UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Crawler de Faces na Web

Vinicius de Carvalho Paes

Itajubá, Novembro de 2012

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Vinicius de Carvalho Paes

Crawler de Faces na Web

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciências em Ciência e Tecnologia da Computação

Área de Concentração: Sistemas de Computação

Orientador: Prof. Dr. Enzo Seraphim

Novembro de 2012

Itajubá – MG

Ficha catalográfica elaborada pela Biblioteca Mauá –
Bibliotecária Cristiane N. C. Carpinteiro- CRB_6/1702

P126c

Paes, Vinicius de Carvalho

Crawler de faces na web. / por Vinicius de Carvalho Paes. -- Itajubá (MG) : [s.n.], 2012.

69 p. : il.

Orientador: Prof. Dr. Enzo Seraphim.
Dissertação (Mestrado) – Universidade Federal de Itajubá.

1. Crawler de faces. 2. Máquina de busca. 3. Reconhecimento de faces. I. Seraphim, Enzo, orient. II. Universidade Federal de Itajubá. III. Título.

"Aqueles que passam por nós,
não vão sós, não nos deixam sós.
Deixam um pouco de si,
levam um pouco de nós."
Antoine de Saint-Exupery

Agradecimentos

Dedico meus sinceros agradecimentos para:

– Professor Dr. Enzo Seraphim, meu orientador e grande amigo, sempre dedicado a auxiliar em todas as etapas do projeto e a qualquer dúvida pertinente ao mesmo a qualquer momento. Sempre paciente e com grande boa vontade, não mediu esforços para ajudar, auxiliando a vencer todos os obstáculos enfrentados.

– A minha família, pelo apoio psicológico, extrema paciência e compreensão. Essencial para incentivar a luta em busca dos objetitvos. Sempre dispostos a auxiliar de todas as formas possíveis, visando sempre a minimizar qualquer outro problema inferente.

〃Ou nós encontramos um caminho,
ou abrimos um."
Aníbal

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Contexto

O grande volume de informação presente na *Internet* afeta proporcionalmente o interesse no tratamento dos dados e as oportunidades no meio. É fato o presente desafio no tratamento correto de todas as informações de modo que estas sejam apresentadas de forma satisfatória aos usuários. Neste aspecto é visível as mudanças expressivas na maneira como a informação é organizada, tratada e recuperada em diversas áreas de conhecimento. Há grande estudo e desenvolvimento no tratamento da informação em modo texto, porém atualmente há diversos outros formatos de interesse, e desafios em como recuperar apropriadamente tais informações. É interessante ressaltar as informações presentes como: imagens, vídeos, animações, entre outras bases de interação de usuários. Surge então um paradigma computacional em como tratar diferentes bases afim de garantir uma otimização da utilização das mesmas e apresentá-las aos usuários de interesse. Tal paradigma é objetivo, localizar a informação desejada pelo usuário da maneira mais precisa possível, tal papel cabe as famosas máquinas de busca que sofrem ajustes frequentes afim de garantir a recuperação de conteúdo relevante e minimizar o conteúdo fraudulento.

É interessante ressaltar os parâmetros usualmente utilizados pelos mecanismos ou máquinas de buscas: apenas palavras. Porém é visível a preocupação em outros métodos de pesquisa que não envolvam somente textos, abrindo um novo caminho na busca por similaridade entre imagens, que pode ser usada também para vídeos com busca sucessiva entre quadros do mesmo. No método tradicional nas máquinas de busca modernas, as mesmas possuem uma base de dados dos endereços de páginas da *Internet*, estas que são atualizadas periodicamente, analisando a relevância de conteúdo indexado, afim de garantir um meio eficiente (algoritmo) para busca de informações atualizadas [Lawrence e Giles 2000].

A *Internet* atualmente possui um infinidade de conteúdo de diversas áreas e muitas vezes funciona como uma grande enciclopédia para leigos. Outro ponto relevante diante o volume de informação é o crescente aumento de imagens armazenadas na rede. Há

diversos motivos para este aumento substancial, principalmente pela popularização de câmeras digitais e também pela maior afinidade do público em geral com a utilização da grande rede de computadores. Desta forma também surgiu o crescimento e interesse por mecanismos de busca em utilizar imagens como parâmetros de busca, afim de buscar um conjunto de características semelhantes em outras imagens [Jing e Baluja 2008].

Há diversas possibilidades relacionadas a pesquisa eficiente de imagens na internet, sobretudo de faces. A integração do sistema de busca com um diretório de faces de pessoas desaparecidas poderia garantir mais uma ferramenta afim de auxiliar as famílias na busca de entes queridos. No lado comercial, há também a possibilidade de busca de fotos com direitos de imagem, ou integração para buscas por similaridade de imagens ou pessoas. Um banco de dados global de faces indexadas também poderia ser útil para auxiliar a polícia em busca de criminosos foragidos, porém ativos no meio eletrônico, ou mesmo no auxílio de busca de perfis falsos na rede.

A *Internet* possui muitos desafios focados na busca de informação, além de possuir vasta concentração de conteúdo heterogêneo [Page et al. 1998]. É interessante explorar métodos eficientes de busca de informação de qualidade.

1.2 Objetivo

O foco primordial neste projeto é definir a estrutura básica necessária para o desenvolvimento e aplicação prática de uma máquina de busca de faces, afim de garantir uma busca com parâmetros qualitativos apropriados.

1.3 Estruturação

O trabalho está estruturado nas seguinte etapas: A fundamentação teórica para normalizar pontos-chave utilizados, em seguida será dado um foco nas necessidades e desafios perante o processamento de imagens. O capítulo seguinte é voltado especificamente para a máquina de busca desenvolvida ressaltando aspectos relevantes da mesma. No último capítulo do trabalho serão apresentados os resultados obtidos e aspirações de trabalhos futuros.

2 Fundamentação Teórica

2.1 Web Crawler

Um *web crawler* pode ser definido como um programa de computador que percorre a *Internet* de forma sistemática e pré-definida afim de buscar informações.

A nomeclatura mais comum para o processo em execução é *Web Crawling*. Porém há muitas variações desta nomeclatura relativa aos indexadores automáticos, como: *bots*, *web spiders*, *web robot*, *web wanderer* e *web scutter* [Kobayashi e Takeda 2000]. Por fim, há grande confusão nos termos do processo de *Crawling* e *Indexing*. O processo de *crawling* somente cria a lista de *links* das páginas percorridas. Já na indexação, é feito o *download* de uma cópia da página para que a informação seja processada e classificada.

O método mais comum de navegação é através de um conjunto de *links* pré-selecionados, também conhecidos como a semente (*seed*). Deste modo, novos *links* são obtidos à partir da cadeia de *links* iniciais.

Desta forma, fica claro como é obtido o conjunto de caminhos possíveis, sendo necessário apenas definir qual a metodologia de qual *link* seguir dentro da cadeia de *links* obtidos.

É interessante ressaltar que os *web crawlers* possuem uma infinidade de aplicações grande parte delas envolvendo métodos automatizados. É possível sua utilização no auxílio as máquinas de busca, fazendo uma cópia da página a ser indexada, ou mesmo executar processos sistemáticos para manutenção de portais na *Internet*, seja para verificação da consistência de *links*, ou mesmo validação de códigos em *HTML* [Baeza-Yates et al. 2005]. Há também processos visando coleta direta de informação pessoal, como informações sobre endereços de *email*, tendências sobre produtos, opiniões, entre outros.

Tais processos de coleta de informação na *Web* são datados desde o primórdio da *Internet* [McBryan 1994] [Pinkerton 2000]. A complexidade dos mecanismos de coleta de informação pode variar drasticamente diretamente proporcional ao nível de extração e

classificação de conteúdo. É interessante destacar o desenvolvimento de *crawler* de nível acadêmico voltado a um universo amostral restrito para fins de estudo com complexidade menor, sendo possível citar também aqueles de caráter estritamente comercial, como o AltaVista, Bing, Yahoo! Search e Google [Brin e Page 1998].

Outra questão pertinente é quando o *crawler* deve parar seu funcionamento, pois com o número sempre crescente de páginas o processo poderia demorar muito a encerrar. Existem diversas formas para tal e cada *crawler* possui uma determinada finalidade e um parâmetro de parada próprio. O quesito de parada pode ser relativo ao volume indexado, o tempo de execução, a limitação física de processamento ou unidade de armazenamento, entre outros.

Destaca-se três problemas que são usualmente desafiadores aos *crawlers*, como citado por Edwards em [Edwards, McCurley e Tomlin 2001]: a grande aquisição de informação, a volatilidade da informação obtida e o conteúdo em páginas geradas dinamicamente ocasionando redundância em informação em *URL* distintas.

Diante o paradigma de desafios do *crawler*, relativo ao grande acúmulo de informação não se pode garantir que o conteúdo seja diversificado oriundo de páginas distintas, é possível que o *crawler* possa ter adquirido informação de apenas uma parte da *web*, sendo necessário uma política de acesso as páginas. Já no aspecto de volatilidade de informação é interessante ressaltar que há constantes mudanças nas páginas acessadas, sendo que o conteúdo pode ter sido alterado, novas páginas adicionadas, ou excluídas. É necessário então ter uma política de revisita para garantir a confiabilidade do processo. Um ponto chave também no processo de aquisição de informação com o *crawler* é diante as páginas geradas dinamicamente e os parâmetros a serem combinados afim de visualizar o conteúdo: ordenação de conteúdo, escolha de temas de conteúdo, data de inserção da informação entre outros. Desta forma, é possível gerar uma infinidade de informações combinadas para visualização, gerando grande redundância da informação, necessitando um tratamento especial pelo *crawler*.

Analisando estes três desafios principais, a política de metodologia de comportamento de um *crawler* segue alguns princípios básicos:

- A política de paralelização: é fundamental para garantir o funcionamento do *crawler* em ambientes distribuídos.
- A política de convivência: não é possível executar o *crawler* de forma a usar todo seu potencial sem um controle. É necessário haver esta política para garantir que o

mesmo respeite os servidores de páginas na *web* a fim de evitar sobrecarga.

- A política de seleção: é a política base de aquisição de *URL* afim de garantir uma lista de páginas a serem visitadas.
- A política de revisita: levando em conta a volatilidade de toda a informação contida na *web*, é interessante garantir uma política de revisita das páginas para verificar eventuais mudanças.

2.1.1 Política de Paralelização

É fundamental a execução em vários processos paralelos para otimização do *crawler*. As vantagens básicas são a minimização da sobrecarga da paralelização, além a maximização da taxa de carregamento das páginas, com isso há um ganho afim de evitar sobrecarga ao carregar páginas de forma repetida. Este processo de evitar o carregamento de páginas repetidas pode parecer trivial, porém com o aumento do volume de descobertas de *URL* é necessário uma estratégia específica para evitar tal sobrecarga [Shkapenyuk e Suel 2002].

2.1.2 Política de Convivência

A política de convivência é primordial no desenvolvimento de um *crawler* que respeite a estrutura de demais servidores na *Internet*. O poder de processamento de um *web crawler* é diretamente proporcional a estrutura onde o mesmo está instalado. Desta forma, como é possível que o mesmo adquira um volume muito grande de informação em pouco tempo, fazendo requisições múltiplas em determinado servidor *web*, é interessante definir uma política de convivência para que não haja sobrecarga neste servidor, pois o mesmo também poderá estar servindo páginas para demais usuários, sendo interessante manter seu serviço estável sem que haja um acesso prejudicial ao mesmo [Cho e Garcia-Molina 2003].

Como definido, o *crawler* pode obter informações de maneira muito rápida, e este poder de aquisição é relativo a toda a sua estrutura:

- Acesso a *Internet*: como um dos objetivos principais de um *crawler* é relativo a tarefas de aquisição de informação na *web*, é interessante que a estrutura de rede satisfatória e uma largura de banda considerável.
- Servidor: estrutura de servidor satisfatória afim de garantir o número de conexões simultâneas para satisfazer a aquisição de informações.

É interessante destacar que *crawlers* escritos de forma incorreta, ou com metodologia falha podem vir a causar danos de disponibilidade a servidores, ou até mesmo a roteadores, devido a grande demanda de conexões simultâneas, ou mesmo por acesso a páginas no qual não conseguem processar o conteúdo. Há também o problema com *crawlers* de cunho acadêmico ou pessoal que se utilizados por muitos usuários podem vir a interromper o bom funcionamento de servidores de páginas na *web*.

Uma solução usual para o problema de convivência e de tratamento de páginas é através do protocolo *Robot Exclusion Protocol*, sendo um padrão para determinar as páginas que não devem ser acessadas por determinados *crawlers* [Koster 1996]. Basicamente é necessário definir um arquivo chamado *robots.txt* com as configurações de acesso (permitir ou negar) a determinadas páginas e diretórios. É possível determinar também um arquivo do mapa do site em XML, contendo todas as *URLs* do site a serem indexadas, e alguns *crawlers* aceitam um parâmetro chamado *craw-delay* indicando uma boa política de convivência (número de segundos de diferenças nas requisições, afim de evitar sobrecarga)

Existem diversas propostas na política de intervalo entre as conexões, sendo as mesmas propostas em diferentes *crawlers*. A proposta usual deste intervalo era de 60 segundos, porém é apresentado no estudo do *WIRE Crawler* [Baeza-yates e Castillo 2002] uma proposta de 15 segundos de intervalo de tais requisições. Outros trabalhos propostos utilizam tempos ainda menores, como em [Cho e Garcia-Molina 2003] utilizando 10 segundos e Dill em [Dill et al. 2002] utilizando apenas 1 segundo de intervalo. Uma proposta interessante é do *MercatorWeb* que leva em consideração o tempo de latência do servidor, com uma política de convivência adaptativa, levando em consideração o tempo de aquisição da informação do servidor esperando dez vezes este tempo para a nova solicitação.

Outro aspecto relevante na política de convivência é evitar que com um intervalo muito curto o processo de crawling seja confundido como um ataque ao servidor. O mesmo pode ocorrer de fato, pois o *crawler* tem potencial para realizar múltiplas requisições simultâneas que podem sobrecarregar o servidor de páginas e até mesmo derrubá-lo. É interessante destacar também as políticas de defesa dos servidores, que muitas vezes possuem estratégias automáticas em sobrecarregar o solicitante afim de garantir a sobrecarga reversa, podendo ocasionar um travamento no *crawler*, obrigando a reinicialização do mesmo. Desta forma, é interessante que haja uma notificação de qual *User-Agent* está solicitando a página, sendo interessante informar que o mesmo seja um *crawler*.

2.1.2.1 Identificação de um Web Crawler

Os *web crawlers* possuem um mecanismo para se identificar quando estão executando o processo de busca de informação na *Internet*. Este mecanismo é a definição de seu *User-Agent* quanto estão fazendo uma requisição *HTTP* de uma página na *Internet*. Os administradores dos servidores e de *web sites*, visualizam periodicamente os logs de acesso dos sites para verificar as fontes de tráfego, além de outras informações afim de determinar também quais *web crawlers* acessaram informações do sites e em qual intervalo de tempo. Os administradores também podem verificar a *URL* associada ao *User-Agent* do *web crawler* afim de obter informações adicionais sobre os objetivos deste *crawler*. Desta forma, esta identificação é necessária pois existem *web crawlers* com objetivos maliciosos, seja para obtenção de emails em sites para criar listas de *spam*, entre outros. Tais mecanismos malicosos, usam estratégias para mascarar sua identidade, seja se identificando como um navegador de *Internet*, ou como outro *crawler* conhecido. Desta forma, a identificação do *crawler* é importante para garantir um canal de comunicação entre os administradores de diversos sites e os desenvolvedores do *crawler*. É interessante ressaltar, que é comum alguns problemas diante a política de convivência do *crawler* mediante a requisições em um dado site. Muitas vezes, estes problemas podem estar relacionados a uma armadilha para o *crawler*, mecanismo de defesa dos administradores afim de garantir segurança em seus servidores, em alguns casos, o excesso de requisição por má configuração também pode vir a ocorrer, sendo necessário a intervenção do desenvolvedor do *crawler*, para que o mesmo seja desligado. Do ponto de vista dos benefícios da identificação de um *crawler* por parte dos adminitradores é diante o entendimento de como seu site é visto pelo *web crawler*, e a expectativa do mesmo na obtenção de informação, ou mesmo indexação.

2.1.3 Política de Seleção

A política de seleção é muito importante tendo em vista o vasto conteúdo que compõe a *web*. Até mesmo os *crawlers* mais sofisticados tem dificuldades e conseguem varrer apenas uma parcela da *Internet*. No estudo desenvolvido por [Diligenti et al. 2000] os mesmos relataram que os *crawlers* na época conseguiam mapear apenas 16% das páginas na *web*. Um ponto de destaque na obtenção de página na *web* pelos *crawlers*, é que o mesmo seja feito de modo otimizado afim da obtenção da maior quantidade possível de páginas relevantes. Como a obtenção é apenas de uma fração da mesma, este ponto-chave é primordial afim que garanta que não há apenas uma aquisição aleatória de informações.

Afim de garantir uma métrica relevante na obtenção de páginas, e também garantir

certa priorização nas mesmas, é interessante definir uma função que avalia a sua qualidade perante sua finalidade. Desta forma é utilizado alguns conceitos da teoria de grafos, para determinar as arestas incidentes (*links* de entrada para a página), as ligações destas páginas em relação aos seus vizinhos (*links* de saída da página, e a relação de pertinência nas demais páginas afim de garantir um mesmo contexto), o número de visitas, e a densidade geral de palavras-chave da página afim de evitar conteúdo com spam.

É necessário entender diante a política de seleção que há certos pontos de convergência diante o modelo quantitativo e qualitativo da informação. Este ponto se relaciona diretamente ao modelo parcial de aquisição de informação, pois é de conhecimento apenas parte do universo amostral da *web*. Desta forma, o processo de coleta tem início por um conjunto inicial de *URLs*, chamadas de sementes, estas que referenciam páginas *web* [Chakrabarti, Berg e Dom 1999]. Após a extração inicial, é tratado a aquisição de conteúdo das páginas, e posteriormente a extração de ligações (URLs) para outras páginas que passarão pelo mesmo processo na próxima iteração.

É interessante destacar o estudo de Cho em [Cho, Garcia-Molina e Page 1998] para políticas de escalonamento na busca, onde foi elaborado uma coleta de 180.000 páginas da própria universidade (stanford.edu) utilizando estratégias distintas na busca. As estratégias de busca focaram inicialmente em cálculos simétricos ao *PageRank* de cada página, a contagem de *links* de entrada, e busca em largura. Nos resultados obtidos, foi concluído que as estratégias focadas no *PageRank* obtiveram resultados mais precisos, seguidas da busca em largura e contagem de *links* de entrada. Nota-se que o resultado é referente a apenas ao domínio da faculdade, que está dentro de um contexto geral na área de educação. No trabalho elaborado por Najork e Wiener em [Najork e Wiener 2001], foi realizado um conjunto de busca de informação dentro de um universo amostral de 328 milhões de páginas utilizando a busca em largura. Um ponto peculiar neste tipo de busca foi que nos resultados houve uma incidência alta de páginas com *PageRank* elevado. É interessante relatar que também no estudo de Abiteboul em [Abiteboul, Preda e Cobena 2003] utilizando seu algoritmo próprio denominado OPIC (On-line Page Importance Computation), foram obtidos resultados similares ao do algoritmo de *PageRank*, porém com um menor custo computacional.

Há também outros estudos realizados por Boldi em [Cothey 2004] utilizando testes comparativos com algoritmos de busca em largura contra busca em profundidade. Nos resultado obtidos por Baeza em [Baeza-Yates et al. 2005] realçava as vantagens na utilização de algoritmos baseados na estratégia OPIC, juntamente com algoritmos que utlizavam

comprimentos de fila em relação aos algoritmos de busca em largura e profundidade.

É interessante ressaltar demais considerações relativas as políticas de seleção, afim de caracterizar alguns desafios a serem alcançados, juntamente com eventuais estudos afim de garantir a otimização nos meios de aquisição de informação:

- O *crawler* pode encontrar dificuldade ao interpretar algum tipo de extensão de arquivo não familiar ao seu processo de aquisição de informação. Este é um fato comum, visto que a *web* pode conter diversos tipos de formatos. Uma estratégia para evitar perda de processamento em requisições onde o *crawler* não possui algoritmos para tratamento é a limitação a tipos suportados.

- Um ponto importante é relativo a geração de conteúdo dinâmico em algumas páginas *web*. É possível que o *crawler* faça requisições múltiplas nestas páginas, podendo gerar redundância de informações obtidas.

- Há uma proposta interessante por Cothey em [Cothey 2004] propondo buscas ascendentes nos caminhos que formam cada URL. Utilizando esta técnica Chothey apresentou que é possível encontrar páginas isoladas que não possuem ligação por nenhuma outra página.

- Estudos propostos por [Menczer 1997] [Menczer e Belew 1998] e por [Chakrabarti, Berg e Dom 1999] fazem relacionamento por uma função de similaridade na busca entre páginas. Outros pontos-chave e desafios utilizando a busca focada são citados também por [Pinkerton 2000] e por [Diligenti et al. 2000].

- Há também páginas sem incidência de *links* externos, sendo acessíveis somente por uma consulta direta a uma base de dados. O protocolo proposto pelo Google, denominado Sitemap (Mapa-do-Site) e o *mod oai* proposto por [Nelson et al. 2005] visam garantir um meio de documentar estas páginas para que possam ser recuperadas por mecanismos de busca.

2.1.3.1 Páginas sem referência direta

É interessante destacar que uma boa parte do conteúdo disponível na *Internet* permanece fora de um escopo visível. Este escopo visível pode ser definido como as páginas que possuem *links* de entrada a partir de outras páginas. Desta forma o escopo de páginas invisíveis na *web* é definido como as páginas que não possuem nenhuma referência direta de *links* para ela. Uma das características que podem destacar este escopo de páginas

invisíveis são aquelas que são visíveis apenas após envio de informações de busca (query) a um banco de dados. Desta forma, *crawlers* usuais não possuem mecanismos eficientes de busca destas informações, caso não haja uma metodologia padronizada conhecida, ou mesmo referência de *links*. Uma alternativa para facilitar a busca e indexação destas informações pelos *crawlers* é utilizando o protocolo de sitemaps das páginas juntamente com o *mod oai* que de forma pré-definida garantem referências a tais páginas de forma que os mecanismos de busca entendem.

A técnica de *Deep Web Crawling* também intensifica a busca e procura de *links* de determinadas URLs. Basicamente, ao buscar *links* em uma página, o *crawler* pode utilizar apenas parte da URL, ou mesmo todo o conteúdo de hipertexto para buscar novos *links*.

2.1.3.2 Similaridade entre páginas

É interessante ressaltar a importância de uma página para um *crawler* também pode ser quantificada, ou expressada como uma função da similaridade de uma página para uma expressão dada. Desta forma, por definição *web crawlers* que utilizam da técnica de download de páginas por similaridade entre elas, são chamados de *focused crawler* ou também como *topical crawlers*. Tais definições sobre *crawlers* foram introduzidas primeiramente por [Menczer 1997] e [Chakrabarti, Berg e Dom 1999].

Um grande dilema na técnica de *focused crawling* (similaridade entre páginas) é nas definição do contexto do *web crawler*, pois é interessante ter a possibilidade de prever a similaridade do texto de uma página dada para uma expressão antes mesmo de efetuar o *download* da página. Uma das possibilidades é utilizar o texto âncora dos *links* como uma possível técnica de previsão, como feito no desenvolvimento dos primeiros mecanismos de busca da *web*. O estudo de Diligenti em [Diligenti et al. 2000] propôs a utilização de todo o conteúdo das páginas já visitadas para fazer uma inferência na similaridade entre a expressão dada para as páginas que ainda não foram visitadas. Em termos de otimização, a performance de um *focused crawler* depende muito da qualidade dos *links*, do texto âncora definido e da expressão a ser pesquisada. Uma característica também deste processo é que o *focused crawler* utiliza de um mecanismo de busca afim de obter os primeiros *links* para iniciar o processo.

2.1.3.3 Restringir links Seguidos

Para otimizar o processo de obtenção de informação, usualmente o *web crawler* é configurado afim de buscar apenas conteúdo em formato texto (*HTML*), e ignorar os demais tipos de conteúdo. De forma prática, este processo é executado durante a requisição, buscando primeiramente o *HTTP HEAD* do conteúdo afim de determinar o tipo do conteúdo antes de requisitar todo o conteúdo com uma requisição do tipo *GET*. Como este processo pode causar um overhead, e afim de garantir uma política de convivência satisfatória, o *web crawler*, pode ser configurado primariamente a analisar o tipo de *URL* da requisição, afim de determinar quais tipos de *URL* são potencialmente páginas em formato texto.

É interessante definir na configuração do *web crawler* se este tem ou não a intenção de buscar conteúdo gerado dinamicamente. O processo de negar busca de conteúdo dinâmico (possivelmente definido quando há um caracter de interrogação na *URL*) também evita as famosas armadilhas de *crawler*, pois devido a inúmeras páginas que podem ser produzidas com o conteúdo dinâmico, pode direcionar o *web crawler* a fazer o download e armazenar infinitas *URLs* de um mesmo site. Porém, caso os sites possuam a configuração de *URL rewrite* para simplificar a *URL*, é possível mascarar se a página é produzida por conteúdo dinâmico ou não.

2.1.3.4 Normalização de URL

Os *web crawlers* usualmente executam o processo de normalização de *URL* afim de evitar que busquem a mesma página mais de uma vez. Este termo de normalização também é conhecido como canonização de *URL*, que pode ser definido como a modificação afim de padronizar uma *URL* de forma consistente. Existem vários processos afim de normalizar uma *URL*, desde a modificação de todos os caracteres para minúsculo, remoção de pontos e segmentos de pontos e outros caracteres.

2.1.3.5 Crawler de Caminho Ascendente

É interessante ressaltar que nem todos os tipos de conteúdo na *Internet* possuem *links* direcionados. Desta forma, há uma proposta de *web crawler* exatamente afim de buscar informação utilizando uma técnica de ascenção de caminho. Dado o exemplo de uma *URL*: http://www.unifei.edu.br/graduacao/cursos/ciencia-computacao.html, o *crawler* tentará buscar páginas em todos os diretórios definidos pela *URL*, como: /graduacao/cursos/, /graduacao/ e /.

Muitos *web crawlers* que utilizam a técnica de busca por ascenção de caminho também são conhecidos como *web harvesting*, pois são utilizados para buscar todo um conteúdo de um domínio, como um exemplo um site de galeria de fotos, onde há uma separação por diretório entre elas.

2.1.4 Política de Revisita

A volume de informação na *Internet* possui uma característica muito peculiar relativo a volatilidade de mudanças, sendo também que uma determinada pesquisa em uma parcela da *web* possa demandar um espaço de tempo muito longo que dependendo da estrutura de redes e servidores, pode levar muitas semanas e até mesmo meses. Relativo agora as características do *crawler*, o mesmo possui algumas políticas de parada, por limitação de recursos, ou mesmo por meta de aquisição de informação concluída. O grande paradigma em questão é o fato que muitas mudanças podem ter ocorrido, como atualização das páginas com inserção de conteúdo, ou mesmo exclusão.

O mecanismo ou máquina de busca não detecta de forma direta os eventos de mudança das páginas. Existem diversos mecanismos de notificação as máquinas de busca sobre atualizações na página, mas nem todas as páginas estão configuradas de modo a fazer tal notificação. Desta forma, as máquinas de busca possuem conteúdo desatualizado das páginas. Em [Cho e Garcia-Molina 2000] é proposto funções *freshness* para obter um meio de medir o nível de confiabilidade (atualização/desatualização) dos conteúdos de tais páginas. Esta é definidida por S = p_1,...,p_n, onde p_i representa uma página e N é a quantidade de elementos desta página.

O termo *freshness* e *age* proposto por [Cho e Garcia-Molina 2000] propõe uma medida binária para indicar se a cópia local da página está atualizada ou não. É possível definir a medida de *freshness* de uma página p_i no banco de dados local, no tempo t utilizando a Função 2.1.

$$\mathrm{F}(\mathrm{p}_i, t) = \begin{cases} 1, & \text{se } p_i, \text{ é igual a cópia local no tempo t} \\ 0, & \text{caso contrário} \end{cases} \quad (2.1)$$

Assim, temos a função *freshness* de um determinado banco de dados S no tempo t

definida na Equação 2.2.

$$F(S,t) = \frac{1}{N}\sum_{i=1}^{n} F(p_i,t) \tag{2.2}$$

Já a função *age* determina a medida de quão desatualizada a cópia local da página está. O *age* de uma página p no banco de dados, no tempo t é definida na Equação 2.3

$$\mathrm{A}(\mathrm{p}_i,t) = \begin{cases} 0, & \text{se } p_i \text{ não está modificado no tempo } t \\ T, & \text{modificado no tempo } p \text{ caso contrário} \end{cases} \tag{2.3}$$

Assim definida, a função A de um banco de dados local S é determinado na Equação 2.4.

$$A(S,t) = \frac{1}{N}\sum_{i=1}^{n} A(p_i,t) \tag{2.4}$$

Há uma proposta interessante idealizada por [Coffman, Liu e Weber 1997], o conceito basicamente era a minimização de tempo das páginas desatualizadas na base de dados local. Este conceito é semelhante com o proposto por [Cho e Garcia-Molina 2000], porém seguia uma abordagem distinta. É interessante ressaltar também outros aspectos de políticas de revisita desenvolvidas por [Cho e Garcia-Molina 2003], como destacadas a seguir:

- Política Proporcional: nesta política, é verificada as páginas mais voláteis onde o conteúdo é modificado mais frequentemente afim de revisitá-las com uma frequencia maior. Esta frequencia de revisita é proporcional as frequências de mudanças.
- Política Uniforme: Ignora a frequência de mudanças das páginas, desta forma o processo de revisita possui a mesma frequência em todas as páginas.

2.1.5 Web Crawlers Populares

Há uma infinidade de *web crawlers* desenvolvidas para diversas atividades, porém é interessante citar aqueles que ganharam grande destaque com os mecanismos de busca

modernos.

- *GoogleBot* - *Crawler* do Google que tem grande destaque pela revolução dada aos mecanismos de busca modernos.
- *Bingbot* (antigo *MsnBot*) - É o *crawler* do mecanismo de Busca da *Microsoft*, o *Bing*. Seguindo a evolução dada pelo *Google*, a *Microsoft* não ficou atrás e também criou sua solução moderna para buscas.
- *Yahoo! Slurp* - Também um *crawler* famoso dos mecanismos de busca, seguindo os passos do *crawler* do *Google* afim de garantir uma forma mais inteligente na busca por informação na *web*.

Outros tipos de *web crawlers* que receberam destaque:

- *Fast Crawler* - *crawler* distribuído utilizado pela empresa *Fast Search Transfer*.
- *PolyBot* - *crawler* distribuído desenvolvido na linguagem *C++* e *Python*. Possui uma central de configuração onde é possível configurar diversas características para a captação de informação.
- *Rbse* - primeiro *web crawler* publicado. Sua estrutura envolve a integração de 2 sistemas: o primeiro é denominado um *spider* que mantém a fila de *URL* em um banco de dados relacional e o segundo sistema denominado *mite* é basicamente um browser modificado que baixa as páginas da *Internet*.
- *World Wide Web Worm* - *crawler* com finalidade de apenas indexar títulos de documentos e *URLs*, utilizando o comando *grep* do *unix*.

Alguns *web crawlers open source* de destaque:

- *Aspseek (www.aspseek.org)* - *crawler*, indexador e mecanismo de busca escrito em *C++* sob licença *GPL*.
- *GNU Wget (www.gnu.org/software/wget)* - *crawler* em linha de comando desenvolvido em *C* sob licença *GPL*.
- *GRUB (www.gnu.org/software/grub)* - *crawler* distribuído utilizado pela *Wikia Search* para formar sua base de dados.

- *Nutch (nutch.apache.org)* - *crawler* desenvolvido em *Java* e licenciado com a licença *Apache*.
- *Open Search Server (www.open-search-server.com)* - máquina de busca e *crawler* licenciado em *GPL*.
- *PHP-Crawler (astellar.com/php-crawler)* - *crawler* desenvolvido utilizando as tecnologias *PHP* e *MySQL* sob a licença *BSD*.
- *Larbin (larbin.sourceforge.net/index-eng.html)* - *crawler* desenvolvido em *C/C++*, robusto e com divesas opções de parametrização de busca.
- *Crawler4J (code.google.com/p/crawler4j)* - *crawler* desenvolvido em *java*, com diversos recursos de configuração e integração com demais ferramentas.

2.2 Máquina de Busca

2.2.1 Objetivo de uma Máquina de Busca

Uma máquina de busca *web* é desenvolvida primordialmente com a finalidade de buscar informação consistente na *Internet* a partir da interação do usuário com palavras-chave de interesse. Os resultados de pesquisa são geralmente apresentados como uma lista de *links* associados ao conteúdo de cada página, como texto, imagens, ou outros tipos de arquivo. Estes resultados são conhecidos como as páginas de resultado do mecanismos de busca. (do inglês: *SERPS*, *search engine results pages*). É interessante destacar que alguns mecanismos de busca também fazem o processo de busca de informação diretamente de banco de dados e diretórios de informação na *web*. Tais mecanismos de busca também mantém informação em sua base de dados, sendo estas bases populadas automaticamente através dos algoritmos de *web crawler*.

2.2.2 Arquitetura de um Mecanismo de Busca

É interessante destacar a arquitetura de alto nível de um mecanismo de busca usual. Basicamente neste aspecto, um mecanismo de busca opera em uma determinada ordem. Primeiramente é necessário recuperar o conteúdo desejado, com os algoritmos específicos sendo executados pelo *web crawler*. Após recuperar tal conteúdo, é necessário indexar o mesmo afim de definir uma metodologia da estruturação da informação. Após obter o conteúdo e organizar o mesmo é possível executar o processo de busca da informação.

Na Figura 1 é possível observar de um modo geral a estrutura de um *web crawler*, com o processo de obtenção de informação da *Internet*, para que os processos de indexação e armazenamento para futura busca sejam executados.

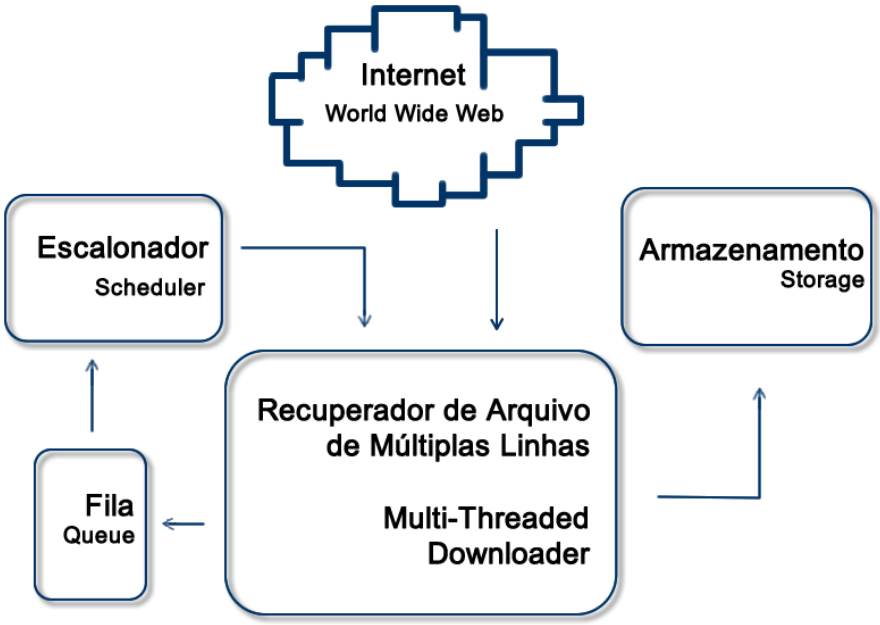

Figura 1: Estrutura Web Crawler.

De forma geral, o foco principal da maioria os processos ligados aos mecanismos de busca é o tratamento da informação. Como dito, este processo segue resumidamente por 3 etapas: obtenção da informação pelo *crawler*, indexação da informação relevante, sistema de recuperação de informação (busca).

No proceso de obtenção de informação, utiliza-se o *web crawler*, que de maneira geral, varre a *Internet* de forma sistemática na busca de conteúdo usualmente proveniente de páginas *HTML*. Com esta busca é também necessário tratar a informação a ser extraída, afim de determinar como a mesma deverá ser indexada. É interessante ressaltar que neste aspecto há também um banco de dados especial de índices relativos ao conteúdo extraído, para auxiliar no processo de busca futura. O objetivo deste banco de dados de índices é relacionando a otimização do processo de busca, visando uma recuperação mais rápida da consulta.

O processo de indexação de informação varia de acordo com cada mecanismo de busca, por exemplo, o *Google* e seu algortimo pode armazenar parte do conteúdo, ou todo o código fonte do mesmo. É possível verificar este processo pelo *Google Cache*. Já o mecanismo de busca *Altavista* armazena todo o conteúdo, ou informação das páginas

visitadas. Desta forma, a página no cache do *Altavista* detém também a palavra-chave de consulta, apesar do *overhead* de disco, este processo pode ser bastante útil quando há modificação nas páginas, e a mudança do foco das palavras-chave do texto. Estes casos de mudança de foco do conteúdo podem ser intencionais, como manobras afim de garantir visitas com termos bastante procurados.

Afim de garantir uma grande experiência do usuário, os mecanismos de busca utilizam vários algoritmos afim de determinar a qualidade dos conteúdos, e através do *cache*, podem verificar também se houve alguma estratégia intencional na manipulação da informação. É sempre interessante que o usuário encontre a informação desejada nos resultados de pesquisa, afim de garantir a qualidade do mecanismo de busca e a expectativas dos utilizadores.

Quando há interação do usuário com o mecanismo de busca, usualmente com a digitação de um termo de pesquisa, também conhecido como palavra-chave de busca, o mecanismo de busca acessa seu índice de dados e retorna uma lista contendo endereços de páginas que melhor assemelha a tal busca, de acordo com seu critério de escolha, é comum também que junto desta lista, haja também uma pequena descrição de cada página e também com seu título. O índice é formado pela informação armazenada com os dados e a metodologia utilizada para indexar a informação. Muitos mecanismos de busca da atualidade permitem a utilização de operadores *booleanos* (*and*, *or* e *not*) para otimizar a busca de informação. Operadores *booleanos* são utilizados para busca literal que permitem que o usuário refine as diferenças entre as palavras-chave.

A utilidade de uma máquina de busca é diretamente relacionada na relevância das páginas de resultados. Mesmo que existam milhões de páginas que contenham algumas frases ou palavras relevantes a busca relacionada, algumas páginas podem ser mais relevantes, populares, ou contenha conteúdo mais rico do que outras. Para garantir um método de comparação e seleção de resultados levando em consideração diversos fatores além somente da presença da palavra-chave principal da busca, os mecanismos de busca criam um *ranking* para resultados, para desta forma classificar da melhor forma para o usuário. É interessante destacar que a metodologia utilizada para classificar e ranquear as páginas para os resultados varia muito de acordo com cada mecanismo de busca. Além desta variação de algoritmos de resultados, os mecanismos de busca visam sempre atualizar os mesmos afim de evitar resultados tendenciosos, acompanhar as mudanças nos hábitos de busca e garantir cada vez mais resultados relevantes aos usuários. De forma geral, é possível citar 2 técnicas subjetivas na construção das metodologias de busca:

- Sistema de hierarquia de palavras-chave predefinido – programação manual de hierarquia de lista de palavras.
- Índice Invertido – sistema que produz uma tabela de índice invertido analisando textos e localidades.

Basicamente estes 2 sistemas se diferem na utilização de algoritmos automáticos para busca e geração dos resultados, contra formas manuais de criação de hierarquias de classificação.

2.2.3 Qualidade da Busca

O objetivo primordial das novas máquinas de busca é garantir cada vez mais a qualidade da busca no meio vólatil de crescimento constante de informação. É válido lembrar que no início não existiam mecanismos de busca eficientes, e a busca era baseada na localização de sites em diretório. Neste aspecto de evolução da qualidade de busca, acreditava-se que com um índice de busca completo seria possível encontrar qualquer informação de forma precisa e sem grandes dificuldades. Porém é verificada a evolução dos mecanismos de busca, assim como a evolução do perfil do usuário e da busca da informação. Desta forma, em 1994 havia a crença de que o melhor serviço de busca, seria aquele que retornasse qualquer tipo de informação buscada na *web*, desde que todo os dados das páginas estivessem dentro do índice de pesquisa. De certa forma, é intuitivo a excelência do serviço estar diretamento ligada a qualidade do mesmo, porém seguindo a evolução da *Internet* foi verificado nos anos seguintes que mesmo com um índice bem formado, ainda era encontrado muito conteúdo fora de escopo sendo retornado nas pesquisas. Vários fatores estão relacionados, a crescente queda da qualidade nos mecanismos de busca por volta de 1997. É interessante ressaltar, que a magnitude de crescimento do índice de pesquisas era cada vez maior, enquanto que a habilidade do usuário de diferenciar conteúdos de qualidade nos resultados de pesquisa não acompanhou o crescimento. Os usuários possuem a tendência de visualizar apenas os resultados da primeira página de resultados de pesquisa, desta forma, seria necessário novas ferramentas e metodologias para garantir resultados mais precisos (garantir grande precisão na primeira página de resultados). Obviamente, existe muitas páginas com informações relevantes aos termos de pesquisa, porém esta alta precisão desejada para formar a página de resultados também é importante no ponto de vista de infra-estrutura, afim de determinar o total de informações relevantes que o sistema é capaz de processar e retornar. Seguindo nesta etapa de otimizações, houve bastante propostas de otimizações, principalmente focada na utilização das informações hipertextuais,

texto âncora e *links* [Weiss et al. 1996] [Marchiori 1997] [Spertus 1997] [Kleinberg 1999]. Com a utilização de diversos mecanismos de otimização na busca de informação, indexação, e a relevância da *Internet* vista como um grafo [Page et al. 1999] foi possível fazer diversas constatações afim de aprimorar os mecanismos de busca, garantir um ranking das páginas, tendo assim o início dos mecanismos de busca modernos.

2.2.4 Classificação das Páginas de Resultados e PageRank

Uma grande contribuição para o desenvolvimento dos mecanismos de busca modernos foi a visualização das páginas disponíveis na *Internet* como um grande grafo direcionado. De forma sucinta, cada vértice deste grafo é constituído de uma página, e cada aresta seria o *hiperlink* de conexão de uma página para outra [Page et al. 1998]. Com esta simples inferência de *links* direcionados, aliada ao texto âncora associado ao mesmo, foi possível dar início ao projeto de cálculo de força de cada página, ou mesmo o chamado *PageRank*. Com a utlização de *PageRank* aliado a diversos outros algoritmos e inferências sobre a teoria dos grafos, foi possível tornar as pesquisas de termos-chave cada vez mais precisas.

É interessante ressaltar a justificativa intuitiva de busca quando analisamos as considerações de popularidade de uma página., ou mesmo, o seu fator *PageRank*. Desta forma, o modelo de constatação de popularidade de uma página, pode ser definido principalmente de acordo com o comportamento do usuário. De forma análoga, é possível imaginar um usuário aleatório, no qual acessa uma página *web* aleatória, e clica nos *links* das páginas, nunca utilizando o voltar, mas eventualmente pode se entediar e iniciar suas visitas em outra página aleatória. Este pensamento de "surfista aleatório da *web*"e como é definido seu comportamento, também é conhecido como o algoritmo de *PageRank* [Page et al. 1998].

Outro fator determinante de uma página com grande popularidade do ponto de vista do *PageRank*, é que esta recebe vários *links* de diversas outras páginas, ou mesmo que exista menos páginas, porém estas páginas possuem alto valor de *PageRank* e enviem *links* para ela. De forma intuitiva, páginas que são referenciadas por diversas outras páginas tem maior tendência de possuir conteúdo relevante e de qualidade, sendo assim, são interessantes e potencialmente devem ser visitadas. Em suma, uma página que possui um *link* de entrada proveniente de uma página muito popular (canal de notícias, ou portal de informações) recebe grande relevância a ser visitada, porém se esta página não possui alta qualidade, ou mesmo possui problemas em seus *links* (*links* quebrados), potencialmente nenhuma página importante irá referenciá-la. Os algoritmos que validam a popularidade das páginas possuem rotinas que testam diversos casos de relevância de

informação, atualizando os pesos dos *links* propagando recursivamente toda a estrutura de *links* na visão da *web* como um grafo.

2.2.4.1 Impacto do Texto Âncora

Os mecanismos de busca modernos possuem um foco especial em como tratar o texto âncora de um *link*. Não somente o impacto que o texto âncora possui na página em que está associado é utilizado, mas também a página alvo em que este se associa é analisado. Esta visão analítica visualizando a *Internet* como um grafo direcionado, e associando *Ranking* as páginas associadas aos *links* de entrada possui diversas vantagens.

Uma das vantagens mais específicas dos textos-âncora é a relação de precisão do termo chave da página alvo, esta precisão é muitas vezes maior que a própria descrição da página (meta description). Outro aspecto interessante sobre os textos-âncora é que muitos destes podem apontar para diversos tipos de arquivos (páginas, imagens, vídeos, banco de dados, documentos, arquivos de mídia em geral), tais arquivos ou páginas que podem ainda não ter sido indexadas, sendo possível então que as mesmas sejam alcançadas. De maneira objetiva, tais páginas que ainda não foram indexadas podem causar problemas, pois ainda não foram verificadas para que possam ser retornadas de forma satisfatória ao usuário pelo mecanismo de busca, sendo assim, neste caso, o mecanismo de busca pode retornar uma página que não existe, ou que nunca existiu (*link* quebrado). Obviamente, existe uma metodologia neste processo, para evitar que tais tipos de problemas ocorram.

É interessante ressaltar os primórdios da utilização da metodologia de utilização do texto âncora para inferências da página destino, implementada no *crawler World Wide Web Worm* [McBryan 1994]. Esta proposta mostrou também eficaz por auxiliar na busca de informações que não eram baseadas em texto, e também ampliou a cobertura de busca com uma menor necessidade de *download* de documentos.

2.3 Processamento de Imagem

O processamento de imagens preocupa-se em manipular figuras para sua representação final. O processamento de imagens é um estágio para novos processamentos de dados, como exemplo, aprendizagem de máquina, reconhecimento de padrões, ou indexação dos dados. As técnicas, em sua maioria, envolvem o tratamento da imagem como um sinal bi-dimensional.

O processamento de imagens, também é utilizado dentro do escopo de extração de características de uma imagem de *web*.

A simplificação do conjunto de dados requeridos para descrever um grande conjunto com mais decisão participa da extração de características. Quando se está executando a averiguação de dados complexos um dos maiores problemas é procedente do número de valores envolvidos. Averiguar uma série muito grande de variações numéricas requer uma grande quantidade de memória e poder de processamento ou um algoritmo de classificação estatística que sobrecarrega a amostra de treinamento e faz pouca generalização para novas amostras, fazendo com que se gere muitas séries, ou séries desnecessárias.

A extração de características é uma forma especial de se reduzir a dimensão quando se fala em processamento de imagens e reconhecimento de padrões.

Em um algoritmo, quando os dados de entrada são muito grandes e notoriamente redundantes (grande quantidade de dados e pouca informação inserida) para serem processados, deverão ser transformados em um vetor, conjunto reduzido de características melhor representativo.

Extração de características é o ato de transformar dados de entradas em um conjunto de características. Quando as características são extraídas criteriosamente espera-se que esse conjunto simbolize a parte relevante da informação para que seja executada a tarefa desejada e não usufruir dos dados de entrada na íntegra.

O alcance dos melhores resultados acontece quando é gerado, por um especialista, um conjunto de características conforme a aplicação a ser feita para estes dados; se a disponibilidade de tal conhecimento é negativa, técnicas de redução dimensional genéricas podem ajudar. Algumas delas são: análise de componentes principais, agrupamento semi-preciso, redução dimensional com múltiplos fatores, redução dimensional não-linear, isomap, análise semântica latente e análise de componentes independentes.

2.3.1 Representação de uma imagem digital

O termo imagem é definido como uma função de intensidade luminosa bidirecional, ou seja, a cada ponto do espaço representado em duas dimensões (x, y) é associado uma intensidade luminosa $f(x, y)$ finita e superior a zero [Gonzalez e Woods 2006].

Apesar disso, a representação de uma imagem em computador necessita que a imagem tenha uma quantidade de pontos com intensidade variando sobre uma faixa fixa de

valores. Afim de obter a representação, deve-se digitalizar as coordenadas espaciais (x, y) e quantizar a amplitude $f(x, y)$ de acordo com alguma representação. Assim, a representação da imagem passa a ser uma simples matriz x, y que pode ser representada na Função 2.5.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,y-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,y-1) \\ f(2,0) & f(2,1) & f(2,2) & \dots & f(2,y-1) \\ \vdots & & & & \\ f(x-1,0) & f(x-1,1) & f(x-1,2) & \dots & f(x-1,y-1) \end{bmatrix} \tag{2.5}$$

A denominação *pixel* (do inglês, *picture element*, elemento da figura) se refere a cada elemento da matriz. Geralmente, os *pixels* tem números inteiros como representação. Esses números podem corresponder ao valor do *pixel* na escala de cinza (*grayscale*), ou então, a tupla de valores que representam cores. A maneira mais comum de representar cores diferentes é o padrão RGB (do inglês, Red-Green-Blue, Vermelho-Verde-Azul). Existem outras maneiras e outras representações para manipulação de imagens coloridas ou espaços de cor, as quais podemos citar: CMYK (do inglês, Cyan-Magent-Yellow-blacK, Ciano-Magenta-Amarelo-Preto), HSV (do inglês, Hue-Saturation-Value, Tonalidade-Saturação-Valor), Y'CbCr (que indicam as características de luminância, chroma azul e chroma vermelho).

A quantidade de bits em cada pixel vai variar de acordo com a imagem. Naquelas do tipo *grayscale*, os valores mais comuns de bits são 8 ou 16. Nas imagens coloridas, a quantidade de bits é chamada de profundidade de cor e o valor varia dependendo da representação escolhida.

2.3.1.1 Espaço de cores

Quando o objeto manipulado é uma imagem colorida, existe a necessidade de uma representação lógica para as cores. Um espaço de cores é uma maneira de representar uma cor de uma forma computacional atráves de uma tupla de números. Cada número dessa tupla corresponde a um componente da cor. Alguns dos vários espaços de cor que foram desenvolvidos são detalhados a seguir.

RGB – O espaço RGB divide as cores em três componentes: uma vermelha (do inglês, *red*), uma verde (do inglês, *green*) e uma azul (do inglês, *blue*). Para a obtenção de outras cores através dessas componentes, a intensidade de cada uma delas é variada e os valores somados. Inexiste um nível pré-estabelecido das componentes vermelha, verde e azul, ou seja, padronizações diferentes vão resultar em cores diferentes mesmo que a combinação das colorantes seja a mesma.

Digitalmente, o comum é que uma cor em RGB tenha 24 bits, sendo um byte (8 bits) para cada componente. Quando a precisão de cores da foto é maior, a imagem pode ter o total de 48 bits, sendo 2 bytes (16 bits) para cada componente.

O espaço de cores RGBA é uma variação do espaço RGB. A diferença entre os dois consiste na inclusão de uma quarta componente na representação das cores, chamada de alfa, que define o nível de transparência da cor, usada nas ocasiões em que imagens sejam formadas por sobreposição de outras imagens. A quantidade de bits por componente, normalmente, é a mesma, totalizando, assim, 32 bits por cor.

CMYK – Um outro espaço de cores existente é o CYMK. As cores utilizadas como componentes são ciano (do inglês, *cyan*), magenta (do inglês, *magent*) e amarelo (do inglês, *yellow*). Além das componentes serem distintas das utilizadas no RGB, outra diferença estabelecida é a operação efetuada com as componentes para obtenção de outras cores: enquanto no RGB os valores são somados, no CYMK os valores são subtraídos. A letra K da sigla representa a cor preta (do inglês, *black*), que é usada para garantir uma melhor representação para pixels com as cores cinza ou preto, porém o preto não é considerado uma cor primária no espaço CYMK.

HSV – HSV é uma transformação do espaço de cores RGB. Ao invés de cores consideradas primárias, representa as cores pelas componentes de tonalidade (do inglês, *hue*), saturação (do inglês, *saturation*) e valor de brilho (do inglês, *value*). Seu uso é uma alternativa ao RGB. O HSV é mais comumente aplicado em ferramentas de processamento de imagem, pois permite uma mudança nos tons dos pixels de forma mais eficiente e conveniente do que variando tons de vermelho, verde e azul.

Y'CbCr – Y'CbCr é outro espaço de cores criado com o propósito de ser mais eficiente para transmissão de dados do que os outros epaços de cor. Ao se usar o RGB, uma variação de valor baixo em alguma das componentes pode resultar em uma grande alteração na

cor resultante. Em um espaço Y'CbCr, uma das componentes é equivalente digital da luminância (luma representada por Y'), a outra é o chroma azul (do inglês, *blue chroma*, representada por Cb) e croma vermelho (do inglês, *red croma*, representada por Cr). As duas últimas componentes desse espaço de cor são as diferenças das componentes azul e vermelho, do RGB, em relação à componente luma.

Uma imagem em *grayscale* tem sua representação em uma componente apenas, a Y'. As outras duas são responsáveis pela parte colorida da imagem, dão os acréscimos necessários para que outras cores sejam adicionadas a imagem. Como essas componentes croma carregam informações cujas variações são menos visíveis pelo olho humano, para compactação de dados o mais comum é subamostrar tais componentes. Esse espaço de cores é utilizado nos formatos de armazenamento JPEG e JPEG 2000 e é a versão do espaço YUV, utilizado em transmissões analógicas de tv e em sinais de vídeo componente, que foi adaptada para a digitalização de imagens.

2.3.1.2 Paleta de cores

Como descrito, as imagens usualmente são representadas como uma matriz de *pixels*. No ponto de vista de estrutura de dados e armazenamento, esta técnica não se mostra muito eficiente, ou seja, é possível utilizar de recursos computacionais afim de otimizar o armazenamento.

Uma característica encontrada na representação das imagens é que estas possuem uma paleta de cores (ou um conjunto de cores) que podem ser repetidas em diversas posições na matriz. Verificando esta tendência, foi possível verificar alguns métodos de otimização, visto que nem todas as cores do espectro de cores é utilizado. Assim, visando a otimização do armazenamento da imagem, é definido sumariamente uma paleta de cores referente a esta, definindo o conjunto de cores a serem utilizados na imagem. Em suma, esta paleta de cores é o subconjunto do espaço de cores conhecidas, no qual só contém as cores utilizadas na imagem. Para fins de utilização, a paleta de cores possui um índice específico para cada cor utilizada.

Para fins práticos, em um caso hipotético, analisando uma imagem de 256 pixels por 256 pixel, sendo definida esta no espaço RGB, com uma profundidade de cores de 24 bits, sendo 3 bytes por pixel, observamos que seu armazenamento em disco seria em torno de 192 kB. Porém, para fins te otimização, foi verificado que só é utilizado 32 cores dentro do espaço de cores RGB. Desta forma, é possível definir uma paleta de cores que contenha somente as cores a serem utilizadas na imagem, supondo que esta paleta utiliza somente

5 bits para o índice das cores. Caso esta otimização seja feita, é observado uma redução no armazenamento da imagem, sendo que a matriz definida por esta imagem ocupe em disco apenas 40kB. Desta forma, quando esta imagem fosse utilizada, é necessário que o usuário possua também a paleta de cores utilizada, para que possa acessar corretamente as definições de cores. Assim, a paleta de cores também deve ser adicionada no arquivo de imagem, e mesmo assim esta garante grande otimização, pois seu tamanho não é significativo. Neste exemplo é observado que a paleta ocupa apenas 116 bytes de memória, com o arquivo final próximo a 40,1 kB.

2.3.2 Extração de características

É interessante ressaltar o modo de representação usual das imagens, esta que é através de matrizes bidimensionais. Estas matrizes são compostas por *pixels*, estes que são representados geralmente entre 8 a 32 bits. Observa-se o grande universo amostral de imagens que é a *Internet*, e analisando as tecnologias de armazenamento atuais, é verificado que o armazenamento destas para processamento é inviável.

Afim de auxiliar neste desafio de captação dos arquivos de imagem e seu armazenamento para processamento e indexação, é necessário otimizar este processo. De maneira geral, caso seja necessário o armazenamento local, que este seja otimizado, armazenando apenas as informações primordiais, e que a partir destas, seja possível criar uma representação da original.

O processo de otimização na extração de atributos e características relevantes das imagens é conhecido como extração de características, e possui algoritmos próprios que utilizam técnicas para compactação de dados e transformadas de funções.

2.3.2.1 Transformada discreta do cosseno

A técnica de transformada discreta do cosseno (DCT) [Watson 1994] utiliza a representação da imagem a partir da soma de cossenos com diferentes magnitudes e frequências. É verificado que os *pixels* de uma determinada imagem possui certa correlação com seus vizinhos em ambas as dimensões e não apenas com uma dimensão, sendo assim a DCT que é utilizada na otimização e compressão de imagens deve ser bidimensional. A seguir, a Equação 2.6 destaca a DCT bidimensional para uma imagem $\mathbf{I}$ com tamanho $w \times w$:

$$D_{xy} = \frac{1}{\sqrt{2w}} T_x T_y \sum_{i=0}^{w-1} \sum_{j=0}^{w-1} I_{ij} \cos\left(\frac{y(2j+1)\pi}{2w}\right) \cos\left(\frac{x(2i+1)\pi}{2w}\right), para 0 \leq x, y \leq w-1 \tag{2.6}$$

onde $T_a = \begin{cases} 1, & a > 0 \\ \frac{1}{\sqrt{2}}, & a = 0 \end{cases}$

A transformada discreta do cosseno é também considerada como uma rotação, ou mesmo duas rotações seguidas em dimensões distintas. A DCT é considerada também como uma base ortogonal dentro de um espaço vetorial com w dimensões. Como descrito, é necessário a recuperação dos dados iniciais da imagens, afim de criar uma representação fiel da mesma. Tal recuperação dos dados da imagem pode ser feita utilizando a transformada inversa, também conhecida como IDCT bidimensional. É possível verificar a representação da mesma na Equação 2.7 a seguir:

$$I_{ij} = \frac{1}{4} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} T_x T_y D_{xy} \cos\left(\frac{x(2i+1)\pi}{2w}\right) \cos\left(\frac{y(2j+1)\pi}{2w}\right) \tag{2.7}$$

De forma análoga, a transformada bidimensional tem por resultado uma matriz no qual os coeficientes mais relevantes são agrupados nos índices iniciais (começo da matriz). Desta forma, os demais coeficientes possuem pequenos valores, podendo ser quantizados afim de otimizar a compressão de perdas.

2.3.2.2 Transformada discreta de Wavelet

Uma função característica de *Wavelet* deve possuir as características a seguir [Mallat 1989]:

1. O total da área sob a curva da função é 0, ou seja $\int_{-\infty}^{\infty} \psi(t)dt = 0$
2. A energia da função é finita, ou seja $\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty$

Tais condições garantem que $\psi(t)$ é quadrado integrável ou que pertence ao conjunto $\mathcal{L}^2(R)$ das funções quadrado integráveis. Desta forma, as características e propriedades sugerem que $\psi(t)$ possui a tendência a oscilar acima e abaixo do eixo t, e que tem sua energia posicionada em uma determinada região, pois esta é finita.

É interessante destacar a necessidade da transformada inversa de *Wavelet*, para que a função *Wavelet* possa ser utilizada com êxito na análise de sinais.

Há diversos tipos de funções *Wavelets* explorados na literatura [Stollnitz, Derose e Salesin 1995, Shnaider e Papliriski 1996]. A finalidade e utilização destas é totalmente relacionada a sua relevância diante a aplicação. Há diversas regras e metodologias associadas a construção de *Wavelets*, estas propostas por diversos centros acadêmicos. Obviamente há certas características, garantindo um limiar de restrições relacionadas as aplicações específicas.

É interessante destacar a *Wavelet* de *Daubechies* [Daubechies 1992], com características favoráveis ao suporte compacto e suavidade parametrizável, esta *Wavelet* que possui uma capacidade de síntese e análise mais efetiva comparada as *Wavelet* de *Haar* [Haar 1910] devido principalmente por tal característica de suavidade já mencionada e pela qualidade na aproximação de funções em $\mathcal{L}^2(R)$. Sendo assim, estas também possuem ótimos resultados na compressão devido as seguintes características:

- ψ_r ter suporte compacto no intervalo $[0, 2r+1]$,
- ψ_r ter $\frac{r}{5}$ derivadas contínuas,
- $\int_{-\infty}^{\infty} \psi_r(x)dx = \ldots = \int_{-\infty}^{\infty} x^r \psi_r(x)dx = 0$

Onde ψ_r é a *Wavelet* mãe e $r \in Z$.

$$\frac{\sqrt{3}+1}{4\sqrt{2}}, \sqrt{3}\frac{\sqrt{3}+1}{4\sqrt{2}}, \frac{\sqrt{3}-1}{4\sqrt{2}}, \sqrt{3}\frac{\sqrt{3}-1}{4\sqrt{2}} \tag{2.8}$$

$$-\sqrt{3}\frac{\sqrt{3}+1}{4\sqrt{2}}, -\frac{\sqrt{3}-1}{4\sqrt{2}}, \sqrt{3}\frac{\sqrt{3}+1}{4\sqrt{2}}, -\frac{\sqrt{3}+1}{4\sqrt{2}} \tag{2.9}$$

É possível verificar diante experimentos as diferenças e peculiaridades diante as transformadas *Wavelet* de *Daubechies*. Estas quando parametrizadas $r > 2$ apresentam um melhor agrupamento de energia, sendo assim garantindo uma maior preservação de informação de tendência nos sinais, ao levar em consideração apenas o filtro passa-baixa da Equação 2.8 do que o filtro de passa-alta da Equação 2.9.

A função *Haar*, conhecida também como *Wavelet* de *Haar*, é considerada como *Daubechies v2*, sendo esta a primeira função *Wavelet* com um único momento nulo.

Uma característica das funções de *Haar* é que estas são de passos descontínos, sendo assim não muito favoráveis na análise de funções estáveis com derivações contínuas. Sendo assim, de forma geral não é usualmente verificado resultados otimizados e satisfatórios utilizando a *Wavelet* de *Haar*, pois é comum encontrar nas imagens tais regiões estáveis.

2.3.3 Reconhecimento de Faces

A detecção e reconhecimento de faces é uma área de processamento de imagens que propõe muitos desafios e oportunidades. Um sistema projetado afim de armazenar, agrupar e classificar faces do mundo real deve ser robusto o suficiente afim de suportar um grande número de indíviduos em seu banco de dados de pesquisa. Outras oportunidades também são encontradas em sistemas de busca de faces na *web*, afim de serem utilizadas em diversos escopos, como fotos com direito autoral, pessoas desaparecidas, entre outros. Estes sistemas podem optar em utilizar apenas as comparação de faces, não sendo necessário um banco de dados robusto para armazenamento, pois a busca, será somente nas imagens oriundas da *Internet*.

É interessante destacar o custo computacional necessário para identificar e acrescentar novas faces ao banco de dados. De forma objetiva é utilizado o algoritmo "Principal Component Analysis", sendo utilizado todo o banco de dados afim de processar a necessidade de inserção de nova face.

Existem quatro tarefas primordiais que são relevantes na identificação das faces [Gong, McKenna e Psarrou 2000], destacadas:

1. **Classificar:** etapa com objetivo de identificar uma dada face x partindo do pressuposto que esta pertence a uma pessoa do conjunto Ω. Assim, é possível assumir que x pode ser identificado e classificado com o padrão de uma classe ω_i, de forma que $\omega_i \in \Omega$, o processo de classificação é consistido em calcular o valor de i.

2. **Conhecer:** esta etapa visa testar a face afim de determinar se esta pertence ou não ao grupo Ω, ou seja, se é possível classificar a face x com um padrão de uma classe de Ω.

3. **Verificar:** dada uma face dado x no qual a identidade ω_i foi verificada por um meio meio não visual, esta etapa tem por objetivo determinar a identidade desta pessoa utilizando imagens de face, em suma, determinar se x é da classe ω_i. Este processo equivale ao item "Conhecer", com $c = 1$.

4. **Validar:** tem por objetivo validar se uma determinada face pertence a classe de Ω, caso seja verificado, determinar sua identidade ω_i.

O grande desafio em agrupar faces em categorias é destacado na classificação de algumas características, como o gênero, idade, raça, entre outros. Neste aspecto de classificação, as classes definidas representam diretamente as categorias nas quais os indivíduos

pertencem. Assim, esta etapa é equivalente a "Classificar", com c representando o universo amostral de categorias de todo o problema a ser abordado.

Os processos de reconhecimento de características relativas as expressões faciais, utilizam variáveis e parâmetros que visam explorar pontos que não possuem influência direta por alterações da estrutura facial (forma da boca, dos olhos, contorno facial). É verificado a utilização de métodos parecidos ao de categorização de faces, no qual na etapa de treinamento e classificação são utilizados de forma que cada classe definida represente um tipo de expressão facial.

Outro ponto importante e primordial é a distinção entre imagens que contém faces, de outras que não as possuem. Este problema está relacionado a duas classes, a classe de faces, e a classe de não faces, esta que pertence ao escopo da detecção de faces.

Os diversos métodos utilizados para detecção de faces podem ser classificados nas categorias abaixo:

- **Atributos (locais)**: utilizam regras básicas de construção de vetor de características a partir de características da face e medidas de distância e ângulos entre pontos específicos [Cox, Ghosn e Yianilos 1995]. São aplicadas, também a imagens de faces de perfil. O uso de informações não disponíveis em imagens frontais bidimensionais é a principal vantagem do reconhecimento a partir de perfis. O reconhecimento de faces por atributos, é através de um sistema de reconhecimento de padrões, podem ser estatístico ou por redes neurais.

- **Holísticas:** abrangem todos os *pixels* da imagem ou de regiões características da face. O número de *pixels* das imagens consideradas é igual a dimensão dos dados. Para esquivar de problemas referentes a dimensão, pode-se utilizar métodos estatísticos de redução de dimensionalidade, como por exemplo, Análise de Componentes principais (PCA) [Turk e Pentland 1991], discriminantes lineares
  [Belhumeur, Hespanha e Kriegman 1997] e redes neurais [Lauwrence et al. 1996]. O PCA é o método mais popular, e para melhorar o desempenho tem sido frequentemente utilizado em associação com pré-processamentos de normalização de imagens.

- **Técnica baseada na transformada de Gabor:** é uma técnica muito promissora e mais recente. Pode-se utilizar um filtro linear, chamado de filtro de Gabor, cujo impulso de resposta é dado por uma função harmônica multiplicada por uma função Gaussiana. Por causa da propriedade de convolução-multiplicação, proveniente do teorema de Convolução, a transformada de Fourier de um impulso resposta do filtro

de Gabor é a convolução da transformada de Fourier da função harmônica e a transformada de Fourier da função Gaussiana.

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{2.10}$$

onde $x' = x\cos\theta + y\sin\theta$, e $y' = -x\sin\theta + y\cos\theta$. Nesta equação λ representa o comprimento da onda do coseno; θ representa a orientação do normal para a faixa paralela de uma função de Gabor; ψ é a fase do conjunto; σ é o taxa espacial e especifica a elipticidade do suporte da função Gabor.

- **Tridimensionais:** são muitos os métodos de reconhecimento relacionados as informações tridimensionais, a vantagem desses métodos existe na possibilidade de aquisição de informações relevantes da cena real que não podem ser adquiridas através de imagens bidimensionais. As informações tridimensionais combinam informações da profundidade das superfícies e também da textura.

2.3.3.1 Reconhecimento de Faces usando OpenCV

Originalmente desenvolvida pela Intel em 2000, a OpenCV (*Open Source Computer Vision Library*) é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão Computacional, bastando seguir o modelo de licença da BSD Intel. A OpenCV possui módulos de Processamento de Imagens e Vídeo I/O, Estrutura de dados, álgebra Linear e GUI (Interface Gráfica do Usuário).

Na OpenCV estão definidos mais de 350 algoritmos de Visão Computacional como filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. É utilizado o detector de faces chamado *Haar Cascade Classifier* (classificador em cascata do tipo *Haar*). Para a utilização do classificador é necessária uma escala fixa para o tamanho da face na imagem, dada em pixels. O classificador foi treinado com um conjunto de imagens da mesma escala que contém um objeto de interesse, chamadas de positivas, além de imagens arbitrárias, chamadas negativas. Depois de treinado, o classificador é capaz de dizer se certa região de uma imagem é susceptível de mostrar o objeto para o qual ele foi treinado.

A função na OpenCV que detecta um determinado objeto previamente treinado em uma imagem é a `cvHaarDetectObjects()`. Esta função necessita de sete parâmetros. Os

três primeiros são o ponteiro para a imagem, os dados em XML e o *buffer* de memória. O quarto parâmetro da função especifica a velocidade que deverá ser incrementada a escala para o reconhecimento das faces a cada vez que a imagem é percorrida. Para executar mais rápida deve se colocar valores mais altos (isto pode fazer com que algumas faces escapem do resultado).

Durante a busca das regiões candidatas ocorrem muitas sobreposições de regiões que representam o mesmo objeto com escalas diferentes. Quando a função gera a lista de retângulos candidatos, geralmente regiões isoladas (sem sobreposições) são detecções falsas, portanto são descartadas pelo detector. O problema com as múltiplas detecções, então, é resolvido mesclando estas em uma única região para cada face, primeiramente agrupando os retângulos sobrepostos e em seguida substituindo todos eles por um retângulo médio. A quantidade de retângulos que serão mesclados é definida pelo quinto parâmetro da função (*Minimum Neighbors Threshold*), logo, se o número de retângulos sobrepostos é maior ou igual a este valor, eles são mesclados, senão são descartados. Caso seja 0, será retornada a lista completa de detecções do classificador *Haar*.

Na detecção também podem ser aplicadas algumas heurísticas para reduzir o numero de regiões analisadas, como *Canny Prunning*, por exemplo, representada pelo sexto parâmetro da função que pode ser 0 ou *CV_HAAR_DO_CANNY_PRUNING*, em que o detector pula algumas regiões improváveis de conter uma face, eliminando, assim, algumas detecções falsas. As regiões a pular são identificadas executando um reconhecimento de bordas (*Canny edge detector*) sobre a imagem antes do detector de faces.

O sétimo parâmetro chamado escala mínima de detecção é o menor tamanho de face que será buscada. O valor padrão para este parâmetro pode ser visto no arquivo XML utilizado, ele está localizado entre os identificadores `<size>` e `</size>`.

As faces detectadas são armazenadas como uma lista de ponteiros da estrutura `CvRect`, e representam um retângulo dentro da imagem.

Existem também funções para selecionar uma região de interesse da imagem, chamada `cvSetImageROI()`, para salvar uma região de interesse num arquivo.

O detector da OpenCV na verdade utiliza um método que Paul Viola e Michael Jones publicaram em 2001 [Viola e Jones 2001]. Usualmente chamado de método Viola-Jones, que visa detectar objetos em imagens combinando quatro conceitos chave:

- Simples feição retangular, chamado feição tipo *Haar*

- Uma imagem que passou por um processo de integração, para uma rápida detecção de feições (*Integral Image*)
- O método *AdaBoost* de aprendizagem automática
- Um classificador em cascata para combinar várias feições de forma eficiente.

As feições que Viola e Jones usaram são baseadas nas *wavelets* de *Haar*. Estas são ondas quadradas que representam os comprimentos de onda únicos (um para o intervalo alto e outro para o intervalo baixo). Em duas dimensões, uma onda quadrada é um par de retângulos adjacentes - um claro e outro escuro, como pode ser visto na figura 2.

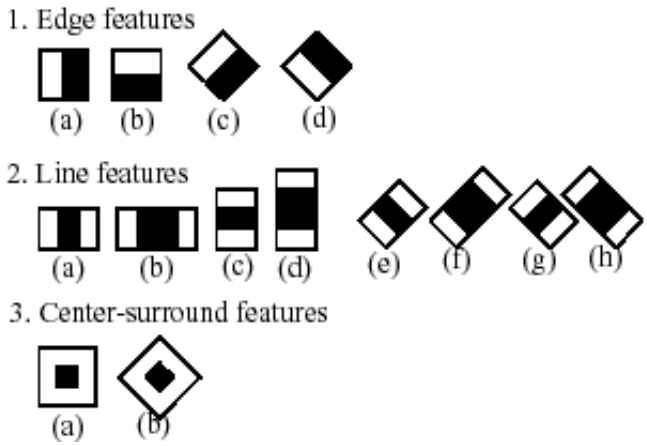

Figura 2: Feições do tipo Haar para detecção de objetos

As combinações de retângulos usadas para a detecção de objetos na verdade não são realmente *wavelets* de *Haar*. Ao invés disto, elas contêm combinações mais apropriadas para as tarefas de reconhecimento visual. Por causa dessa diferença, estas feições são chamadas de feições tipo *Haar*, que é melhor do que *wavelets* de *Haar*.

A presença de uma feição tipo *Haar* é determinada subtraindo o valor médio de pixel da região escura do valor médio de pixel da região clara. Se a diferença está acima de um limite (informado durante o aprendizado), essa feição é dita como presente.

Para determinar se ocorre à presença de milhares de feições do tipo *Haar* em cada posição da imagem e em várias escalas de forma eficiente, Viola e Jones usaram uma técnica chamada *Integral Image*. No geral, integrar significa somar pequenas unidades; neste caso, as pequenas unidades são os valores dos pixels. O valor da integral para cada pixel é a soma de todos os pixels sobre ele e à esquerda dele. Começando da parte superior esquerda e atravessando até a parte inferior direita, toda a imagem pode ser integrada com poucas operações com números inteiros por pixel.

Para selecionar uma feição tipo *Haar* específica e informar os valores do limites do método, Viola e Jones usaram um método de aprendizado automático chamado *AdaBoost*. Este combina muitos classificadores "fracos"para criar um classificador "forte". "Fraco"significa que o classificador gera um número de respostas corretas um pouco maior que o número de respostas aleatórias gerado. Logo, se tivermos muitos destes classificadores fracos combinados para chegar à solução correta nós criamos um classificador forte. O *AdaBoost* possui vários classificadores fracos combinados e associa um peso a cada um deles, o que gera seu classificador forte.

O limite aceitável em cada fase é fixado baixo para passar por quase todos os exemplos de faces do conjunto de treinamento. Os filtros em cada fase são treinados para classificar as imagens de treino que passaram por todos os estágios anteriores (o conjunto de treinamento é uma grande base de dados de faces). Durante o uso, se algum destes filtros falhar ao passar por uma região da imagem, ele vai para o próximo filtro da cadeia. As regiões das imagens que passam por todos os filtros na cadeia são classificados como "Face". Viola e Jones chamaram esta cadeia de filtragem de cascata.

A ordem dos filtros nesta cascata é baseada na importância dos pesos que o *AdaBoost* associa. Os filtros com maior peso vêm primeiro, para eliminar da imagem as regiões não face o mais rápido possível.

2.3.3.2 Reconhecimento de Faces Usando OpenIMAJ

O projeto OpenIMAJ possui um conjunto de classes que contém implementações de algoritmos no estado de arte de detecção de faces. Tais classes são provenientes como um sub-projeto chamado faces. O tipo de arquétipo do Maven do OpenIMAJ adiciona a biblioteca de face como uma dependência, assim desta forma é possível iniciar o desenvolvimento de programas para detecção de faces rapidamente de forma intuitiva.

A detecção de faces pode ser oriunda de um conjunto de imagens, ou mesmo de arquivos de vídeo. Como um vídeo é um conjunto de imagens por segundo, a abstração na utilização de ambas soluções é simples. Desta forma, para detecção de faces em imagens a interface FaceDetector provê uma API para a detecção de faces. De forma sucinta, o detector HaarCascadeDetector é considerado ser mais robusto no âmbito de detecção de faces, sendo o padrão a ser utilizado na detecção. Exemplo de inicialização:

```
HaarCascadeDetector faceDetector = new HaarCascadeDetector();
```

Assim como a maioria das implementações, o detector possui um método denominado

detectedFaces(), utilizado para identificar as faces encontradas. Outro ponto a ser considerado é devido a necessidade de análise de imagens com apenas 1 faixa (escala de cinza), desta forma é necessário converter todas as imagens multi-faixas (coloridas), para que o classificador consiga ser executado com êxito. Para manipulações de imagens é possível utilizar os métodos da classe ImageUtilities().

O método detectFaces() retorna uma lista com todos os objetos DetectedFace encontradas na imagem selecionada. A partir destes objetos é possível extrair os retângulos em volta das faces identificadas, desta forma, é possível extrair as faces e salvá-las em disco, ou executar qualquer manipulação nas mesmas quando necessário.

Há diversos algoritmos para detecção de faces que vão muito além em apenas encontrar faces na imagem. Há possibilidade de extrair características específicas como contorno de rosto, encontrar os olhos, identificar a boca dos indivíduos, entre outros. E definição de utilizar ou não tais classificadores depende também do poder computacional disponível para a aplicação e a real necessidade de utilização.

3 Aquarela Web Crawler

A proposta primordial do projeto é o desenvolvimento de um *crawler* de faces de imagens de *web* a ser utilizado para busca por similaridade. No desenvolvimento do projeto, optamos por estender funcionalidades e bibliotecas já disponíveis e *open source*.

Há diversos *crawlers open source* disponíveis no mercado. Por motivos de escalabilidade, e experiência em programação houve o interesse em trabalhar com *crawler* nas linguagens C e C++ ou Java. Os *crawlers* que atendiam a esses requisitos foram o Larbin (escrito na linguagem C e C++) e o Crawler4J (escrito na linguagem Java). Com a experiência com demais linguagens e ferramentas disponíveis, é sempre interessante garantir que o *software* fosse independente de plataforma.

Na utilização e testes com algumas soluções *open source* disponíveis, houve sempre problemas relativos ao gerenciamento correto das *threads*, afim de garantir qualidade na busca, para evitar redundância de informação. Houve extenso trabalho de otimização para verificação da necessidade de criação de um semáforo nas requisições entre as *threads*. Testes com alguns *crawlers*, apresentavam falhas de concorrência nas solicitações, problemas de detecção de faces em imagens incompletas. Desta forma, houve sempre comprometimento em garantir a extração de faces de forma satisfatória, com um *crawler* leve, porém robusto.

Após diversos testes de performance, programação e análise de código fonte, ficou definido a utilização de um *crawler* em Java, neste caso, o Crawler4J.

Além disso, houve a necessidade de definir uma biblioteca de análises gráficas. Neste ponto, foram estudadas as bibliotecas ou projetos gráficos conhecidas como OpenCV e OpenIMAJ. Ambas bibliotecas apresentavam implementações semelhantes de algoritmos de reconhecimento de faces utilizando a *Wavelet de Haar*, porém o projeto OpenIMAJ garantiu vantagens diante a compilação instalação e integração com o *web crawler*.

3.1 Arquitetura

A arquitetura proposta, que pode ser visualizada na Figura 3, é constituída por 3 módulos, sendo o primeiro módulo o projeto desenvolvido nesta dissertação.

De forma sucinta, o primeiro módulo de *Crawler* é responsável por todas as tarefas de aquisição de dados oriundos da *Internet*. É um *web crawler* robusto, e otimizado de forma a verificar imagens na *Internet*, detectar faces e salvá-las em disco. Todo o processo de detecção, análise e tratamento da imagem é feito em memória principal, o disco rígido somente é utilizado para salvar as faces detectadas. Uma característica adicional desse módulo é a operação de depuração, que salva todas as imagens detectadas, para testes estatísticos sobre falsos positivos e negativos.

O segundo módulo proposto, referente a Máquina de Busca, é responsável em organizar as características das imagens em uma estrutura de dados métrica afim de otimizar as consultas. Desta forma, há uma aplicação *web*, que recebe solicitações da *Internet* sobre faces a serem pesquisadas, e retorna uma página de resultados ao usuário.

O terceiro módulo proposto, referente a Interface com Cliente, é diretamente ligado na interação do usuário com a solução de busca. Este módulo é responsável pela interface final com o usuário, o extrator de características no lado do cliente, e a comunicação com a ferramenta.

3.2 Crawler

O módulo *crawler* é composto pela solução de aquisição de dados na *Internet*. Este módulo foi chamado de Aquarela *Web Crawler* (*Crawler* de Faces na *Web*). No seu desenvolvimento, houve a preocupação em otimizar o suficiente para gerenciar todas as filas de solicitações de *URL* juntamente com todo o processo de análise gráfica das imagens encontradas.

O *crawler* sofreu diversas modificações e refinamentos para direcionar as buscas para imagens potencialmente referentes a faces. Desta forma, foi necessário também a criação de um filtro, para garantir a aquisição apenas de imagens dos padrões e tipos conhecidos.

O módulo de *Crawler* é composto por um interpretador HTML e um detector de faces. Como pode ser observado na Figura 4, o processo inicia com uma requisição de uma URL que pode ser uma página HTML ou uma imagem (binária). Se a URL é uma

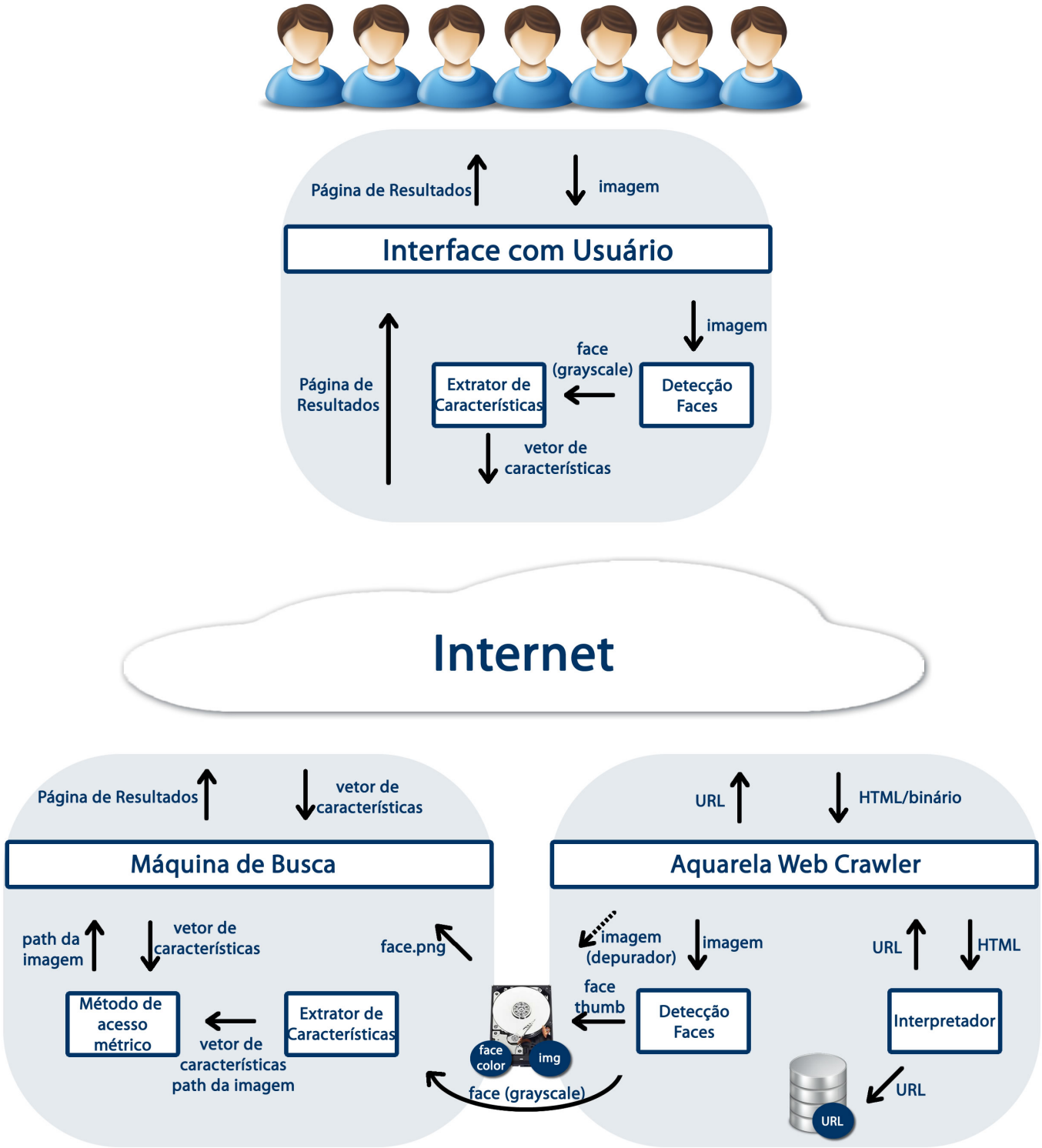

Figura 3: Arquitetura Aquarela Web Crawler para Busca por Similaridade.

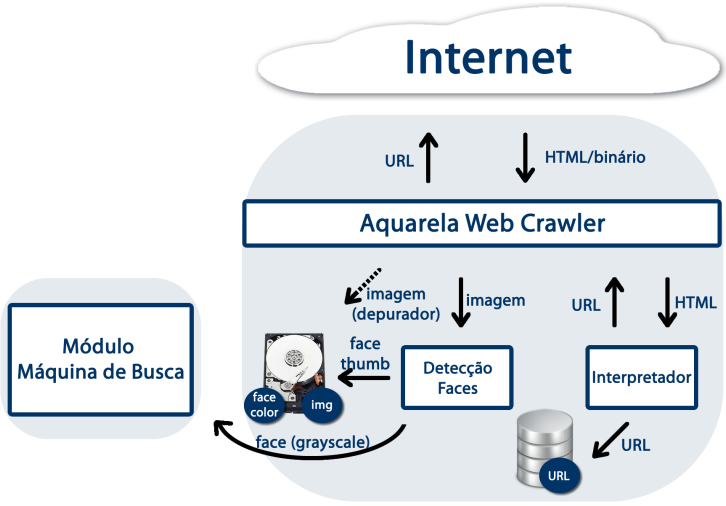

Figura 4: Módulo 1 - Aquarela Web Crawler

página HTML, o interpretador procura nestas outras URLs que podem ser novas páginas HTML ou imagens. Se a URL é uma imagem, a detecção de faces procura nestas faces que serão armazenadas no formato PNG. Se o modo de depuração esta ativo, uma cópia da imagem que contém faces é salva em disco.

Seguindo na linha de garantir excelência no processo de extração de faces de imagens oriundas da *Internet*, houve a preocupação em armazenar em disco apenas os dados pertinentes. Desta forma, o processo de aquisição de imagem da *web* é mantido em memória principal. Uma característica na execução do detector de face na imagem, é que este necessita que esta esteja composta em escalas de cinza para poder executar com êxito o processo de detecção. Desta forma, detectada ou não a face na imagem, é necessário que a mesma seja armazenada em seu formato original (na maioria das vezes colorida) em disco. A solução então, era detectar o retângulo referente as faces em uma imagem em escalas de cinza, e utilizando as mesmas coordenadas, recortar a face na imagem colorida, para garantir o armazenamento das faces em seu formato original.

Um aspecto extremamente relevante no processo de armazenamento é a definição da nomeclatura da estrutura dos arquivos. Desta forma, foi desenvolvido um sistema parametrizável para segmentar os diretórios e os nomes de arquivos. Com a parametrização era possível definir um número máximo de imagens em um diretório, assim como a metodologia dos nomes de arquivos. Desta forma, é possível destacar a metodologia citada:

- Restringido limite de 2000 imagens por diretório.
- Nomes de arquivos compostos pela concatenação da url da imagem encriptada com md5.

- A imagem original possui sufixo entre colchetes com o número de faces encontradas.
- A imagem de face possui sufixo entre colchetes indicando a ordem de reconhecimento de sua face.
- As imagens originais (módulo depurador) são armazenadas no diretório /img
- As imagens de faces são armazenadas no diretório /faces

Exemplo do nome de arquivo de uma imagem composta por $x = 20$ faces salva em disco (módulo depurador).

```
0a8ec0ae6ff3dc3609199acf6f97c916[20].jpg
```

Utilizando metodologia semelhante, as faces geradas do arquivo eram armazenadas com o nome composto pela concatenação da url da imagem original encriptada com md5, juntamente com a ordem de identificação da mesma na imagem. Assim, na pasta correspondente ao armazenamento das imagens de faces, os arquivos gerados são como o exemplo:

```
0a8ec0ae6ff3dc3609199acf6f97c916[0].jpg
0a8ec0ae6ff3dc3609199acf6f97c916[1].jpg
...
0a8ec0ae6ff3dc3609199acf6f97c916[20].jpg
```

3.2.1 Crawler4J

No desenvolvimento do projeto, foi constatado que o Crawler4J atendia positivamente várias necessidades. Primeiramente, por ser desenvolvido na linguagem Java, pela experiência já adquirida na programação da mesma, e também pelas vantagens em escalabilidade. Obviamente que há outras linguagens e projetos que possuem vantagen em escalabilidade, porém o Crawler4J destacou também pela qualidade de código fonte, pelas diversas opções de parametrização, e as demais ferramentas gráficas disponíveis no mercado que garantem suporte a linguagem Java. Neste aspecto, há ganhos significativos em velocidade de desenvolvimento e integração.

O *Web Crawler* Crawler4J é um projeto desenvolvido por Yasser Ganjisaffar sob a licença Apache 2.0. É caracterizado por ser um *web crawler* leve, rápido e utiliza o Banco de Dados de Berkeley para gerenciar a fila de entrada de forma eficiente. Em uma de

suas primeiras versões foi capaz de crawlear a versão da Wikipedia em inglês em apenas 10 horas.

Para iniciar o processo de customização do *crawler*, primeiramente foi necessário fazer o download dos fontes diretamente na página de desenvolvimento do projeto: http://code.google.com/p/crawler4j/

Nesta etapa, em paralelo já estava definido a utilização do projeto OpenIMAJ para gerenciar a identificação das faces. Nesta etapa, é interessante destacar a decisão em utilizar a IDE Netbeans, sendo assim necessário exportar o projeto OpenIMAJ. A versão da IDE NETBEANS utilizada é a 7.1, juntamente com o Java SDK 1.6.

Ao analisar o código fonte do projeto Crawler4J, além dos manuais de utilização disponíveis, foi verificada a necessidade de criação de classes específicas para sua utilização. De forma objetiva, foi então desenvolvida uma classe abstrata de *WebCrawler*, como é possível verificar na Figura 5 para gerenciar as páginas para *download* e a metodologia das *URLs* a serem visitadas. Foi desenvolvida também uma classe específica para encriptação de dados. Há também a necessidade de criação de uma classe *Controller*, com os parâmetros de semente do *crawler*, configurações de destino para salvar os arquivos e a definição de diversos parâmetros do *crawler*, assim como quantas *threads* concorrentes devem ser iniciadas.

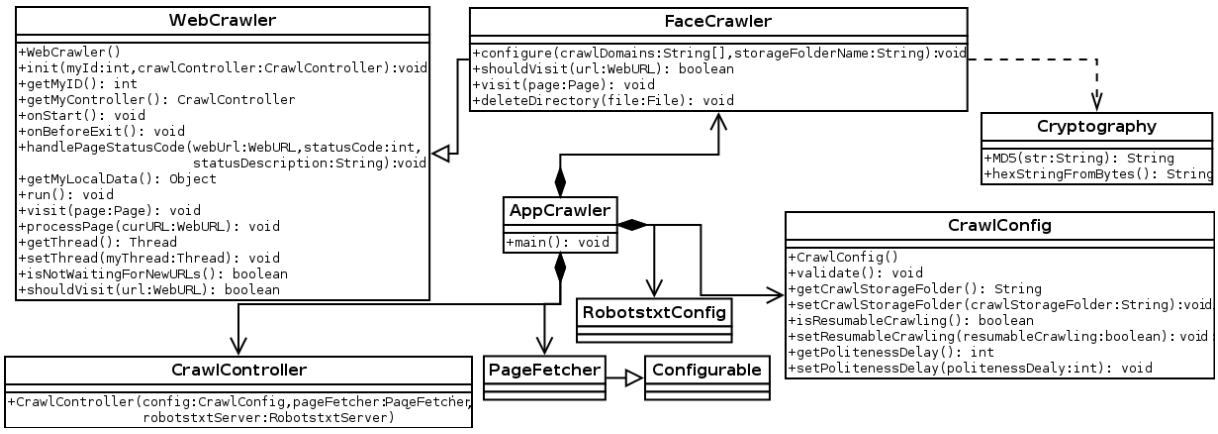

Figura 5: UML - Aquarela Web Crawler

3.2.2 OpenIMAJ

O projeto OpenIMAJ é composto por um conjunto de bibliotecas e ferramentas gráficas para análise multimídia. Desenvolvido na linguagem de programação Java, tendo então a característica de ser independente a plataforma. Desta forma, possui suporte em

diversos sistemas operacionais, como Linux, Windows e OSX. Há também a possibilidade de executá-lo em mobile, com Android.

Testes efetuados entre as bibliotecas gráficas existentes mostraram grandes dificuldades na definição do sistema operacional ideal, e das necessidades para correto funcionamento. Neste ponto, as bibliotecas gráficas OpenIMAJ garantiram resultados extremamente satisfatórios perante as bibliotecas OpenCV.

Uma característica interessante do OpenIMAJ é sua concepção entre módulos. Tais módulos podem ser utilizados de forma independente de acordo com a necessidade, como a utilização ou não de módulo de tratamento de imagens, ou demais recursos de processamento de vídeo.

Como pode ser verificado na Figura 6, os módulos do OpenIMAJ utilizados nesse trabalho foram:

- Core-feature – que define características de todos os tipos primitivos.
- Core-image – que define imagens, pixels e componentes conectados.
- Image-processing – que implementa diversos algoritmos de tratamento de imagem, pixel e componentes conectados (redimensionamento, convolução, detecção de borda, ...).
- Image-extraction – que implementa métodos para extração de características locais. Tais características são descrições de regiões de imagem selecionadas por detectores.
- Image-local-features – que implementa métodos para extração de características locais. Tais características são descrições de regiões da imagem selecionadas por detectores.
- Image-faces – que implementa recursos para reconhecimento facial, inclui diversas funcionalidades de extração e reconhecimento de características.

3.3 Máquina de Busca

O módulo de busca proposto na arquitetura desse projeto é referente a máquina de busca. Neste módulo há o extrator de características a ser utilizado em todas as faces armazenadas pelo *crawler*. Com a extração de características, um método de acesso

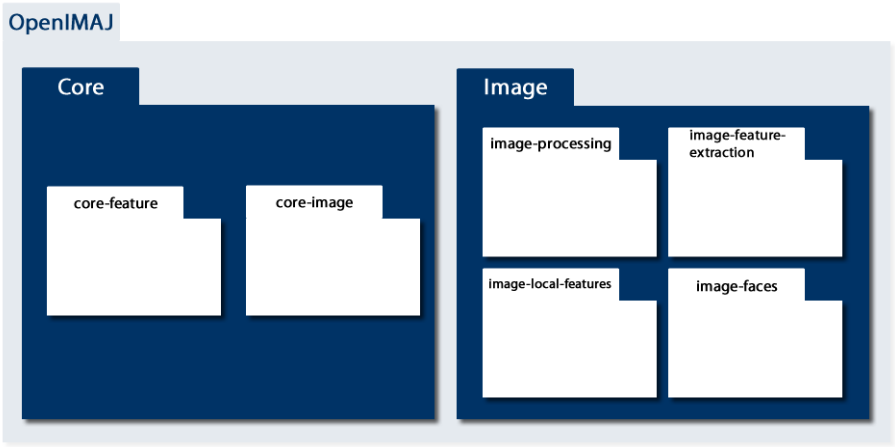

Figura 6: Módulos utilizados do OpenIMAJ

métrico e uma aplicação *web*. Como pode ser visto na Figura 7 esse módulo lê uma imagem de face do disco e extrai suas características. Essas características serão inseridas em uma estrutura de dados juntamente com o caminho do arquivo de face e a URL da imagem original.

Juntamente neste processo há uma aplicação *web* que aceita solicitações de usuários que desejam efetuar buscas referentes as imagens de face.

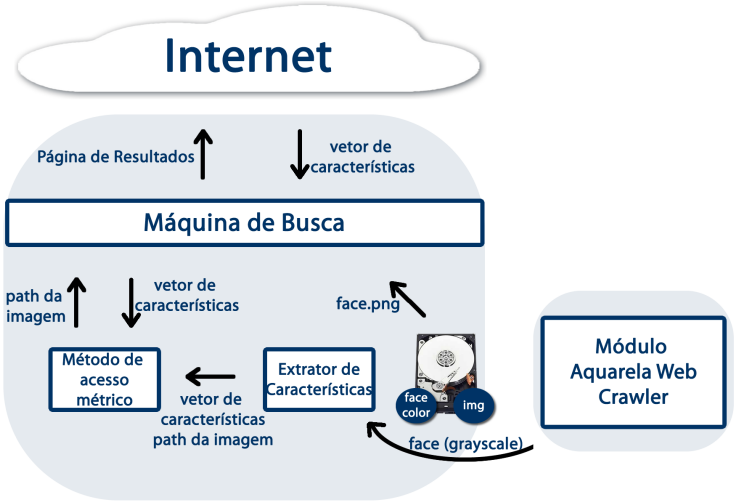

Figura 7: Módulo 2 - Máquina de Busca

3.4 Interface com Cliente

O terceiro módulo proposto, visto na Figura 8, é a interface final com o cliente. Visualizando o projeto como um serviço, o usuário tem a possibilidade de escolher uma foto de face localmente em seu computador. Esse módulo extrai as características da face

no lado do cliente e as envia em uma requisição à máquina de busca (módulo anterior). A máquina de busca responde com a página de resultados para a pesquisa.

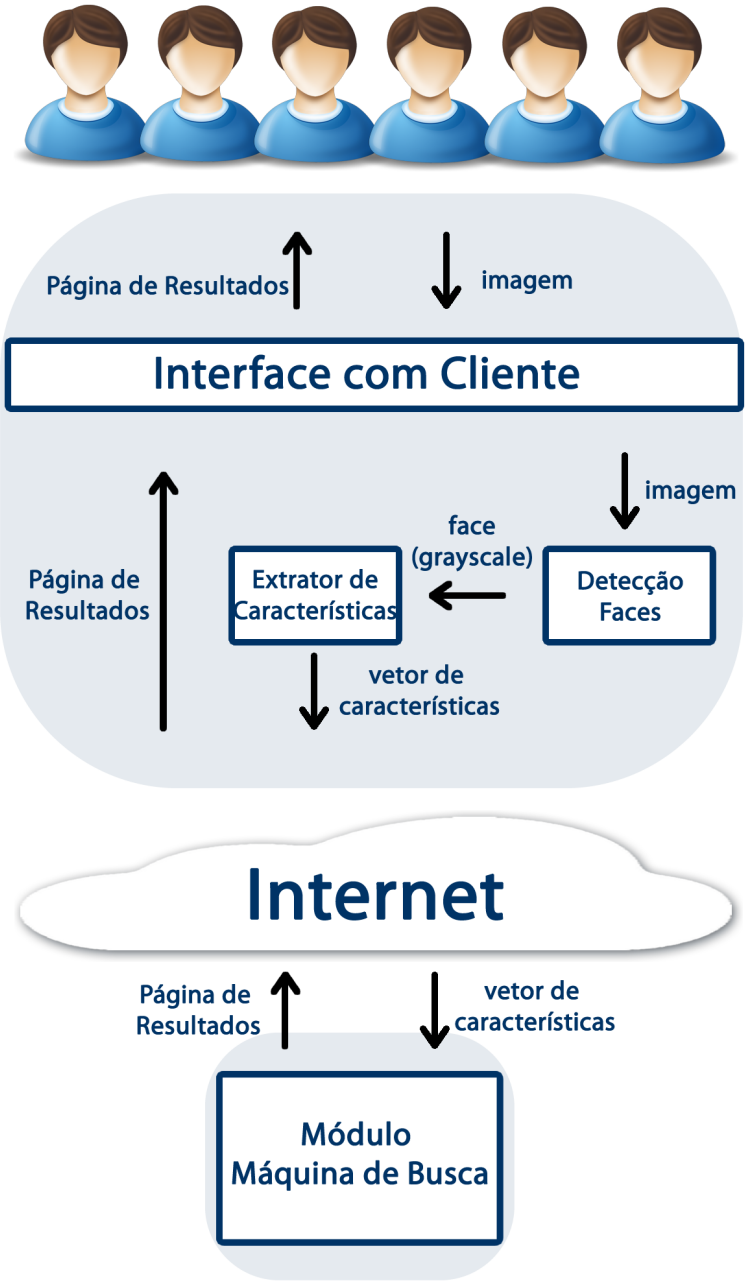

Figura 8: Módulo 3 - Interface com o Cliente

4 Experimentos

Para a realização dos experimentos foi necessário definir as sementes iniciais para o Aquarela *Web Crawler* iniciar suas atividades e definir com cautela as políticas de convivência a serem adotadas. A definição das sementes iniciais seguiram dois pontos de entrave: é interessante executar o *crawler* de forma natural, buscando imagens aleatórias, porém também é interessante buscar em um banco de dados confiável afim de testar com mais facilidade as taxas de acerto na busca de faces afim de propor estratégias de refinamento da ferramenta. Como foi necessário realizar diversos testes antes de iniciar a execução do mesmo de forma satisfatória, foi definida a utilização de *URLs* de sites de universidades públicas brasileiras, visto que o Facebook.com não respondeu a solicitação de permissão de utilizar o *crawler* em suas páginas.

As políticas de convivência são importantes para que o *crawler* não gere qualquer ônus as páginas e servidores visitados. Por ser um *crawler* extremamente robusto, e o experimento estar sendo realizado dentro do campus principal da Universidade Federal de Itajubá, com disponibilidade de todos os seus recursos, principalmente de banda de *Internet*, foi interessante utilizar padrões extremamente conservadores para as políticas de convivência, sendo definido um máximo de duas conexões simultâneas, e um intervalo de um segundo entre as requisições para cada servidor.

Como definido pela política de convivência, há possibilidade de no máximo duas conexões simultâneas por *host*, e um intervalo de um segundo para as requisições. Desta forma foi limitado também o número de threads a serem iniciadas. Foi utilizado o dobro de threads referentes ao número inicial de sementes. Seguindo também com a preocupação em evitar ônus a servidores de terceiros, foi pertinente adotar somente as visitas dos sites de universidades públicas como padrão para o experimento, desta forma também foi limitado a não seguir *links* externos ao domínio.

Na execução do experimento, foi definido o estudo diante a análise de 12.000 imagens. Desta forma, as opções relativas a profundidade de visitas de *URL* em um domínio não

foram limitadas. O tempo máximo de espera relativo ao *socket* e conexão foram configurados para os valores padrões. Desta forma, o tempo de espera para *socket* adotado é de 20s e de conexão de 30s.

Nos diversos experimentos de refinamento da ferramenta, verificamos que usualmente as imagens retornadas raramente possuiam valores superiores a 1MB, porém com o avanço das tecnologias de captura de imagem e visando não restringir o universo amostral de captura, foi definido que imagens de até 15MB poderiam ser solicitadas.

Com o avanço dos sistemas de busca e da preocupação dos *webmasters* na confiabilidade do domínio e de seus *links*, é comum encontrar redirecionamentos no servidor. Em muitos casos, demais sites podem inserir *links* incorretos para os domínios, ou muitas vezes o conteúdo pode ter sido removido ou migrado para outra sessão ou categoria do site. Desta forma, na realização do experimento, foi considerado viável habilitar a funcionalidade de seguir redirecionamentos.

É interessante também analisar o desempenho do *crawler* diante as páginas encriptadas com *SSL*. Desta forma, na realização do experimento, foi definido como viável seguir tais páginas.

Como o volume de imagens adquiridas e faces encontradas eram consideráveis, foi então necessário também o desenvolvimento de um software de análise de resultados, para fins estatísticos, que será apresentado na seção 4.2.1.

4.1 Características do experimento com crawler

Na realização do experimento foi disponibilizado uma workstation que dispunha das seguintes configurações: Processador Intel Core i7 - 930 - 2.8GHz - Primeira Geração, 45nm - Sandy Bridge, com memória principal G.SKill Ripjaws 24GB (6x4GB) DDR3 1600MHz (F3-12800CL9T-12GBRL) em Triple Channel. Estes componentes ligados na placa mãe Asus P6TD Deluxe, utilizando unidade de Armazenamento da Samsung HD502HJ SATA2 com 500GB e com velocidade de 7200 RPM. Com placa gráfica GeForce GT 220.

O sistema operacional utilizado é o Linux, com distribuição Kubuntu. Com kernel na versão 3.0.0-23-generic e o KDE na versão 4.7.4. A instalação do sistema operacional é a padrão. Os softwares adicionais são a IDE Netbeans na versão 7.1, e o software Aquarela *Web Crawler*, desenvolvido neste estudo. A IDE Eclipse foi utilizada inicialmente apenas

para compilar o projeto OpenIMAJ.

Na realização do experimento foram utilizadas as instalações da Universidade Federal de Itajuba, que disponibilizou sua banda de *Internet*, com *link* óptico dedicado de 36Mbps.

4.2 Métodos de Avaliação

Para avaliar o resultado do experimento foi necessário analisar todas as imagens obtidas, não somente as faces resultantes. Desta forma, foi implementada na arquitetura do Aquarela *Web Crawler* um modo de depuração para guardar em disco todas as imagens avaliadas. Desta forma, analisando somente as faces, é possível verificar os falsos positivos, porém, é necessário verificar todas as imagens avaliadas, para determinar os falsos negativos. De maneira geral, os falsos positivos e negativos podem ser identificados quando:

Dado: $NFace_m$ é o número das faces reconhecidas manualmente pela visualização humana e $NFace_a$ é o número de faces reconhecidas pelo de detector de faces do Aquarela *Web Crawler*.

Dado $NFace_m$ e $NFace_a$ de uma imagem de *web* temos que:

$$\begin{cases} se\ \text{NFace}_m - NFace_a > 0, & \text{Falso Positivo.} \\ se\ \text{NFace}_m - NFace_a < 0, & \text{Falso Negativo.} \\ se\ \text{NFace}_m - NFace_a = 0, & \text{Equivalência de reconhecimento.} \end{cases}$$

Na avaliação dos resultados obtidos do experimento, é necessário avaliar as imagens resultantes como sendo faces ou não. Essa análise permite comprovar a exatidão no processo de acerto na detecção de face pelo Aquarela *Web Crawler*. A informação necessária para essa análise e o número de falso positivo.

O falso positivo de uma imagem de faces ocorre quando o Aquarela *Web Crawler* identifica uma face em uma imagem, recorta a mesma, e gera uma nova imagem de face, sendo que esta não é realmente uma face.

Na realização de experimentos de testes, foi verificado também a necessidade de analisar quando o detector de faces não reconhece uma face em uma imagem, mas esta claramente pode ser identificada. Para essa segunda análise foi necessário então criar um módulo de depuração que basicamente consiste em salvar todas as imagens recuperadas pelo Aquarela *Web Crawler* em disco, para posterior análise humana. Neste ponto então,

este processo é necessário para identificar os falsos negativos.

Um falso negativo de uma imagem de face ocorre quando o detector de face não encontra nenhuma face em uma imagem, porém há faces na mesma.

4.2.1 Software de Análise de Resultados

Diante o grande volume de imagens a serem manipuladas e afim de verificar os falsos positivos e falsos negativos fez-se necessário o desenvolvimento de um software específico para auxiliar no processo. Desta forma, foi desenvolvido um *software* em parceria com o PET-TEC Unifei. Sob minha orientação, o aluno Yago Toledo Lima, bolsista do programa PET-TEC Unifei, e cursando graduação em engenharia da computação desenvolveu um programa em *Java*, que atendesse as seguintes especificações:

- Verificar se os nomes da imagens obedecem o formato definido no projeto
- Identificar as imagens que não foram corretamente renomeadas
- Contar o número total de imagens
- Verificar o número total de falsos positivos e negativos
- Prover interface para auxiliar na identificação manual de falsos positivos e negativos para renomear os arquivos

Assim, como podemos observar na Figura 9, a interface inicial do *software* provê as opções de carregar as imagens, ou o diretório de arquivos de imagens para serem analisadas. Após o carregamento dos arquivos de imagens, a interface provê em sua barra lateral esquerda a lista de todas as imagens carregadas, sendo possível selecionar qualquer imagem desejada afim de verificar manualmente as faces visualizadas. Após selecionar a imagem, a mesma será mostrada na barra lateral direita, habilitando então o campo de edição para que o usuário indique as faces encontradas manualmente, e automaticamente o programa irá renomear o arquivo de imagem seguindo a metodologia, alterando o sufixo para: $[NFace_a\text{-}NFace_m]$. É possível ir caminhando com os botões *Anterior* e *Próximo* e verificando assim manualmente as faces em cada imagem. Há também informações pertinentes sobre a imagem, como a dimensão da mesma, o *zoom* aplicado pelo programa, as faces encontradas pelo Aquarela *Web Crawler* para efeitos comparativos.

Figura 9: Software de Análise de Resultados: Verificação de Faces

No software há também a possibilidade de limpar a lista atual de imagens carregadas, caso tenha sido carregada uma lista incorreta, além de opções de ajuda, que informam como utilizar corretamente o programa.

Para efeitos estatísticos, podemos observar na Figura 10 as opções de saída relativas a falso positivo, falso negativo além de outros dados pertinentes do experimento. Para este cálculo, basicamente o software analisa o nome do arquivo da imagem já analisada. Caso encontre o sufixo no formato [$NFace_a$-$NFace_m$], é então executada a rotina de cálculo dos dados estatísticos. Caso haja alguma figura fora do formato, ou mesmo que não tenha sido ainda analisada, o software notifica o usuário, indicando quais imagens ainda não foram analisadas, ou que possuem o nome fora do padrão adotado.

Figura 10: Software de Análise de Resultados: Dados Estatísticos

4.3 Estatísticas com o software de análise

Experimento realizado com um universo amostral de 12.000 imagens coletadas seguindo busca aleatória diante sites de universidades públicas brasileiras. Na composição da Tabela 1, foram utilizados os dados provenientes do software desenvolvido especialmente para tratar dados estatísticos do experimento.

Diretório	Imagens	$NFace_m$	$NFace_a$	Falso Negativo	Falso Positivo
d0000000	2000	4871	4325	2258	1622
d0000001	2000	5477	3342	2866	771
d0000002	2000	2460	4248	423	2211
d0000003	2000	1989	3763	345	2119
d0000004	2000	4111	3006	1766	661
d0000005	2000	2683	3560	412	1289

Tabela 1: Resultado do Experimento

Para a identificação das faces não reconhecidas, conhecidas também como falso negativo foi necessário o trabalho conjunto dos alunos de engenharia da computação do programa PET-TEC Unifei. Para tal verificação, 12 alunos do programa se prontificaram no auxílio a verificação dos falsos negativos, desta forma, para este experimento cada aluno ficou responsável na análise de 1000 imagens. Com o auxílio do software desenvolvido, todo o processo é otimizado, afim de garantir facilidade na identificação manual das faces e renomear os arquivos apropriadamente.

Durante o processo de análise das imagens pelos alunos, o *software* se encarrega em renomear os arquivos alterando o sufixo final do arquivo, inserindo entre colchetes o número de faces reconhecidas pelos alunos. Como entre colchetes já há também o número de faces reconhecidas pelo Aquarela *Web Crawler*, a estrutura segue o padrão: MD5[$NFace_m$-$NFace_a$].png Este padrão é adotado, pois desta forma o software de análises gráficas já utiliza tais dados para calcular também as estatísticas do experimento analisando diretamente os arquivos. Desta forma, facilitando também a aquisição de dados de Falso Positivo.

É possível verificar na Tabela 2 a soma dos dados dos resultados do experimento. Tais dados são importantes afim de identificar o universo amostral de percentual de acerto relativo ao falso positivo e ao falso negativo. Partindo do princípio que as imagens de faces identificadas visualmente pelos alunos estão corretas, estas identificam o universo amostral de todas as faces existentes. Assim é possível verificar estatisticamente a partir destes dados a relação de falsos positivos, e o percentual de eficiência na detecção. Paralelamente,

utilizando os dados de faces encontradas pelo Aquarela *Web Crawler*, é possível verificar os dados estatísticos referentes ao falso negativo, ou seja, o percentual de faces existentes que não foram encontradas.

Diretórios	Imagens	$NFace_m$	$NFace_a$	Total Falso Negativo	Total Falso Positivo
6	12000	21591	22244	8070	8673

Tabela 2: Soma de Resultados do Experimento

Desta forma, o dado estatístico referente ao falso negativo do experimento pode ser observado na Figura 11 e o dado estatístico do falso positivo pode ser verificado na Figura 12.

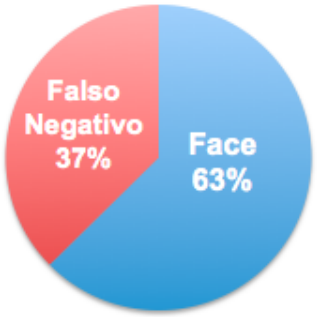

Figura 11: Resultado Experimento: Falso Negativo

Figura 12: Resultado Experimento: Falso Positivo

Diante cada universo amostral de análise, o percentual de falso negativo corresponde a 37% diante as imagens que possuem faces e não foram detectadas, e o falso positivo corresponde a 39% de imagens que foram identificadas com faces, porém as mesmas não existiam.

5 *Conclusão*

Os objetivos definidos no início do projeto contribuiram intrinsecamente na definição das metas primordiais. A arquitetura idealizada e desenvolvida garante perspectiva de continuidade nas contribuições acadêmicas, e na formalização da ferramenta de busca de faces na *web*.

5.1 Principais Contribuições

As principais contribuições deste trabalho são:

- Proposta de uma arquitetura para buscas de faces na *web*. Esta arquitetura é composta por 3 módulos: Módulo de *Crawler*, Módulo de Máquina de Busca, Módulo de Interface com o Cliente.
- Implementação e validação do Módulo de *Crawler*. Este módulo permitiu a aquisição de imagens e extração de faces da *web*.

5.2 Contribuições Secundárias

As contribuições secundárias deste trabalho são:

- Analisando somente os falsos positivos do experimento, conclui-se que: os métodos de detecção de faces são satisfatórios, garantindo mais de 61% de acerto. Sendo possível ainda aplicação de demais técnicas de detecção de olhos afim de otimizar ainda mais os resultados.
- Analisando somente os falsos negativos do experimento, conclui-se que: há possibilidade de refinamento nos processos de detecção criando filtros para imagens de baixa resolução. Porém, ainda assim os métodos são satisfatórios, garantindo um acerto superior a 63%.

- *Software* de análises estatísticas para falsos positivos e negativos.

5.3 Projetos Futuros

Desde o início da concepção e idealização dos estudos referentes ao foco do projeto de dissertação, houve a preocupação nas contribuições acadêmicas e também na possibilidade de desenvolvimento de um serviço. Com a definição da proposta da arquitetura, foi possível traçar metas de desenvolvimento afim de garantir o objetivo final: o projeto de *crawler* e máquina de busca de faces. Este projeto teve por objetivo explorar a implementação do módulo de *crawler* de faces.

Um projeto que já está em andamento é o módulo de máquina de busca. Este módulo engloba o armazenamento em estrutura de dados métricas dos vetores de características que representam as faces identificadas pelo módulo do *crawler*.

A integração do módulo de máquina de busca com o módulo de *crawler* de faces garante a concepção do serviço de busca. Obviamente que há também todos os desafios perante infraestrutura e escalabilidade do serviço e a disponibilidade aos usuários.

Há diversas aplicações perante o serviço de busca de faces, desde busca de pessoas desaparecidas, busca de fotos com direitos autorais, ou mesmo busca de faces que possuem características em comum.

Além disso é possível citar como propostas de trabalhos futuros:

- Implementar módulo interface com o cliente. Este módulo pode ser implementado com tecnologia que usa processamento no cliente, para extrair o vetor de características que será enviado como parâmetro da consulta paginada.

- Suportar extração de características de outros tipos de imagens diferentes de faces. Generalizar a estrutura do projeto afim de garantir métodos parametrizáveis para vários tipos de imagem, como por exemplo: carro, moto, flores, prédios, animais, etc.

- Adicionar *ranking* de imagens no serviço de busca. Melhorar a página de resultados de pesquisa de imagens para o usuário, adicionando o *ranking* para intensificar o posicionamento das imagens na primeira página.

Referências

[Abiteboul, Preda e Cobena 2003]ABITEBOUL, S.; PREDA, M.; COBENA, G. Adaptive on-line page importance computation. In: *Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM, 2003. (WWW '03), p. 280–290.

[Baeza-yates e Castillo 2002]BAEZA-YATES, R.; CASTILLO, C. Balancing volume, quality and freshness in web crawling. In: *In Soft Computing Systems - Design, Management and Applications*. [S.l.]: IOS Press, 2002. p. 565–572.

[Baeza-Yates et al. 2005]BAEZA-YATES, R. et al. Crawling a country: better strategies than breadth-first for web page ordering. In: *Special interest tracks and posters of the 14th international conference on World Wide Web*. New York, NY, USA: ACM, 2005. (WWW '05), p. 864–872.

[Belhumeur, Hespanha e Kriegman 1997]BELHUMEUR, P. N.; HESPANHA, J.; KRIEGMAN, D. J. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 19, n. 7, p. 711–720, 1997.

[Brin e Page 1998]BRIN, S.; PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 30, n. 1-7, p. 107–117, abr. 1998.

[Chakrabarti, Berg e Dom 1999]CHAKRABARTI, S.; BERG, M. van den; DOM, B. Focused crawling: a new approach to topic-specific web resource discovery. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 31, n. 11-16, p. 1623–1640, maio 1999. ISSN 1389-1286.

[Cho e Garcia-Molina 2000]CHO, J.; GARCIA-MOLINA, H. Synchronizing a database to improve freshness. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 29, n. 2, p. 117–128, maio 2000.

[Cho e Garcia-Molina 2003]CHO, J.; GARCIA-MOLINA, H. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.*, ACM, New York, NY, USA, v. 28, n. 4, p. 390–426, dez. 2003.

[Cho, Garcia-Molina e Page 1998]CHO, J.; GARCIA-MOLINA, H.; PAGE, L. Efficient crawling through url ordering. *Comput. Netw. ISDN Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 30, n. 1-7, p. 161–172, abr. 1998.

[Coffman, Liu e Weber 1997]COFFMAN, E.; LIU, Z.; WEBER, R. R. *Optimal Robot Scheduling for Web Search Engines*. 1997.

[Cothey 2004]COTHEY, V. Web-crawling reliability. *J. Am. Soc. Inf. Sci. Technol.*, John Wiley & Sons, Inc., New York, NY, USA, v. 55, n. 14, p. 1228–1238, dez. 2004.

[Cox, Ghosn e Yianilos 1995]COX, I. J.; GHOSN, J.; YIANILOS, P. N. *Feature-based recognition using misture-distance*. [S.l.], 1995.

[Daubechies 1992]DAUBECHIES, I. *Ten Lectures on Wavelets (C B M S - N S F Regional Conference Series in Applied Mathematics)*. [S.l.]: Soc for Industrial & Applied Math, 1992. Paperback.

[Diligenti et al. 2000]DILIGENTI, M. et al. Focused crawling using context graphs. In: *Proceedings of the 26th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (VLDB '00), p. 527–534.

[Dill et al. 2002]DILL, S. et al. Self-similarity in the web. *ACM Trans. Internet Technol.*, ACM, New York, NY, USA, v. 2, n. 3, p. 205–223, ago. 2002. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/572326.572328>.

[Edwards, McCurley e Tomlin 2001]EDWARDS, J.; MCCURLEY, K.; TOMLIN, J. An adaptive model for optimizing performance of an incremental web crawler. In: *Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, 2001. (WWW '01), p. 106–113.

[Gong, McKenna e Psarrou 2000]GONG, S.; MCKENNA, S.; PSARROU, A. *Dynamic Vision: From Images to Face Recognition*. [S.l.]: Imperial College Press UK, 2000.

[Gonzalez e Woods 2006]GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[Haar 1910]HAAR, A. Zur theorie der orthogonalen funktionensysteme. *Annals of Mathematics*, v. 69, p. 331–371, 1910.

[Jing e Baluja 2008]JING, Y.; BALUJA, S. Pagerank for product image search. In: *Proceedings of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008. (WWW '08), p. 307–316.

[Kleinberg 1999]KLEINBERG, J. M. Authoritative sources in a hyperlinked environment. *J. ACM*, ACM, New York, NY, USA, v. 46, n. 5, p. 604–632, set. 1999.

[Kobayashi e Takeda 2000]KOBAYASHI, M.; TAKEDA, K. Information retrieval on the web. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 32, n. 2, p. 144–173, jun. 2000.

[Koster 1996]KOSTER, M. *A standard for robot exclusion*. 1996. Http://www.robotstxt.org/wc/robots.html. Disponível em: <http://www.robotstxt.org/wc/exclusion.html>.

[Lauwrence et al. 1996]LAUWRENCE, S. et al. *Face recognition: a hybrid neural network approach*. [S.l.], 1996.

[Lawrence e Giles 2000]LAWRENCE, S.; GILES, C. L. Accessibility of information on the web. *Intelligence*, ACM, New York, NY, USA, v. 11, n. 1, p. 32–39, abr. 2000.

[Mallat 1989]MALLAT, S. G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 11, p. 674–693, 1989.

[Marchiori 1997]MARCHIORI, M. The quest for correct information on the web: hyper search engines. *Comput. Netw. ISDN Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 8-13, p. 1225–1235, set. 1997.

[McBryan 1994]MCBRYAN, O. A. Genvl and wwww: Tools for taming the web. In: *In Proceedings of the First International World Wide Web Conference*. [S.l.: s.n.], 1994. p. 79–90.

[Menczer 1997]MENCZER, F. *ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery*. 1997.

[Menczer e Belew 1998]MENCZER, F.; BELEW, R. K. Adaptive information agents in distributed textual environments. In: *Proceedings of the second international conference on Autonomous agents*. New York, NY, USA: ACM, 1998. (AGENTS '98), p. 157–164.

[Najork e Wiener 2001]NAJORK, M.; WIENER, J. L. Breadth-first crawling yields high-quality pages. In: *Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, 2001. (WWW '01), p. 114–118.

[Nelson et al. 2005]NELSON, M. L. et al. mod_oai: an apache module for metadata harvesting. In: *Proceedings of the 9th European conference on Research and Advanced Technology for Digital Libraries*. Berlin, Heidelberg: Springer-Verlag, 2005. (ECDL'05), p. 509–510.

[Page et al. 1998]PAGE, L. et al. *The pagerank citation ranking: Bringing order to the web*. [S.l.]: Technical report, Stanford Digital Library Technologies Project, 1998, 1998.

[Page et al. 1999]PAGE, L. et al. *The PageRank Citation Ranking: Bringing Order to the Web*. 1999.

[Pinkerton 2000]PINKERTON, B. *WebCrawler: Finding What People Want*. 2000.

[Shkapenyuk e Suel 2002]SHKAPENYUK, V.; SUEL, T. Design and implementation of a high-performance distributed web crawler. In: *In Proc. of the Int. Conf. on Data Engineering*. [S.l.: s.n.], 2002. p. 357–368.

[Shnaider e Papliriski 1996]SHNAIDER, I.; PAPLIRISKI, A. P. Compression of fingerprint images using wavelet transform and vector quantization. In: *International Symposium on Signal Processing and its Applications (ISSPA)*. Australia, Gold Coast: [s.n.], 1996. p. 436–440.

[Spertus 1997]SPERTUS, E. Parasite: mining structural information on the web. *Comput. Netw. ISDN Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 8-13, p. 1205–1215, set. 1997.

[Stollnitz, Derose e Salesin 1995]STOLLNITZ, E. J.; DEROSE, A. D.; SALESIN, D. H. Wavelets for computer graphics: A primer.1. *IEEE Computer Graphics and Applications*, v. 15, n. 3, p. 76–84, 1995.

[Turk e Pentland 1991]TURK, M. A.; PENTLAND, A. P. Face recognition using eigenfaces. In: *Proceedings of the IEEE Computer Society Conference*. [S.l.: s.n.], 1991.

[Viola e Jones 2001]VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 1, p. 511–I–518 vol.1, 2001.

[Watson 1994]WATSON, A. Image compression using the discrete cosine transform. *Mathematica Journal*, v. 4, p. 81–88, January 1994.

[Weiss et al. 1996]WEISS, R. et al. Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In: *PROCEEDINGS OF THE SEVENTH ACM CONFERENCE ON HYPERTEXT*. [S.l.: s.n.], 1996. p. 180–193.

APÊNDICE A – Instalação OpenIMAJ

Na elaboração do experimento, foi utilizado a biblioteca de aplicações gráficas OpenIMAJ desenvolvida em Java e integrada ao Crawler4J.

Para iniciar o trabalho utilizando a biblioteca OpenIMAJ foi necessário verificar se a estação de trabalho possuia alguns *softwares* necessários:

- Java 1.6.0
- Maven
- Subversion
- Eclipse IDE

A estação de trabalho estava configurada com o sistema operacional Linux com a distribuição Ubuntu, configurada com o KDE (conhecida também como Kubuntu). Para instalar a versão do java recente, o seguinte comando foi utilizado no terminal:

```
sudo apt-get install openjdk-6-jdk
```

Após a instalação, foi possível verificar a versão do Java instalado, juntamente com a versão de seu compilador com os seguintes comandos:

```
java -version
javac -version
```

Após verificar a correta instalação do Java, foi necessário a instalação do *software* conhecido como Maven. O Maven é mantido pela Apache, sendo uma ferramenta desenvolvida com a finalidade de gerenciar projetos e automizar o processo de construção destes. O projeto OpenIMAJ utiliza a ferramenta Maven para sua construção. Em suma, o Maven é responsável em fazer o *download* e instalar todos os arquivos .jar necessários

a todos os projetos e sub-projetos da OpenIMAJ. É da resposabilidade do Maven testar e gerar arquivos de ajustes necessários para permitir a importação dos projetos do OpenIMAJ para o eclipse. Para instalação do Maven na estação de trabalho, o seguinte comando foi utilizado:

```
sudo apt-get install maven2
```

Após instalado, foi possível verificar a versão do mesmo:

```
mvn -version
```

Após a instalação correta do Java e do Maven, faz-se necessário a instalação do *software* conhecido como Subversion. O Subversion é um *software* utilizado para repositório de controle de versões. Desta forma é possível utilizá-lo para fazer download de uma determinada versão de *software* que esteja em um repositório. Para instalar o subversion, basta executar o comando:

```
sudo apt-get install subversion
```

Após instalado, é possível verificar a versão do mesmo:

```
svn --version
```

O ambiente de desenvolvimento conhecido como Eclipse, ou Eclipse IDE foi utilizado para desenvolver inicialmente o projeto. A versão utilizada do eclipse foi a 3.7.2, sendo possível fazer o *download* do mesmo diretamente do site do projeto em eclipse.org/downloads/.

Com todos os softwares necessários instalados, é possível assim iniciar a utilização da biblioteca OpenIMAJ. Primeiramente é necessário fazer o *download* de todos os arquivos necessários, para isso utilizamos o *software* subversion com o comando:

```
svn checkout svn://svn.code.sf.net/p/openimaj/code/trunk openimaj
```

Com a utilização do comando acima, será criado uma pasta chamada openimaj no diretório onde o comando foi utilizado, e dentro deste será salvo todos os arquivos do projeto openimaj. Após finalizar o *download* de todos os arquivos, será necessário compilar, testar e instalar o projeto OpenIMAJ, sendo assim utilizamos o software maven. Dentro do diretório base de download do projeto do OpenIMAJ, é necessário executar o comando:

```
mvn install
```

Após a execução do último comando o projeto do OpenIMAJ está compilado e instalado. Os arquivos .JAR gerados de cada projeto e sub-projeto serão salvos na pasta base de cada

diretório dos respectivos projetos. Nesta etapa em diante, foi interessando a utilização de uma IDE para dar continuidade na programação do experimento, neste caso foi escolhido primeiramente o ECLIPSE IDE. Foi possível então utilizar o comando abaixo para criar de forma apropriada os arquivos de configuração dos projetos próprios para o ECLIPSE IDE:

```
mvn eclipse:eclipse
```

Após todos estes passos de instalação e configuração foi iniciado a integração da biblioteca OpenIMAJ com o Crawler4J.

APÊNDICE B – Sementes Aquarela Web Crawler

www.unifei.edu.br	www.ufmg.br
www.ufop.br	www.usp.br
www.ufscar.br	www.unicamp.br
www.unifesp.br	www.unesp.br
www.ufu.br	www.ufac.br
www.ufv.br	www.ufam.edu.br
www.unirio.br	www.unir.br
www.ufrj.br	www.ufrr.br
www.ufrrj.br	www.uft.edu.br
www.unifal-mg.edu.br	www.ufrb.edu.br
www.fafeod.br	www.ufba.br
www.ufsj.edu.br	www.ufgd.edu.br
www.furg.br	www.unb.br
www.ufrgs.br	www.ufg.br
www.ufsc.br	www.ufmt.br
www.ufpr.br	www.ufms.br
www.ufsm.br	www.ufabc.edu.br
www.utfpr.edu.br	www.ufes.br
www.unipampa.edu.br	www.uff.br
www.ufcspa.edu.br	www.ufjf.br
www.ufla.br	

Tabela 3: Sites das universidades públicas brasileiras utilizados como semente do crawler