é uma linguagem de marcação XML usada para criar documentos XSL que, por sua vez, definem a apresentação dos documentos XML nos Web Browsers e outras aplicações que a suportem