MetaCluster.PT

Um Meta-Motor de Pesquisa para a Web Portuguesa

Nuno Miguel Salvado Amador

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Pedro Sousa
Orientador:	Prof. Pável Pereira Calado
Vogais:	Prof. Bruno Martins

Outubro de 2009

Abstract

The exponential increase of available Web pages present great challenges to actual search engines, which have to deal with enormous quantities of information. One of the problems is that a single search engine only indexes a small part of the whole Web. When we search using one of these engines, we can see that some return results that others do not. Thus, combining some of the engines, lets us gain access to more indexed documents. A Meta-Searcher is a search engine that searches various individual search engines, and then combines the results and shows them to the user, avoiding the user to do the individual searches himself. Additionally to the common result list, these can also be grouped and presented by topics with the same semantic meaning, in an automatic manner and without previous knowledge of the data, in a process called clustering. In the particular case of Portuguese Web, we see that a lot of work still has to be done. Commonly, we can only find "directories" with manually pre-classified pages. This Thesis presents a prototype of a Meta-Searcher engine for the Portuguese Web, with result clustering capability. This Prototype is implemented using and combining available open source components. Besides that, necessary modifications and parameterizations are described to make the system capable of dealing with the particularities of the Portuguese language.

Keywords: *Information Retrieval, Meta-Search, Web Clustering, K-Means*

Resumo

O número exponencialmente crescente de páginas disponíveis na Web coloca grandes desafios aos motores de pesquisa, os quais têm de lidar com enormes quantidades de informação. Um dos problemas é que cada motor de pesquisa apenas indexa uma pequena parte de toda a Web. De facto, ao efectuar pesquisas em vários destes motores constatamos que alguns retornam resultados que outros não retornam. Um *Meta-Searcher* consiste num motor que efectua pesquisa sobre vários motores individuais, alargando o acesso a uma maior quantidade de documentos indexados, que depois são combinados e mostrados ao utilizador, evitando que este tenha de fazer várias consultas individuais. Adicionalmente à comum lista de resultados, estes podem ainda ser agrupados e apresentados por tópicos com o mesmo significado semântico, de uma forma automática e sem conhecimento prévio dos dados, num processo denominado *clustering*. No caso particular da Web Portuguesa, constatamos que ainda está muito atrasada nesta matéria. O comum é encontrar "directórios" com as páginas classificadas previamente à mão. Esta Tese apresenta o protótipo de um *Meta-Searcher* para a Web Portuguesa, com a capacidade de efectuar *clustering* de resultados. Este protótipo é implementado recorrendo e combinando componentes disponíveis em código fonte aberto. Além disso, são descritas as modificações e parametrizações necessárias para tomar um tal sistema capaz de lidar com as particularidades da língua portuguesa.

Palavras-Chave: *Recuperação de Informação, Meta-Pesquisa, Clustering Web, K-Means*

Agradecimentos

Gostaria de dedicar o presente trabalho a toda a família, namorada, amigos e colegas.
Agradecimento especial vai para todos aqueles que responderam às questões dos inquéritos,
e para o Prof. Dr. Pável Calado, pela sua orientação, disponibilidade e constante motivação.
Agradeço também ao Prof. Dr. Bruno Martins pela sua ajuda e contribuições.

Obrigado a todos, sem vocês este trabalho não seria possível.

Lisboa, Outubro de 2009

Nuno Amador

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A popularidade da Internet causou um aumento massivo da quantidade de páginas Web disponíveis ao público. Esta explosão de informação coloca novos desafios aos sistemas de recuperação de informação, em particular aos motores de pesquisa, os quais têm de guardar parte dessa informação para posteriormente a recuperar de uma forma eficiente e eficaz. No entanto, os motores de pesquisa actuais apresentam ainda muitos problemas. Quer seja a interface, quer sejam as funcionalidades, muitos ainda estão longe do ideal. Um dos problemas é que não existe nenhum motor que seja capaz, por si só, de indexar toda a Web. Os resultados indexados e retornados por um motor podem não existir noutro, para os mesmos termos da pesquisa. Este facto deu origem a um novo tipo de motores, chamados *Meta-Searchers*. Um *Meta-Searcher* executa pesquisas sobre outros motores de pesquisa, de modo a ter uma lista de resultados mais vasta do que seria possível de obter apenas com esses motores individualmente. Outra possibilidade interessante seria a capacidade de agrupar esses resultados por tópicos, de forma automática. Isto pode ser realizado através de um processo denominado *clustering*.

Este capítulo começa por dar a conhecer ao leitor o contexto e a definição de alguns problemas relativos às pesquisas Web e ao uso de motores *Meta-Search* para a tentativa de resolução desses problemas. Em seguida são descritos os objectivos do trabalho, a metodologia e a abordagem seguida. Por último é referida a organização do documento.

1.1 Contexto e Definição do Problema

A grande motivação para o presente trabalho deve-se ao facto dos motores de busca actuais ainda apresentarem muitos problemas. Um dos problemas tem a ver com a "cobertura" da Web. Com efeito, cada motor de busca actualmente existente apenas consegue indexar uma pequena parte da Web indexável [12]. Em 2008, Eric Schmidt, CEO do actual maior indexador do mundo (`Google`), estimou o tamanho da Web em cerca de 5 milhões de terabytes, ou seja, 5×10^{18} bytes[1]. Segundo ele, em 7 anos de operações, o `Google` indexou cerca de 200 terabytes, o que equivale a 0,004% do tamanho total. Outros estudos publicados na Web, em Dezembro de 2008, referem a existência de cerca de 186,7 milhões de sítios[2] e pelo menos 25

[1] `http://www.wisegeek.com/how-big-is-the-internet.htm`

[2] `http://news.netcraft.com/archives/2008/12/24/december_2008_web_server_survey.html`

biliões de páginas indexáveis[3]. Estudos de 2005 indicavam que os maiores motores de busca existentes, tais como Google, Yahoo!, MSNe Teoma[4] apenas têm em comum cerca 28,8% dos documentos indexados [13]. Num teste aleatório de 91 buscas, o Google e o Yahoo! apenas mostraram em comum 23% dos primeiros 100 resultados. Outro estudo da mesma fonte, desta vez centrado nos utilizadores, mostra que 44% usa regularmente apenas um motor de busca, 48% usa dois ou três motores, e apenas 7% mais de três. O mesmo artigo enfatiza que, se os motores retornam os seus primeiros resultados de uma forma tão diferente, então os pesquisadores que usam esses motores individualmente podem estar a perder resultados importantes. Interessa, por isso, obter os resultados de vários motores de busca de forma automática, através de um sistema que apresenta uma interface de pesquisa única ao utilizador, e que serve de mediador entre essa interface e os vários motores de busca.

Por outro lado, os motores de busca respondem à pesquisa do utilizador com uma lista ordenada de resultados que são considerados relevantes. A maioria destas pesquisas consistem apenas numa ou poucas palavras (ou termos), que muitas vezes são ambíguas demasiado genéricas para especificar com precisão as necessidades de informação do utilizador. Como consequência, apenas uma pequena parte dos resultados são realmente relevantes pois, muitas vezes, apesar de partilharem palavras com a consulta do utilizador, não são semanticamente relacionados com o tema que este procura. Por exemplo, uma procura por "jaguar" normalmente retorna tanto documentos sobre carros como sobre grandes felinos. Uma solução para este problema é organizar os resultados de acordo com os diferentes temas retornados, separando assim os carros dos felinos. Dada a enorme quantidade de informação, e a necessária disponibilidade imediata de classificação de resultados ainda por organizar, é desejável que esta classificação seja realizada de forma automática. Por outras palavras, há necessidade de efectuar *clustering* às páginas dos resultados [41].

O *clustering* foi inicialmente proposto para melhorar a precisão e recolha dos sistemas de recuperação de informação [41]. Com a ajuda dos algoritmos de *clustering*, um utilizador passa a ter uma percepção dos tópicos dominantes num conjunto de documentos, que correspondem ao resultado da sua pesquisa. O utilizador pode então concentrar e refinar a sua pesquisa num tópico do seu interesse, sem a necessidade de percorrer a lista de resultados que não lhe interessam, e ao mesmo tempo aumentando a precisão do sistema. Ainda em 2008, estima-se que a Internet tenha sido usada por pelo menos 1 bilião de pessoas, e destas, cerca de 500 milhões acedem pelo menos uma vez por semana. Com o aumento do número do utilizadores, tal sistema tem igualmente de ter implementados algoritmos eficientes de modo a darem uma resposta rápida às pesquisas, e serem usados de forma interactiva sem aborrecer os seus utilizadores. O MetaCluster desenvolvido no âmbito deste trabalho, é precisamente um sistema que combina um motor *Meta-Search* com funcionalidades de *clustering* de páginas Web.

1.2 Objectivos

Os motores de busca Portugueses são poucos, e os motores do tipo *meta-search* são inexistentes, tanto quanto foi possível apurar. Com base nos pressupostos e nos requisitos já mencionados, apresenta-se

[3] http://worldwidewebsize.com

[4] http://www.google.com, http://www.yahoo.com, http://www.msn.com, http://www.teoma.com

neste trabalho o protótipo de um novo *meta-searcher* de busca para a Web portuguesa. Esta proposta incluiu não só a criação de um *meta-searcher* português com a capacidade de *clustering* de resultados, como também as modificações e parametrizações necessárias para tornar um tal sistema capaz de lidar com as particularidades da nossa língua. Pretende-se que o sistema tenha como fontes motores de busca (indexadores) puramente Portugueses, e também outros motores de busca que sejam capazes de retornar resultados em Português (podendo igualmente incluir o Português do Brasil).

Finalmente, e já numa fase de experimentação e avaliação do protótipo, pretende-se também executar testes com a ajuda de utilizadores independentes. Em resumo, os objectivos do protótipo são: (1) ter uma Interface intuitiva para o utilizador; (2) processar um ou mais termos de pesquisa, combinando-os de alguma forma; (3) consultar mais do que um motor de busca convencional, que retorne resultados em Português; (4) ordenar e agrupar os resultados das buscas obtidos dos motores individuais; (5) organizar os resultados por tópicos semânticos; e (6) minimizar o tempo de resposta ao utilizador.

1.3 Metodologia e Abordagem Proposta

Dada a alargada quantidade de código fonte aberto disponível na Internet, não há propriamente a necessidade de fazer um sistema totalmente novo de raiz, bastando apenas obter os módulos e os componentes necessários, já existentes, e finalmente adaptando-os e combinando-os de modo a construir o novo sistema. Em seguida, e já numa fase de avaliação e experimentação, a interface do protótipo será alterada, e serão elaboradas páginas dinâmicas que serão construídas com resultados reais, retornados e agrupados pelo sistema. As páginas serão apresentadas a vários utilizadores independentes, sob a forma de inquéritos. Assim, os utilizadores estarão a avaliar resultados relativos à sua própria pesquisa, e não resultados previamente “fabricados” e impostos, que nem sequer poderão ser do seu interesse, minimizando assim respostas aleatórias. Os resultados das submissões de inquéritos serão guardados em ficheiros no servidor, para posteriormente serem tratados e analisados, de forma a avaliar a capacidade do sistema para retornar resultados com alta precisão e a qualidade informativa dos grupos (*clusters*) formados.

Como metodologia de desenvolvimento, foi seguido o modelo da cascata invertida [35], ilustrado na Figura 1.1. Esta metodologia clássica e simples foi escolhida porque os objectivos e requisitos à partida encontram-se bem definidos e, como o objectivo consistiu em usar código fonte aberto, para alguns requisitos algumas das etapas podiam ser facilmente completadas ou mesmo eliminadas. Durante a fase de *Análise*, foi efectuado todo o trabalho de pesquisa e investigação relativa aos temas de recuperação de informação, motores de pesquisa e meta-pesquisa, fusão e ordenação de resultados, aprendizagem não-supervisionada, algoritmos de *clustering*, prospecção de texto na Web, indexação de texto e algoritmos de radicalização. Na fase de *Desenho*, foi estudada a forma de combinar os vários componentes de *software*, as bibliotecas a serem usadas e as novas classes/objectos a desenvolver. A fase de *Implementação* consistiu no desenvolvimento em Java das funcionalidades pretendidas, por exemplo, as classes para obter resultados das novas fontes, os algoritmos de *clustering* e de radicalização, e as classes necessárias para a geração dinâmica dos formulários dos inquéritos da avaliação experimental. Durante a fase de *Testes*, foram efectuados testes unitários sobre o código desenvolvido na fase anterior, e tendo em consideração as

funcionalidades presentes nos requisitos. A última fase da metodologia corresponde à *Aceitação* do protótipo desenvolvido.

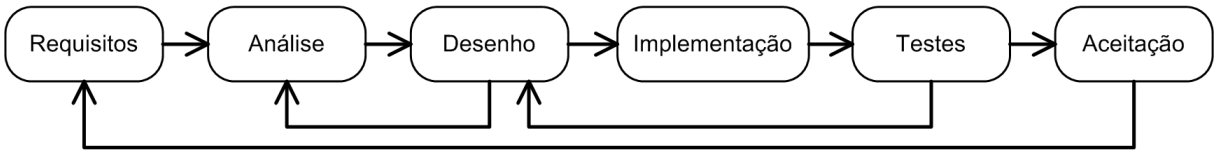


Fig. 1.1: Metodologia de Desenvolvimento adoptada.

1.4 Organização da Dissertação

O documento está organizado do seguinte modo: o Capítulo 2 efectua uma abordagem teórica sobre os motores de busca convencionais e *Meta-Search*, e também sobre o que é o *clustering*, neste caso aplicado a páginas Web; no Capítulo 3 descrevem-se alguns algoritmos usados em *clustering* de documentos Web e os sistemas onde esses algoritmos foram implementados; no Capítulo 4 é apresentada a arquitectura do protótipo implementado; no Capítulo 5 apresentam-se os resultados experimentais obtidos através de inquéritos preenchidos por vários utilizadores que usaram o sistema; finalmente, no Capítulo 6 são apresentadas as conclusões e sugestões de trabalho futuro.

Capítulo 2

Enquadramento Teórico

Este capítulo começa por dar a conhecer ao leitor alguns conceitos teóricos relacionados com os temas em estudo. Primeiro, é descrita a arquitectura geral de motores de busca convencionais e de motores *Meta-Search*, nomeadamente os módulos que os compõem. Em seguida, são descritas algumas das etapas que constituem a primeira fase (ou pré-processamento) nos sistemas que efectuam prospecção de texto. Finalmente, são apresentadas algumas definições teóricas de algoritmos clássicos de *clustering*.

2.1 Arquitecturas dos Motores de Busca

Os motores de busca (também chamados *indexing services*) são basicamente Sistemas de Recuperação de Informação [7]. Estes sistemas permitem aos utilizadores procurarem informação na Web, na qual entretanto são cada vez disponibilizados mais documentos. De acordo com [46], os motores de busca na Web podem ser classificados como se mostra na Figura 2.1.

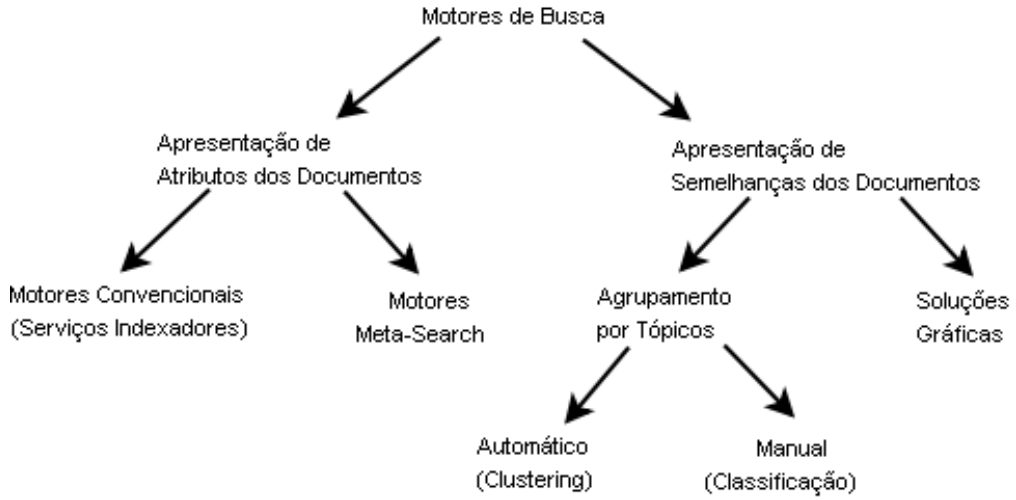


Fig. 2.1: Classificação dos Motores de Busca.

Como ilustrado na figura, os motores de busca dividem-se em dois grandes grupos. O primeiro grupo engloba os sistemas que fazem uma apresentação dos documentos por atributos, numa lista ordenada. Estes

atributos podem ser relacionados com o documento (tais como tamanho, data, fonte ou popularidade), ou então relacionados com o utilizador (tópicos ou termos de busca pré-definidos) e condicionam a posição do documento na lista [48]. O segundo grupo engloba os sistemas que usam esses atributos de modo a encontrar semelhanças entre os documentos, de modo a apresentá-los de forma agrupada ou de forma gráfica. O agrupamento dos documentos pode ser realizado de forma automática (*clustering*), através de um algoritmo, ou então manualmente (classificação) através de intervenção humana. Tendo em conta a mesma figura, o `MetaCluster` referido neste trabalho consiste, portanto, num motor *Meta-Search* com funcionalidades de *clustering*.

As secções seguintes descrevem as arquitecturas de um motor convencional (*Indexing Service*), o qual fornece apenas um serviço de indexação e busca das páginas, e de um motor *Meta-Search*, que combina resultados de vários motores convencionais. Na secção 3.3.6 serão mencionados alguns motores de busca nos quais os resultados são apresentados de forma gráfica.

2.1.1 Motores de Busca Convencionais

A arquitectura geral de um motor de busca convencional é apresentada na Figura 2.2.

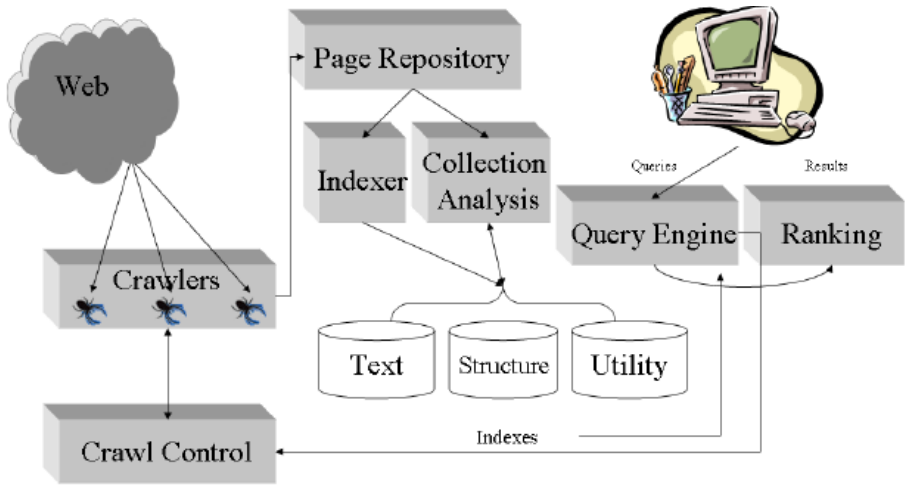


Fig. 2.2: Arquitectura típica de um Motor de Busca convencional [12].

Os principais módulos são [12]:

1. *Web Crawler* - Também conhecido por *Web Spider* ou *Web Robot*, consiste num módulo de software que visita os sítios da Web, segundo determinadas regras (note-se a ligação ao módulo *Crawl Control*), e recolhendo as páginas desses sítios e armazenando-as no repositório (*Page Repository*).

2. *Crawl Control* - Módulo que condiciona as acções realizadas pelo módulo *Web Crawler*. Recebe os termos de busca do *Query Engine* e algum feedback dos documentos já indexados pelo *Indexer* e analisados pelo *Collection Analyzer* e transmite esta informação ao *Web Crawler* de modo a que este visite novos sítios Web.

3. *Repository* - Base de dados que guarda as páginas antes de serem indexadas e classificadas.

4. *Indexer* e *Collection Analyzer* - Periodicamente, o indexador e o classificador indexam e classificam as páginas do repositório, e armazenam o resultado em bases de dados separadas, para uma rápida consulta posterior. No caso particular do indexador, é usado um tipo especial de ficheiro denominado *Inverted Index File* [45], que consiste num ficheiro de termos/palavras em que, para cada termo, é guardada a lista de documentos em que este ocorre. Isto significa que, para cada termo de pesquisa, é possível localizar imediatamente no repositório todos os documentos que contêm o termo [41].

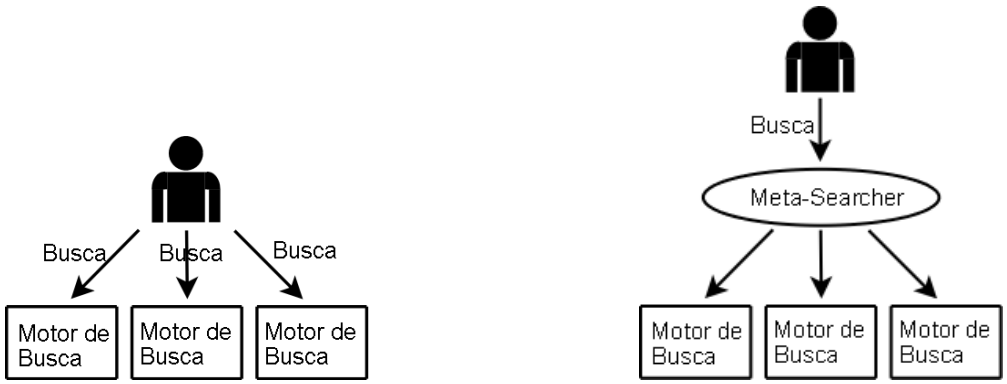
5. *Query Engine* - Corresponde ao módulo que combina os vários termos de pesquisa introduzidos pelo utilizador e os traduz para um formato interno de modo a serem usados pelos restantes módulos.

6. *Ranking* - Módulo responsável por ordenar, segundo determinados critérios, as páginas já indexadas e classificadas, em resposta a uma pesquisa de um utilizador efectuada através do *Query Engine*.

Note-se que as bases de dados destes motores de busca são limitadas, apenas contêm informação relativa a uma amostra de toda a Web. Daí a ideia de combinar vários destes motores para dar origem a um novo motor *Meta-Searcher*.

2.1.2 Motores de Busca *Meta-Search*

Os motores de busca *Meta-Search* são sistemas que podem automaticamente e simultaneamente efectuar pesquisas em vários motores de busca convencionais, e em seguida interpretar os resultados e mostrá-los num formato uniforme. Essencialmente, um motor de busca *Meta-Searcher* consiste num motor que se interpõe entre vários motores convencionais e o utilizador, como se pode ver na Figura 2.3.



(a) Convencional (b) *Meta-Search*

Fig. 2.3: Pesquisa Convencional (a) *vs Meta-Search* (b).

Ao contrário dos motores de busca convencionais, os motores *meta-search* não acedem directamente aos conteúdos dos documentos Web, mas sim ao conjunto das pequenas descrições retornadas pelos motores convencionais.

A Figura 2.4 mostra a arquitectura típica de um motor *Meta-Search*. A numeração das setas indica a sequência das acções. Os componentes desta arquitectura são [23]:

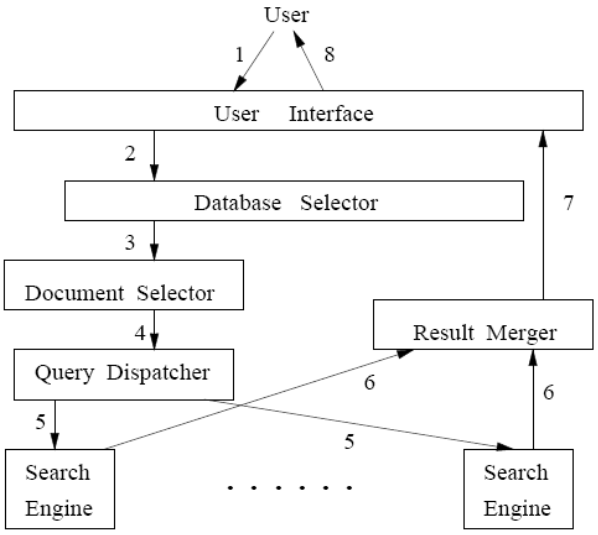


Fig. 2.4: Arquitectura típica de um Motor de Busca *Meta-Search* [23].

User Interface. Módulo que é responsável pela interacção com o utilizador. Converte os termos introduzidos pelo utilizador num formato interno e apresenta os resultados da busca.

Database Selector. Se o número de motores de busca fonte individuais a consultar por parte do *Meta-Searcher* é pequeno, bastará enviar a *query* do utilizador para cada um deles. No entanto, se o número de motores fonte é elevado, esta estratégia já não é aceitável. Isto porque alguns desses motores não irão trazer uma mais valia para os resultados, fazendo com que a percentagem de documentos relevantes seja menor. Além disso, existem outros problemas. Primeiro, o envio de *queries* para as fontes consome recursos do lado do *Meta-Searcher*. Segundo, o envio de dados para as fontes e a recepção dos resultados das mesmas consome largura de banda na rede e tempo de transmissão. Terceiro, também são desperdiçados recursos nos sistemas fonte. Quarto, quantos mais resultados forem retornados, mais tempo precisará o *Meta-Searcher* para processar os resultados, e identificar documentos relevantes. Deste modo, o principal objectivo deste módulo é identificar as fontes que poderão retornar os melhores resultados em resposta à *query* introduzida pelo utilizador, e eliminar as fontes redundantes.

Document Selector. Este módulo tem objectivos semelhantes aos do módulo *Database Selector*. Neste caso, o módulo tenta eliminar documentos que, à partida, serão irrelevantes. Assim, para cada motor fonte são identificados os documentos que, potencialmente, deverão ser retornados. Por exemplo, ao consultar um motor de busca podemos querer apenas os documentos escritos em português.

Query Dispatcher. O objectivo principal deste módulo é traduzir a *query* do utilizador no formato de *query* específico de cada motor fonte. O módulo também poderá alterar a *query* de maneira a ter em conta o peso relativo de cada termo que a constitui. Por último, o módulo saberá lidar com o protocolo e com a interface de cada motor fonte, enviando cada pedido separadamente e, eventualmente, em paralelo, de forma a minimizar o tempo de espera do utilizador.

Search Engine. Motor de busca convencional, o qual possui uma ou várias bases de dados com documentos indexados.

Result Merger. Após os resultados provenientes das fontes escolhidas serem retornados ao *Meta-Searcher*, este módulo é responsável por combinar esses resultados numa lista. Em seguida poderá eliminar resultados duplicados ou bastante semelhantes, e colocá-los ordenadamente numa lista, tendo em conta a sua semelhança com a *query* do utilizador. No final, os primeiros resultados do topo da lista são apresentados ao utilizador.

As vantagens dos motores *Meta-Search* são [1, 11, 13, 23]:

1. O utilizador só precisa de interagir com um sistema, e com uma interface;
2. A possibilidade de combinar os resultados dos vários motores convencionais, pois um documento poderá só estar indexado num deles, aumentando assim a "cobertura" da Web e dos resultados e, consequentemente, aumentando também a eficácia das pesquisas;
3. Resolvem o problema da escalabilidade das pesquisas na Web, pois trata-se de uma tarefa impossível para um motor apenas;
4. Efectuar uma fusão (*merging*) dos resultados que se referem ao mesmo documento, quando estes são retornados por vários motores individuais, evitando assim resultados duplicados;
5. Efectuar uma ordenação (*ranking*) dos resultados obtidos dos vários motores antes de os mostrar ao utilizador;
6. Escolher dinamicamente os motores individuais mais adequados a usar face a uma determinada pesquisa.

No entanto estes motores também apresentam as desvantagens:

1. Perdem-se algumas das funcionalidades mais avançadas oferecidas pelos motores individuais. Por exemplo, o `Google` oferece um serviço de tradução dos resultados para outras línguas;
2. Usam apenas os primeiros resultados retornados pelos motores individuais;
3. Demoram mais tempo a apresentar os resultados ao utilizador pois, além de terem de obter os resultados dos motores individuais, têm ainda de processar esses resultados de forma a eliminar eventuais duplicados, ordená-los e por vezes ainda agrupá-los antes de os apresentar.

Resumindo, ao combinarmos os resultados obtidos por vários motores, estamos a aumentar a hipótese de encontrarmos algo relevante para a nossa pesquisa. Após obter os resultados, o *Meta-Searcher* pode ponderar qual o peso de cada motor e de cada resultado obtido, antes de processar e reordenar a lista que será apresentada ao utilizador. Os motores *Meta-Search* podem aumentar efectivamente a cobertura da Web, mas no entanto, estão limitados pelos motores individuais no que diz respeito ao número e à qualidade dos resultados. Importa notar que as *queries* enviadas aos motores individuais são as que melhor reflectem as necessidades de informação dos utilizadores.

Seria também de grande utilidade, a possibilidade de apresentar a lista de resultados agrupada por tópicos, de modo a facilitar a tarefa de encontrar os resultados pretendidos, sem obrigar o utilizador a percorrer toda a lista. Além disso, com a apresentação dos tópicos ao utilizador, este ficaria também a conhecer os diversos temas relacionados com a sua pesquisa, e por que caminhos este poderia prosseguir. Para este propósito são usados os algoritmos de *clustering*.

2.2 Filtragem e Pré-Processamento

Na Web estão disponibilizados vários tipos de documentos. É necessária a remoção da estrutura dos vários documentos na Web (HTML, XML, PDF, ...), de modo a ficar só com o conteúdo correspondente ao texto. No presente trabalho, este tratamento já é realizado nos motores de busca individuais, os quais disponibilizam o texto dos títulos e dos *snippets*[1] dos documentos, que depois são usados pelo sistema `MetaCluster`. No entanto, após a meta-pesquisa, e antes dos documentos Web serem processados, é ainda necessário eliminar algum ruído que estes contenham, e efectuar alguns passos auxiliares de pré-processamento. Estes passos são importantes, pois influenciam bastante o resultado final da pesquisa e da construção dos tópicos nas fases seguintes de processamento. Em seguida descrevem-se alguns dos passos mais importantes da fase de pré-processamento de texto, a qual corresponde à primeira fase de um sistema que efectua prospecção de texto (*Text Data Mining*).

2.2.1 Palavras Frequentes e Sinónimos

Antes das etapas de radicalização e de tokenização, cada termo obtido passa pela etapa de limpeza. Nesta etapa, primeiro são removidas as palavras frequentes (*stopwords*) [2], e depois é verificada a existência de sinónimos (*thesaurus*) da mesma palavra no dicionário, antes da fase de radicalização. A lista de palavras frequentes, é uma lista de termos não representativos para um documento, ou seja, palavras que só por si não acrescentam informação, e podem por isso ser descartadas. Geralmente esta lista é composta por preposições, artigos, advérbios, números, pronomes e pontuação. No entanto, os documentos na Web são escritos em várias línguas. Por este motivo, é necessário saber previamente qual a língua que se está a tratar, de forma a aplicar a lista correcta para a remoção de palavras frequentes, bem como o correcto dicionário de sinónimos. Ainda outros problemas relacionados com a linguagem são, por exemplo: (1) a gramática usada na constituição das frases; (2) a sua composição sintáctica; (3) seu significado, ou a semântica; (4) existem linguagens em que se escreve da esquerda para a direita, outras da direita para a esquerda, e ainda outras em que se escreve de cima para baixo.

2.2.2 Radicalização de Palavras

A etapa de radicalização de palavras, também conhecida por *stemming* [2], consiste na redução de um termo ao seu radical, removendo as desinências, afixos, e vogais temáticas. As regras de redução têm de ser cuidadosamente definidas, de forma a evitar os erros de *understemming* e *overstemming*. Ocorre

[1] Um *snippet* corresponde a um pequeno excerto de texto que geralmente descreve o conteúdo de um documento numa busca.

erro de *understemming* quando duas palavras pertencentes ao mesmo grupo conceptual são convertidas para raízes diferentes, ao invés da mesma raiz. Por outro lado, o erro de *overstemming* ocorre quando duas palavras pertencentes a grupos conceptuais diferentes são convertidas para a mesma raiz, ao invés de raízes diferentes. Na prática, a radicalização poderia ser obtida com uma lista relacionando a palavra com a sua raiz. No entanto, para certas linguagens, como a Húngara, cada palavra pode ter várias formas, o que originaria uma lista de biliões de combinações. O algoritmo, como é usado em tempo real, tem de ser também eficiente. Com a sua utilização, os termos derivados de um mesmo radical serão contabilizados como um único termo que, na fase seguinte de tokenização, corresponderá a um único identificador numérico. Com a compactação do número de termos, a radicalização tem a vantagem de reduzir o tamanho dos ficheiros de índice invertido (podendo mesmo chegar aos 50%), e aumentando posteriormente a eficiência da pesquisa. Tal como a etapa de remoção de palavras frequentes, esta etapa necessita de saber qual é a linguagem do documento que se está a processar, de modo a aplicar o algoritmo de radicalização correcto.

2.2.3 Tokenização

É nesta etapa que os documentos são transformados para uma forma numérica. Cada documento da colecção vai ter o seu conteúdo dividido em termos, ou seja, cada palavra significante presente no documento. A tokenização é assim utilizada para decompor o documento em cada termo que o compõe, atribuindo a cada palavra (já reduzida ao seu radical) um identificador numérico único. Desta forma, as fases seguintes do processamento tornam-se mais eficientes, pois apenas têm de lidar com números e não com cadeias de caracteres. Os delimitadores utilizados para a tokenização são geralmente o espaço em branco entre os termos, quebras de linhas, tabulações, e alguns caracteres especiais.

2.2.4 Frequência e contagem dos Termos

Nesta etapa, o conteúdo de cada documento é decomposto em termos e na sua frequência no documento e no conjunto de documentos. Primeiro, os termos menos significativos e identificados como palavras frequentes são descartados. Em seguida, depois de se extraírem os termos representativos de cada documento, será calculado o número de ocorrências de cada termo nesse documento e no conjunto de documentos.

A medida usada para calcular a frequência dos termos é a função *TF-IDF*. Esta função define a importância do termo dentro da colecção de documentos, atribuindo-lhe um peso. Consiste no produto de dois factores, "frequência de termos" (*TF - Term Frequency*) e "frequência inversa de documentos" (*IDF - Inverse Document Frequency*) [2]. A função TF corresponde ao número de ocorrências de um termo num documento, dividido pelo número total de termos na colecção. Uma palavra frequente no texto recebe assim uma maior relevância do que uma palavra menos frequente. A função IDF tenta modelar o poder discriminatório da palavra no âmbito do conjunto. Quanto menos documentos conterem a palavra, maior relevância lhe é atribuída. Existem muitas variantes da medida *TF-IDF*, sendo a mais comum a de Salton [38], definida por

$$tfidf_{t,d} = w_{t,d} = tf_{t,d} * idf_t = \frac{freq_{t,d}}{|D_d|} * log\left(\frac{N}{n_t}\right) \tag{2.1}$$

onde:

$w_{t,d}$ - peso do termo T_t no documento D_d;
$tf_{t,d}$ - frequência do termo T_t no documento D_d;
idf_t - frequência inversa de documentos com T_t;
$freq_{t,d}$ - número de ocorrências de T_t em D_d;
$|D_d|$ - número total de termos em D_d;
N - número total de documentos (colecção);
n_t - número de documentos com T_t.

2.2.5 Representação do Conjunto de Documentos

O Modelo Espaço de Vectores, é a representação amplamente usada na área de recuperação de informação. Neste modelo, os documentos são vistos como vectores lineares multi-dimensionais, assim como as palavras distintas presentes nesses documentos. As palavras em cada documento são tratadas, portanto, como atributos desse documento. Com estes vectores obtém-se assim a matriz A, descrita na equação 2.2, a qual também é conhecida como matriz Termos-Documentos, e que constitui o resultado da fase de pré-processamento.

$$A_{|m\times n|} = \begin{bmatrix} w_{1,1} & \dots & w_{1,j} & \dots & w_{1,n} \\ \vdots & \ddots & & & \vdots \\ w_{i,1} & & w_{i,j} & & w_{i,n} \\ \vdots & & & \ddots & \vdots \\ w_{m,1} & \dots & w_{m,j} & \dots & w_{m,n} \end{bmatrix}, \tag{2.2}$$

Assim, nesta matriz, cada linha i representa um vector

$$t_i = (w_{i,1}, w_{i,2}, \dots, w_{i,n}) \tag{2.3}$$

de termos (ou palavras) e cada coluna j representa um vector

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{m,j})^T \tag{2.4}$$

de documentos, em que m é o número total de termos distintos da colecção e n o total de documentos. O valor $w_{i,j}$ de cada posição i, j da matriz 2.2 corresponde ao valor da frequência *TF-IDF* do termo t_i no documento d_j, ou seja, a representação numérica da frequência de cada termo relativamente a cada documento. A grande vantagem de usar este modelo para a representação, reside no facto de ser possível usar a álgebra linear comum, para o tratamento dos dados.

No entanto, também existem desvantagens, entre as quais: (1) perde-se a estrutura dos documentos; (2) perde-se a posição relativa das palavras nas frases, e os documentos são vistos apenas como um "saco de palavras" (*bag-of-words*), interessando apenas a frequência e o peso relativo de cada palavra; (3) o custo computacional e espacial podem ser elevados para um grande número de documentos e termos de entrada.

De forma a reduzir a dimensionalidade da matriz Termos-Documentos, e consequentemente, o custo computacional e espacial, foi introduzida uma nova técnica em 1990 [5], denominada *LSI - Latent Semantic Indexing*, ou Indexação Semântica Latente. Esta técnica, consiste numa extensão ao Modelo Espaço de Vectores, e tenta resolver além dos problemas já mencionados, o problema das várias semânticas atribuídas pelos humanos às mesmas palavras. A ideia passa por organizar automaticamente os termos com igual semântica, em estruturas mais apropriadas para a recuperação de informação, mas sem a necessidade de usar listas de sinónimos (*thesaurus*). Assim, e fazendo uso do Modelo Espaço de Vectores, a *LSI* é executada recorrendo a um método da álgebra linear denominado *SVD - Singular Vector Decomposition*, ou Decomposição em Vectores Próprios. Neste método, a matriz Termos-Documentos pode ser decomposta em três matrizes

$$A_{|t\times d|} = T_{0|t\times m|} S_{0|m\times m|} D^T_{0|m\times d|} \tag{2.5}$$

onde S_0 é uma matriz diagonal com os valores próprios de A. A matriz D_0 é transposta para ser possível a multiplicação das três matrizes. As matrizes T_0 e D_0 têm colunas formadas por vectores ortonormais, em que se verifica que

$$T_0^T T_0 = D_0^T D_0 = I \tag{2.6}$$

sendo I a matriz identidade, com os valores da sua diagonal a 1, e as restantes posições a 0. Este facto significa que estas colunas são simultâneamente vectores unitários (com tamanho 1), e ortogonais (perpendiculares entre si), formando por isso os vectores base de um espaço vectorial. As matrizes T_0 e D_0 contêm, respectivamente, os vectores próprios esquerdos e direitos da matriz A ou, por outras palavras, a matriz T_0 contém os vectores próprios de A relativos aos termos, e a matriz D_0 contém os vectores próprios de A relativos aos documentos. De forma a reduzir o ruído existente na matriz A, apenas os $k \ll m$ maiores valores de S_0 são retidos, e os restantes são descartados. O resultado é a matriz S. As colunas de T_0 correspondentes aos valores removidos de S_0 são também descartadas, dando origem à matriz T. De igual forma é obtida a matriz D a partir de D_0. A razão para se obterem T e D, consiste no facto de quando se multiplicarem novamente as três matrizes, estas linhas e colunas não contribuírem significativamente para o resultado. Assim, temos

$$A_{|t\times d|} \approx \hat{A}_{|t\times d|} = T_{|t\times k|} S_{|k\times k|} D^T_{|k\times d|} \tag{2.7}$$

onde $\hat{A}$ corresponde a uma aproximação de A, através do método dos mínimos quadrados. Apenas k dimensões são permitidas a $\hat{A}$ para a representação da matriz original A. Desta forma, os termos semelhantes são

compactados, sobressaindo os conceitos latentes dominantes, presentes na colecção. Quanto mais diverso for o texto da colecção, mais dimensões são necessárias (e maior valor de k) para capturar eficazmente os tópicos dominantes. A forma de encontrar o melhor valor de k, ainda é um problema de investigação em aberto.

2.3 Algoritmos de *Clustering*

Clustering consiste em agrupar *automaticamente* i.e., sem intervenção humana, os vários elementos de um conjunto em grupos (*clusters*), em que os elementos de um mesmo grupo são semelhantes entre si, e elementos de grupos separados são diferentes [4]. O *clustering* corresponde, portanto, a aprendizagem não-supervisionada, em que os grupos não são inicialmente especificados, mas sim descobertos através dos dados de entrada.

Da mesma forma, o Web *clustering* consiste em *clustering* aplicado a um conjunto de páginas Web, neste caso retornadas por um ou vários motores de busca. *Clustering* ajuda o utilizador a perceber o agrupamento natural dos documentos numa estrutura de tópicos. É, por isso, interessante agrupar a informação proveniente da Web de uma forma automática, e é aqui que os algoritmos de *clustering* revelam a sua importância. No contexto do presente documento, um elemento é um documento ou página Web, e os *clusters* são os agrupamentos de tópicos dos documentos.

Um bom algoritmo de *clustering* tenta minimizar as semelhanças entre elementos de *clusters* diferentes, ao mesmo tempo que tenta maximizar as semelhanças entre elementos do mesmo *cluster*. Os algoritmos de *clustering* podem ser classificados nas formas que são apresentadas na Figura 2.5.

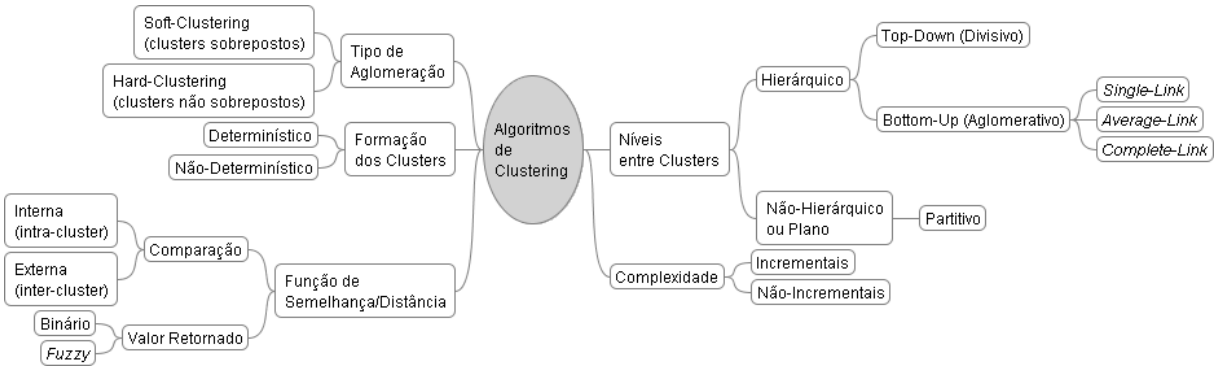


Fig. 2.5: Classificação de Algoritmos de *Clustering*.

Os algoritmos de *clustering* efectuam a aglomeração dos elementos, cujo tipo pode ser classificado em *Hard Clustering* ou em *Soft Clustering* [36]. No *Hard Clustering* cada elemento é atribuído apenas a um *cluster*, e por isso os grupos não se sobrepõem. No *Soft Clustering* cada elemento pode pertencer a mais do que um *cluster*, dando origem a grupos sobrepostos.

Nos algoritmos de *clustering* determinísticos, para os mesmos elementos de entrada, os *clusters* finais formados serão sempre os mesmos, o que não acontece nos algoritmos não-determinísticos, em que duas execuções consecutivas do algoritmo podem dar origem a formações diferentes dos *clusters* [33].

Os algoritmos incrementais de *clustering* efectuam apenas uma passagem sobre todos os elementos, e decidem nessa passagem a que *cluster* esse elemento pertence [50, 44, 17]. Os algoritmos maior complexidade, efectuam mais que uma passagem a cada elemento e, quando isso acontece, os elementos mudam de *cluster* de modo a verificar se a nova distribuição é melhor do que a distribuição anterior.

As secções seguintes descrevem com mais detalhe os restantes tipos de algoritmos de *clustering* descritos na Figura 2.5.

2.3.1 *Clustering* Hierárquico

Quanto ao nível de hierarquias entre *clusters*, o *clustering* pode ser hierárquico e não-hierárquico (ou plano) [20]. No tipo hierárquico, podemos definir relações de hierarquia entre os vários grupos, ao passo que no tipo não-hierárquico todos os grupos estão no mesmo nível. Por exemplo, podemos definir dois grupos com as descrições "desporto" e "ténis", e no tipo hierárquico definir que o grupo "ténis" pertence ao grupo "desporto". No tipo não-hierárquico isso não será possível.

Os métodos hierárquicos de *clustering* podem ser classificados em aglomerativos (*bottom-up*) ou divisivos (*top-down*) [3], como se mostra no exemplo da Figura 2.6. Aglomerativo é o caso em que se começa com os elementos individuais e recursivamente se atribui o elemento ao *cluster* mais apropriado. O divisivo efectua o inverso, começando com um só *cluster* que tem todos os elementos e dividindo-o da forma mais apropriada. O processo continua até que se atinja a um critério de paragem, por exemplo, quando os *clusters* formados ficarem sem alterações entre dois passos consecutivos do algoritmo.

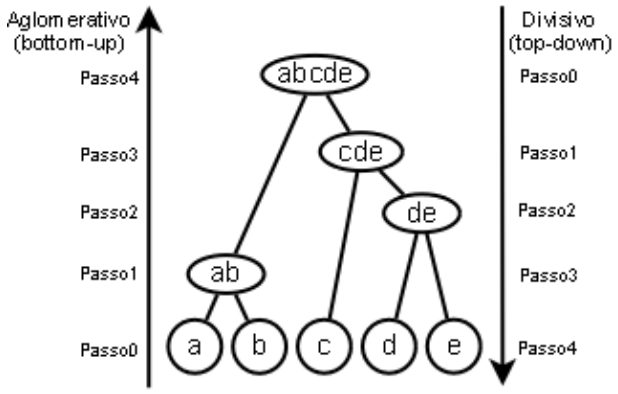


Fig. 2.6: *Clustering* Hierárquico.

O processo final destes algoritmos de *clustering* pode ser visto num tipo particular de árvore, que tem o nome de *dendograma* [16], ilustrada na Figura 2.7.

HAC – Hierarchical Agglomerative Clustering. No caso concreto dos algoritmos hierárquicos aglomerativos, para comparar os grupos entre si é necessário definir uma noção de inter-semelhança. Dados dois *clusters* C_i e C_j, e $d(p, q)$ uma medida de distância entre dois documentos, os critérios de semelhança mais comuns podem ser classificados em três métodos [9, 49]:

1. Ligação-Simples (*single-linkage*) (Figura 2.8(a)):

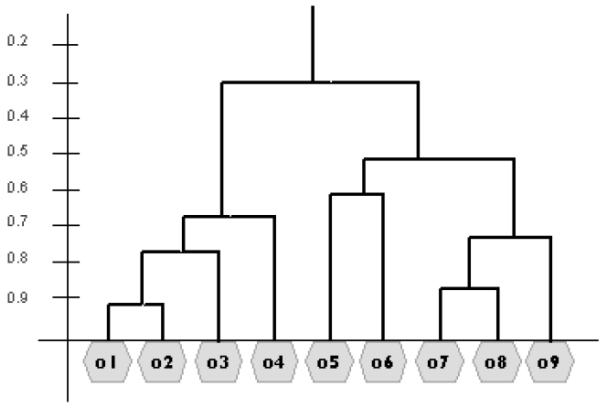


Fig. 2.7: Exemplo de um Dendograma para nove elementos [20].

$$d(C_i, C_j) = min_{p \in C_i, q \in C_j} d(p, q) \tag{2.8}$$

em que a distância é definida pelo par mais próximo de elementos. Neste método, dois *clusters* são combinados com base num único par de elementos muito próximos entre si, apesar dos restantes elementos de cada *cluster* estarem muito afastados, podendo originar *clusters* encadeados (*chaining clusters*) e dispersos.

2. Ligação-Completa (*complete-linkage*) (Figura 2.8(b)):

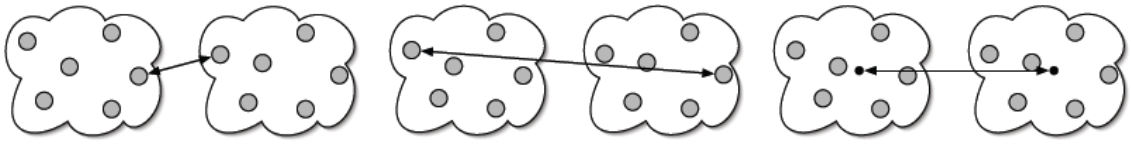
$$d(C_i, C_j) = max_{p \in C_i, q \in C_j} d(p, q) \tag{2.9}$$

em que a distância é definida pelo par mais afastado de elementos. Este método tende a criar *clusters* mais compactos, mas é mais sensível a ruído.

3. Ligação-Média do Grupo (*group average-linkage*) (Figura 2.8(c)):

$$d(C_i, C_j) = \frac{1}{|C_i|\,|C_j|} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q) \tag{2.10}$$

em que a distância é definida pela média das distâncias entre todos os pares de documentos p e q dos *clusters* C_i e C_j, respectivamente. Este é o método mais robusto, pois permite atenuar os problemas dos dois métodos anteriores.



(a) *Single-Linkage* (b) *Complete-Linkage* (c) *Group Average-Linkage*

Fig. 2.8: *Agglomerative Hierarchical Clustering* – Métodos de ligação [42].

Nos métodos hierárquicos, não há necessidade de definir previamente o número de *clusters* [24], constituindo uma vantagem no caso em que é desconhecida a quantidade de tópicos dominantes nos resultados de uma busca, o que acontece geralmente. Outra vantagem é que os algoritmos são determinísticos.

No entanto, e segundo [14, 24], estes métodos também apresentam desvantagens: (1) assim que um *cluster* é construído, o algoritmo não volta a visitar esse *cluster* com intenção de melhorar os membros que lhe pertencem; (2) os documentos podem pertencer a mais do que um *cluster*. Na navegação, estes algoritmos forçam o utilizador a ter de começar com uma categoria (*cluster*), a qual pode ser do desconhecimento deste. Neste caso o utilizador poderá não encontrar o documento nos resultados, ou demora mais tempo a fazê-lo; (3) o custo computacional e espacial são mais elevados. Dependendo do algoritmo usado [20, 49], a complexidade destes métodos pode ser de $O(N^2)$, de $O(N^2 log(N))$ ou mesmo de $O(N^3)$, onde N é o número de documentos.

2.3.2 *Clustering* Não-Hierárquico

Neste tipo de *clustering*, todos os *clusters* estão no mesmo nível, não existindo hierarquias entre si. Os *clusters* são formados num plano, e em geral, de forma partitiva. O algoritmo *K-Means* [21] é o melhor exemplo deste tipo de *clustering*.

***K-Means*.** Um dos algoritmos clássicos de *clustering*, e amplamente usado nas suas muitas variantes é o *K-Means* [21, 4]. Este algoritmo funciona da forma que a seguir se descreve. Dado um número K de *clusters*, são atribuídos K centróides para representar o centro (teoricamente, o "centro de massa") desses *clusters*. Esta atribuição de centróides pode ser aleatória e pode mesmo coincidir com alguns dos pontos do conjunto. Em seguida cada ponto é atribuído ao *cluster* cujo centro centróide é o mais próximo. Os centróides de cada *cluster* são recalculados a cada iteração do algoritmo e o processo repete-se até que os centróides fiquem estáveis. O objectivo do *K-Means* consiste, portanto, em minimizar a função descrita pela equação (2.11)

$$E = \sum_{i=1}^{K} \sum_{x \in C_i} d(x, m_i) \tag{2.11}$$

onde E corresponde ao erro que é definido pela soma das distâncias euclidianas $d(x, m_i)$ entre cada ponto x e o ponto m_i do centro do *cluster* C_i, ao qual o ponto x pertence. O algoritmo gera K *clusters* não-hierárquicos e não sobrepostos, pertencendo a cada *cluster* pelo menos um elemento.

De acordo com [24], o algoritmo *K-Means* tem como vantagens: (1) muito fácil de implementar e (2) a complexidade é de $O(IKNM)$, onde K é o número de *clusters*, N o número de documentos, M o número de termos, e I o número de iterações. Este número de iterações depende da atribuição inicial aleatória dos centróides e da rapidez da convergência do algoritmo para as posições finais dos centróides. A complexidade do algoritmo é por isso linear nas suas dimensões mais importantes, ou seja, nos documentos e termos.

Mas também apresenta as seguintes desvantagens: (1) o resultado final é influenciado pelas condições iniciais, por exemplo, a escolha dos centróides (por outras palavras, o algoritmo é não-determinístico) e (2) é necessário definir um número k inicial de *clusters*, o que nem sempre é possível. Por exemplo, a quantidade de tópicos dominantes num conjunto de resultados de uma busca pode não ser conhecida.

2.3.3 A Função de Semelhança

Para determinar a que *cluster* um elemento pertence é usada uma função de semelhança, que pode ser interna ou externa [20, 33]. A função de *semelhança interna*, avalia a semelhança entre o elemento de um *cluster* e os restantes elementos desse mesmo *cluster*. A função de *semelhança externa*, avalia a semelhança entre o elemento de um *cluster* e os elementos fora desse *cluster*. O valor retornado por esta função de semelhança origina nova classificação de *clustering* [33, 6]:

1. *Binary Clustering* - utilizam uma função binária que indica se um elemento pertence (se a função devolver 1) ou não pertence (se a função devolver 0) a determinado *cluster*;
2. *Fuzzy Clustering* - utilizam uma função que indica a probabilidade (geralmente entre 0 e 1) de um elemento pertencer a determinado *cluster*.

Com a utilização do Modelo Espaço de Vectores, em que cada vector representa um termo presente num documento, a função que determina o grau de semelhança entre dois pares termo/documento, pode ser facilmente calculada pelo ângulo formado entre os dois vectores. Esta função de semelhança é conhecida como *função de semelhança co-seno* em que, dados dois vectores $\overrightarrow{v1}$ e $\overrightarrow{v2}$:

$$s(\overrightarrow{v1}, \overrightarrow{v2}) = cos(\overrightarrow{v1}, \overrightarrow{v2}) = \frac{\langle \overrightarrow{v1} \rangle \cdot \langle \overrightarrow{v2} \rangle}{\| \overrightarrow{v1} \| \| \overrightarrow{v2} \|} \tag{2.12}$$

devolve um valor entre 0 e 1, definindo respectivamente se os dois vectores não são semelhantes (ortogonais) ou são semelhantes (com a mesma direcção e sentido).

Usando a célebre fórmula fundamental da trigonometria:

$$cos^2(\overrightarrow{v1}, \overrightarrow{v2}) + sin^2(\overrightarrow{v1}, \overrightarrow{v2}) = 1 \tag{2.13}$$

sobre a função de semelhança co-seno, a função de distância entre os dois vectores pode ser derivada facilmente, obtendo:

$$d(\overrightarrow{v1}, \overrightarrow{v2}) = sin(\overrightarrow{v1}, \overrightarrow{v2}) = \sqrt{1 - s^2(\overrightarrow{v1}, \overrightarrow{v2})} \tag{2.14}$$

A propriedade mais importante da função de semelhança co-seno, consiste no facto desta não depender do comprimento dos vectores, isto é, $s(\overrightarrow{v1}, \overrightarrow{v2}) = s(\alpha\overrightarrow{v1}, \overrightarrow{v2})$, para $\alpha > 0$. Esta propriedade faz com que a função de semelhança co-seno seja amplamente usada na área de recuperação de informação.

2.3.4 Problemas dos Algoritmos de *Clustering*

Os algoritmos clássicos apresentados, *HAC* e *K-Means*, podem ter alguns problemas ao lidar com certas distribuições dos dados de entrada. A Figura 2.9 ilustra alguns dos problemas comuns, tais como [22]:

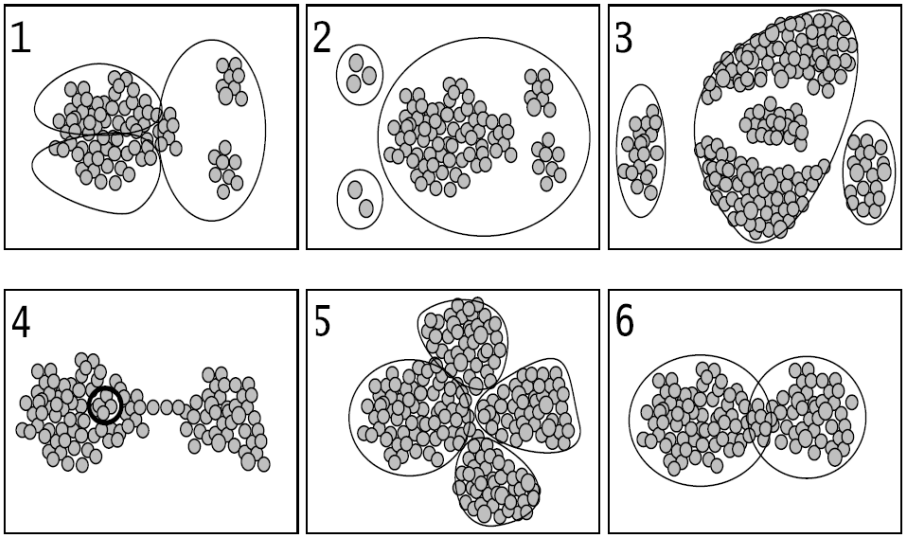


Fig. 2.9: Problemas dos algoritmos clássicos de *clustering* [22].

1. *Clustering* de grupos com tamanho diferente: Se os elementos a agrupar formarem grupos com tamanho desigual, algoritmos clássicos (como o *K-Means*), não efectuarão o *clustering* da melhor forma possível, pois tentam encontrar grupos com formas esféricas de igual tamanho.

2. *Clustering* com elementos muito afastados: Elementos muito afastados e em menor número deverão ser descartados, e não deverão ser atribuídos a nenhum grupo. Por exemplo, nos algoritmos *HAC* que tentam agrupar os grupos mais próximos primeiro, poderá resultar um grupo enorme.

3. *Clustering* de elementos que formam formas não esféricas, como por exemplo, formas côncavas ou alongadas. Nestes casos os algoritmos não encontram os grupos correctos, pois estão preparados para descobrir apenas grupos esféricos formados pelos elementos.

4. O efeito de encadeamento: este efeito, presente nos algoritmos *HAC* de ligação simples, a cada passo tentam adicionar o elemento mais próximo ao grupo, mas sem terem em conta todo o conjunto. Isto pode originar grupos de forma alongada.

5. A necessidade de fornecer, à partida, um número fixo de grupos ou outros parâmetros ao algoritmo. No caso do algoritmo *K-Means*, este tenta obter um número K de grupos, o qual poderá não ser o melhor para a distribuição dos elementos da entrada.

6. Grupos sobrepostos: alguns algoritmos clássicos criam partições bem definidas dos grupos, mas em casos reais (como por exemplo, no *clustering* de páginas Web) é mais natural que um elemento pertença a mais que um grupo.

Adicionalmente a todos estes problemas dos algoritmos clássicos, no caso do *clustering* de texto (ou de documentos Web), existem ainda outros tipos de problemas [39], nomeadamente:

- Descrições sumárias dos grupos, ou seja, o problema da descrição a dar a cada grupo, para que o seu tópico descreva bem o seu conteúdo. O utilizador necessita de saber à partida, e de forma concisa e

precisa, quais os grupos que poderão ser do seu interesse. Se este problema for ignorado, então o utilizador poderá escolher um grupo que lhe é irrelevante, perdendo tempo a ver resultados que não lhe interessam, e que é precisamente um dos problemas que o *clustering* tenta resolver em relação às comuns listas ordenadas de resultados.

- Documentos muito pequenos. Como os utilizadores não podem esperar pelo processamento dos documentos originais, o *clustering* é efectuado apenas com as pequenas descrições (*snippets*) retornadas pelos motores de busca, evitando assim a pesquisa pelos documentos originais. Estas pequenas descrições têm, por vezes, as frases cortadas. Desta forma, os algoritmos têm de ser tolerantes à falta de palavras, procurando nos *snippets* as melhores descrições para os tópicos.

- Outro problema diz respeito à eficiência dos algoritmos *HAC*. O custo computacional e espacial torna estes métodos proibitivos para um grande número de documentos, e também para um sistema onde o utilizador não pode esperar os resultados durante um longo período de tempo, como é o caso de um motor de pesquisa com capacidades de *clustering*. Neste caso existe a necessidade de usar novos algoritmos, que efectuem o *clustering* em tempo útil.

- Incrementabilidade. De forma a poupar o tempo de espera do utilizador, os algoritmos devem efectuar o processamento de um documento assim que este chega da Web, em vez de se iniciar o processamento apenas quando todos os documentos foram obtidos.

Capítulo 3

Trabalho Relacionado

Este Capítulo apresenta trabalhos anteriores efectuados sobre o pré-processamento de texto. Em seguida, descrevem-se alguns algoritmos de *clustering*, com foco nos que foram desenvolvidos especialmente para aplicação na Web e apresentam-se os motores de busca onde esses algoritmos foram pela primeira vez implementados.

3.1 Filtragem e Pré-Processamento

Esta secção descreve o trabalho relacionado efectuado na etapa de pré-processamento, nomeadamente nas colecções de palavras Portuguesas frequentes e dos radicalizadores para a língua Portuguesa.

3.1.1 Palavras Frequentes (*stopwords*)

As *stopwords* são palavras demasiado frequentes para serem informativas. Por exemplo, “e”, “ou”, “que”, entre outras. Na prática, a lista de palavras frequentes depende do contexto de aplicação [7]. Por exemplo, a palavra "A" poderá ser uma palavra frequente num texto sobre engenharia, mas corresponde ao nome de uma vitamina num texto sobre medicina. A lista deverá, por isso, ser cuidadosamente elaborada de modo a serem escolhidas palavras que se adaptem a vários contextos. Uma lista de *stopwords* que tenha certas palavras em falta poderá, na fase final de processamento, originar tópicos sem significado. Por outro lado, uma lista com demasiadas palavras poderá, também na fase final, fazer com que tópicos importantes sejam descartados. De forma a avaliar estas listas, bem como a produzir colecções de textos previamente classificados, apareceram alguns grupos de trabalho, tais como o *TREC*[1] (*Text REtrieval Conference*). No entanto, este grupo tem como alvo apenas colecções de textos em Inglês. De forma a ultrapassar este problema, nasceram outros grupos, tais como o *CLEF*[2] (*Cross Language Evaluation Forum*) em 2000, e cujo foco de trabalho são as línguas da Europa, incluindo a Portuguesa.

Em Portugal, a Linguateca Portuguesa[3] é um centro distribuído de recursos para o processamento computacional da língua Portuguesa, com o apoio da *FCCN* (Fundação para a Computação Científica Nacional).

[1] http://trec.nist.gov
[2] http://www.clef-campaign.org
[3] http://www.linguateca.pt

Desde 2004, a participação da Linguateca na organização do *CLEF* consiste na colecção *CHAVE*[4], cujo nome é justamente a tradução Portuguesa da palavra Francesa *clef*. Esta colecção inclui listas de palavras frequentes Portuguesas, as quais foram usadas no âmbito do presente trabalho.

Sendo a lista de *stopwords*, uma lista das palavras mais frequentes de uma determinada linguagem, poderá também ser usada com o objectivo de adivinhar a linguagem em que um determinado texto foi escrito, de modo a adaptar ou escolher os restantes algoritmos das fases seguintes de processamento. Um exemplo é a escolha do algoritmo de radicalização.

3.1.2 Radicalização de Palavras (*stemming*)

A radicalização de palavras, também conhecida por *stemming*, consiste na redução de palavras ao seu radical. Por exemplo, na família de palavras *terra*, *terrinha*, *terriola*, *térreo*, *terráqueo*, *terreno*, *terreiro*, *terroso*, existe um elemento comum: *terr-*, que é o radical. Para a língua inglesa, o algoritmo de radicalização mais conhecido e usado é o algoritmo *Porter* [32]. Este algoritmo, funciona por truncar sucessivamente letras da palavra, de acordo com regras pré-definidas para os prefixos, sufixos, etc., até se chegar à raiz da palavra [7]. As regras são definidas por expressões regulares, em que os termos da expressão são vogais, consoantes ou letras. A Figura D.1, no Apêndice D, ilustra o fluxo do algoritmo *Porter*. Posteriormente, foram feitas adaptações ao algoritmo para outras linguagens, incluindo a Portuguesa, dando origem ao algoritmo *Porter PT*.

Um dos algoritmos radicalizadores desenvolvidos especialmente para a língua Portuguesa, é o algoritmo *RSLP*, Removedor de Sufixos da Língua Portuguesa [25]. Este algoritmo é composto por oito passos, em que cada um tem um conjunto de regras, mas apenas uma regra pode ser aplicada por passo. Testes efectuados pelos autores do *RSLP* [25] com um conjunto de 1000 documentos pré-radicalizados manualmente, mostraram que o algoritmo consegue obter 96% de raízes de palavras correctamente, quando comparado com os 71% obtidos pelo *Porter PT*. A Figura E.1, no Apêndice E, ilustra o fluxo do algoritmo *RSLP*.

O algoritmo implementado na *framework* `Carrot`, que será descrita na secção 3.3.3, é o *Snowball*. Na realidade, o *Snowball* não é um radicalizador, mas sim uma linguagem de *scripting* que permite representar facilmente algoritmos de radicalização, que depois são traduzidos automaticamente em código C ou Java, para posterior compilação. O radicalizador Português incluído no *Snowball* é, portanto, outra versão Portuguesa do algoritmo *Porter* inglês, mas cuja implementação é diferente do algoritmo *Porter PT* descrito anteriormente.

3.2 Algoritmos de *Clustering* aplicados à Web

Esta secção apresenta o estudo do trabalho relacionado com os algoritmos de *clustering*, com particular foco nos que foram desenvolvidos especialmente para aplicação na Web.

[4] http://www.linguateca.pt/CLEF

3.2.1 *Scatter/Gather*, *Buckshot* e *Fractionation*

O algoritmo `Scatter/Gather` [8, 36] foi um dos pioneiros a efectuar *clustering* sobre um conjunto de documentos. Faz uso de duas rotinas eficientes, chamadas *Buckshot* e *Fractionation* de modo a procurar os centróides dos *clusters* a formar. Num primeiro passo, semelhante ao algoritmo *K-Means*, começa por atribuir os documentos da colecção ao centróide mais próximo e recalcula os centróides iterativamente até não seja observada nenhuma diferença significativa em relação ao passo anterior. No entanto, ao contrário do *K-Means*, inicialmente os centróides não são atribuídos de forma aleatória. Tal atribuição é feita recorrendo à rotina *Buckshot*, a qual encontra K centróides no conjunto retirando uma amostra de $\sqrt{KN}$ documentos e atribuindo-lhes K *clusters* usando uma rotina de *clustering* aglomerativo hierárquico [43]. Esta rotina, cuja complexidade é de $O(N^2)$, juntamente com a *Buckshot* resulta então numa complexidade combinada de $O(KN)$.

A rotina *Fractionation*, por outro lado, procura os K centróides da seguinte maneira: divide o conjunto de documentos em caixas de tamanho M, em que $M > K$. Em seguida atribui cada caixa a um conjunto de PM *clusters*, onde $P < 1$ é uma constante. O processo é repetido, tratando os *clusters* formados como elementos individuais, até se obterem K *clusters*. Resumindo, a rotina *Fractionation* é uma aproximação do *clustering* aglomerativo hierárquico, onde a procura de dois *clusters* semelhantes é feita localmente (para cada caixa) e não globalmente.

Os centros dos *clusters* formados pelas duas rotinas são então usados como pontos centróides iniciais para o algoritmo *K-Means*. Os autores, em [8], mostraram ainda que a complexidade da rotina *Fractionation* é de $O(MN)$.

O algoritmo `Scatter/Gather` apresenta as seguintes desvantagens [36, 34]: (1) tal como o *K-Means* é necessário definir um número inicial de K grupos que, se for maior que o número de tópicos dominantes dos documentos do conjunto, leva a fracas descrições atribuídas aos grupos; (2) os grupos não são distintos, isto é, um mesmo documento pode pertencer a vários grupos; (3) as descrições atribuídas aos grupos nem sempre são as mais apropriadas, em parte devido à sobreposição dos grupos. A principal vantagem do algoritmo é que o utilizador pode desconhecer por completo os tópicos dominantes, e por este motivo não é obrigado a fornecer termos para dar início à pesquisa.

3.2.2 TRC – *Tolerance Rough Clustering*

O algoritmo *TRC*, *Tolerance Rough Clustering* [19], é baseado no algoritmo *K-Means*. No entanto, ao invés de considerar os documentos como pertencentes ou não a um dado *cluster*, o algoritmo tenta encontrar classes de tolerância construídas com base nas semelhanças entre os documentos. Esta semelhança é medida por funções que indicam o grau de relação entre os documentos através dos termos que têm em comum, mas também tendo em consideração a ocorrência destes em todo o conjunto. Pode então ser usado um limite pré-definido, para indicar se o documento pertence ou não a uma determinada classe. Assim, o algoritmo *TRC* permite *soft-clustering* em contraste com o *hard-clustering* proporcionado pelo *K-Means*. Ao mesmo tempo, o *TRC* mantém a mesma eficiência do *K-Means*, mas tenta obter descrições para os *clusters* com melhor qualidade. As descrições são construídas através de n-gramas de palavras (frases de tamanho até

n) recolhidas dos documentos que compõem cada classe.

3.2.3 STC – *Suffix Tree Clustering*

O algoritmo *Suffix Tree Clustering* [47] não trata os documentos apenas como um conjunto de palavras, mas sim como frases, fazendo o uso da informação de proximidade e de ordem entre as palavras. O *STC* baseia-se numa árvore de sufixos para identificar de forma eficiente os grupos base de documentos que partilham frases em comum.

A Figura 3.1 mostra um exemplo da construção da árvore de sufixos para as frases 1-*"cat ate cheese"*, 2-*"mouse ate cheese too"* e 3-*"cat ate mouse too"*. Em cada rectângulo temos o par $\langle p, w \rangle$ onde w corresponde à primeira ocorrência da palavra na frase p.

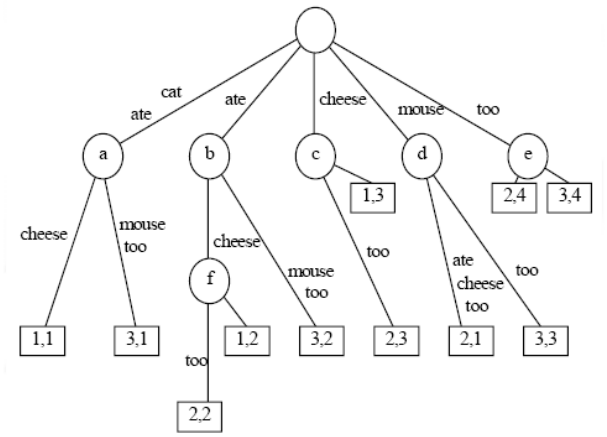


Fig. 3.1: Exemplo clássico de *Generalized Suffix Tree* [50].

O algoritmo tem três passos:

1. Limpeza - onde é aplicado o *stemming*, delimitação das frases, e remoção de *stopwords* e dos termos nas frases que não sejam palavras (como números ou *tags* HTML).

2. Identificação dos *clusters* base - este passo consiste na criação de um índice das frases. Isto é realizado de uma forma eficiente, recorrendo a uma árvore de sufixos, construída incrementalmente. Cada nó da árvore representa um grupo (*cluster*) de documentos e uma frase que é comum a todos eles. Estes nós constituem portanto os *clusters* base. Cada *cluster* base a tem uma frase associada m_a, e é representado pelos documentos d_a que contêm essa frase. Em seguida é calculada a pontuação (*score*) desse *cluster* base a, definida pela equação (3.1)

$$s(a) = |m_a| \times f(|m_a|) \times \sum_{w \in m_a} tfidf(w) \tag{3.1}$$

onde $|m_a|$ é o número de termos na frase m_a, $f(|m_a|)$ é a função que penaliza as frases pequenas (os autores preferem as frases maiores, porque produzem descrições com melhor qualidade) e $\sum_{w \in m_a} tfidf(w)$ é o somatório da função clássica TF-IDF (*term frequency-inverse document frequency*) aplicada a cada palavra w da frase m_a [37]. Só os *clusters* base com uma pontuação maior que um limite configurável, são promovidos para o terceiro passo do algoritmo.

3. Combinação dos *clusters* base - neste passo os *clusters* identificados no passo anterior são agrupados, tendo em conta a sobreposição dos documentos que representam, para formar os *clusters* finais. Evita-se assim a existência de vários grupos quase semelhantes ou mesmo idênticos. A combinação de dois *clusters* base a e b é efectuada quando o conjunto dos documentos que os compõem, respectivamente d_a e d_b se sobrepõem mais que um determinado limite. Assim, o *STC* agrupa os documentos que relacionam a mesma ideia descrita em mais que uma frase. O processo de combinação é uma variante do algoritmo de *clustering* aglomerativo hierárquico, onde α define o limite e $|x|$ define o número de documentos no *cluster* x, o critério de combinação do algoritmo é definido pela equação (3.2)

$$similarity(a,b) = 1 \Leftrightarrow \left(\frac{|d_a \cap d_b|}{|d_a|} > \alpha\right) \wedge \left(\frac{|d_a \cap d_b|}{|d_b|} > \alpha\right) \tag{3.2}$$

   Finalmente, a pontuação é recalculada para cada novos *clusters* combinados e no final os *clusters* com maior pontuação são mostrados ao utilizador.

As vantagens do algoritmo *STC* são: (1) o algoritmo é linear, sendo mesmo possível construir a árvore de sufixos à medida que os documentos chegam da Web; (2) guarda informação da posição das palavras em cada frase, o que dá origem a boas descrições para identificar cada *cluster*; (3) o algoritmo permite a sobreposição de grupos, isto é, um documento pode pertencer a mais do que um *cluster*. A grande desvantagem do *STC* é que os *clusters* não possuem uma hierarquia.

3.2.4 Lingo – *Description-Comes-First*

O algoritmo *Lingo* [26] foi a primeira implementação da abordagem *DCF - Description Comes First*, ou em Português, as "Descrições (dos *clusters*) Primeiro". A grande motivação é o facto das abordagens clássicas só no final da construção dos *clusters* atribuírem uma descrição para cada *cluster*, o que por vezes dava origem a descrições de fraca perceptibilidade aos utilizadores. Assim, o algoritmo procura primeiro as descrições directamente nos *snippets* dos resultados de entrada, para formar cada *cluster*, e só depois são adicionados os documentos a cada *cluster*.

A Figura 3.2 ilustra a nova abordagem seguida no *Lingo*, que consiste em cinco fases [28, 27]:

(a) clássico (b) DCF

Fig. 3.2: *Clustering*: Abordagem clássica *vs* DCF [30].

1. Pré-processamento - onde o texto dos *snippets* da entrada é separado em termos, onde é feita a tentativa de adivinhar qual é a linguagem do documento, de forma a aplicar o algoritmo de radicalização apropriado, e finalmente onde são marcadas as palavras frequentes.

2. Extracção de frases frequentes - os termos e frases frequentes são descobertos, e as frases menos frequentes que um determinado limite são descartadas.

3. Indução de descrições - neste passo, é atribuída uma descrição a cada *cluster*. É usado um método *SVD* descrito na Secção 2.2.5, para extrair os vectores próprios da matriz Termos-Documentos, os quais se espera que representem os tópicos distintos dos dados da entrada. Tópicos com descrições muito semelhantes são descartados.

4. Descoberta do conteúdo das descrições - onde são adicionados os documentos a cada *cluster*. As descrições descobertas são usadas como termos de pesquisa nos documentos. Os documentos com a maior pontuação são então adicionados ao *cluster* respectivo.

5. Formação final de *clusters* - é aplicada uma função de pontuação e semelhança sobre os *clusters* formados, de modo a fundi-los caso necessário, e em seguida ordená-los para a apresentação final.

Uma explicação detalhada sobre os vários passos do algoritmo pode ser encontrada em [26].

Mais tarde, os autores melhoraram o algoritmo *Lingo*, dando origem ao algoritmo *Lingo3G*, que tem melhor desempenho e mostra *clusters* hierárquicos. No entanto, não são conhecidos os detalhes das alterações efectuadas, visto que o algoritmo *Lingo3G* é de aplicação comercial.

3.3 Motores de Busca com *clustering* de Resultados

Em seguida apresentam-se alguns motores de busca com capacidade de efectuar *clustering* dos resultados, e que implementam alguns dos algoritmos descritos na secção anterior.

3.3.1 *Scatter/Gather*

A primeira implementação do algoritmo *Scatter/Gather*, descrito na secção 3.2.1, foi no sistema que tinha o mesmo nome [8, 36]. O sistema *Scatter/Gather* foi desenvolvido nos laboratórios Xerox PARC, e permitia a navegação num conjunto alargado de documentos, sem ser necessário sequer fornecer termos de busca inicial explicitamente. O sistema conhecia à partida todo o universo dos documentos, que na altura eram notícias publicadas no *New York Times News Service*, e sobre este procurava os tópicos dominantes, os quais eram depois mostrados ao utilizador.

O método pretende, de uma forma iterativa e interactiva, convidar o utilizador a explorar novos grupos (*clusters*) que lhe vão sendo apresentados, à medida que este vai interagindo com os resultados devolvidos por cada iteração do algoritmo. Assim, inicialmente o sistema começa por espalhar (*scatter*) todos os documentos do conjunto, em seguida agrupando-os em pequenos grupos (*clusters*), dando a cada um deles uma pequena descrição, que corresponde aos termos mais frequentes encontrados. Os grupos são apresentados ao utilizador, e este escolhe um ou vários do seu interesse, através da sua descrição, e despoleta nova

iteração. O sistema então combina (*gathers*) os subgrupos escolhidos, espalha-os e agrupa-os novamente, atribuindo-lhes novas descrições. A cada iteração, os subgrupos são mais específicos e vão ficando com menos resultados, até se chegar aos documentos individuais. A Figura 3.3 mostra um exemplo de uma sessão *Scatter/Gather* aplicada sobre as notícias mensais de Agosto de 1990 no *New York Times News Service*. Na primeira interacção o utilizador escolhe os grupos (que também correspondem aos termos da busca) *"Iraq"*, *"Oil"* e *"Germany"* dos grupos que lhe são inicialmente mostrados. Na segunda interacção escolhe os grupos *"Pakistan"* e *"Africa"*.

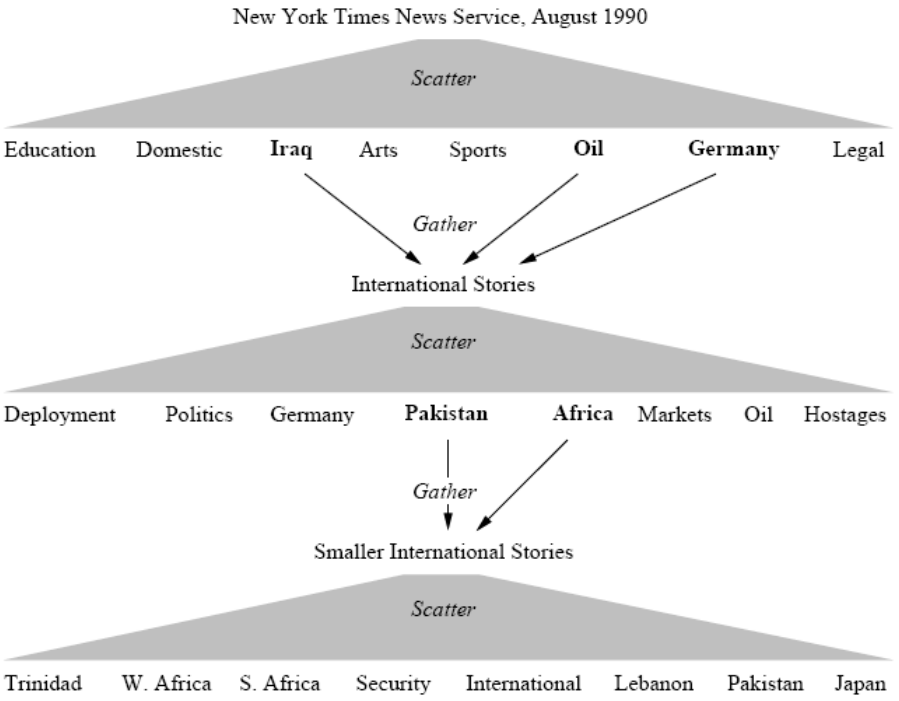


Fig. 3.3: Exemplo de sessão `Scatter/Gather` [8].

Mais tarde, Hearst e Pedersen [15] aplicaram o algoritmo em *clustering* de páginas Web retornadas por um motor de busca.

A grande vantagem do sistema `Scatter/Gather` é que o utilizador pode desconhecer por completo os tópicos dominantes, pois não é obrigado a fornecer termos para dar início à pesquisa [36, 34].

3.3.2 Grouper, HuskySearch e MetaCrawler

O sistema `Grouper` [48, 50], desenvolvido na Universidade de *Washington*, foi o primeiro sistema concebido especificamente para efectuar *clustering* de páginas Web. O algoritmo implementado no `Grouper` foi o *STC* (ver 3.2.3). O sistema consistia numa interface, mostrada na Figura 3.4, que permitia agrupar os resultados das buscas provenientes de um motor chamado `HuskySearch`, o qual apenas fornecia uma lista ordenada, com base nos resultados do motor `MetaCrawler`. O `MetaCrawler` é um *Meta-Searcher* que efectua pedidos em paralelo a vários motores de busca individuais (cerca de 10). O `HuskySearch` foi também usado para estudos relacionados com Recuperação de Informação e Inteligência Artificial, e as suas aplicações em buscas na Web.

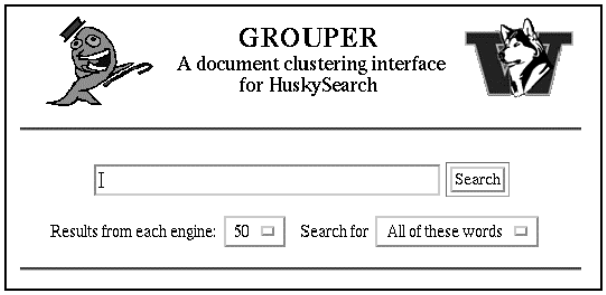


Fig. 3.4: Interface do sistema `Grouper I` - Os utilizadores não necessitam de especificar parâmetros para o algoritmo de *clustering* [48].

O sistema `HuskySearch` permitia também melhorar os termos de busca que eram introduzidos pelo utilizador, por exemplo, fazendo sugestões de novos termos. A página dos resultados do `Grouper` mostrava o número de resultados e o número de *clusters* formados. Os resultados eram apresentados numa tabela (Figura 3.5), com cada linha correspondendo a um *cluster*. Os *clusters* aparecem ordenados pela sua ordem

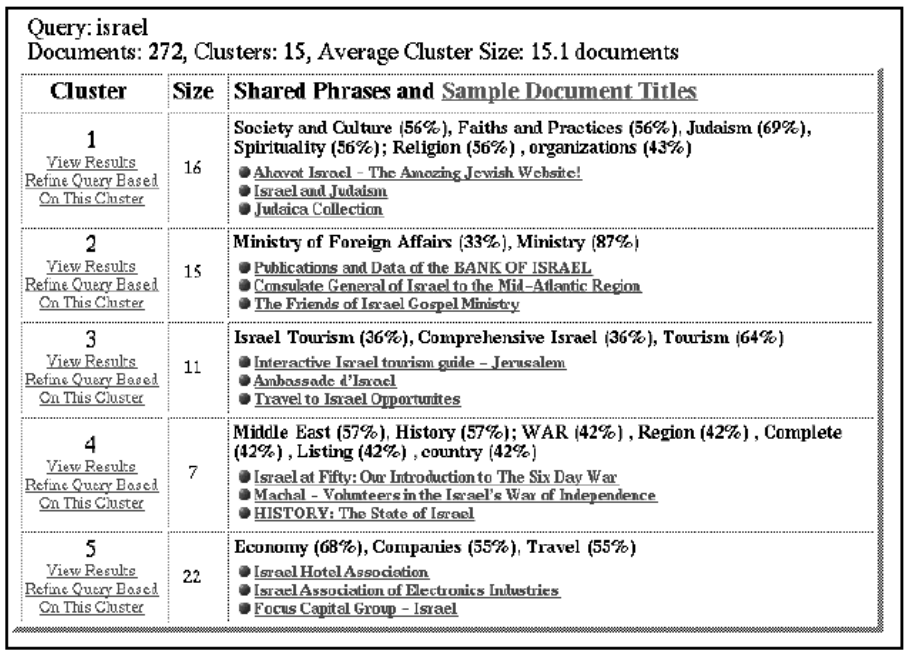


Fig. 3.5: `Grouper I` - Resultados para a busca “israel”, em 1999 [48].

de coerência. O sumário de cada *cluster* inclui o seu tamanho (o número de documentos que agrega), e tenta descrever o conteúdo desses documentos através das frases partilhadas e exemplos de descrições. As frases partilhadas são essencialmente aquelas que aparecem maioritariamente nos documentos que formam o *cluster*. Os números entre parêntesis indicam a percentagem de documentos que contêm a frase, e são também indicados os títulos de três documentos exemplo no sumário de cada *cluster*. Se os resultados forem do interesse do utilizador, este poderá usar a ligação dos documentos exemplo, ou então usar a ligação *“View Results”*, a qual abrirá uma página com os resultados, em que a cada um corresponde uma pequena descrição (*snippet*) do documento. A interface tem ainda a opção de refinar os termos de procura para um determinado *cluster*. Neste caso aparecerá uma nova página (Figura 3.6), que sugere novos termos

de pesquisa num *cluster* ao utilizador.

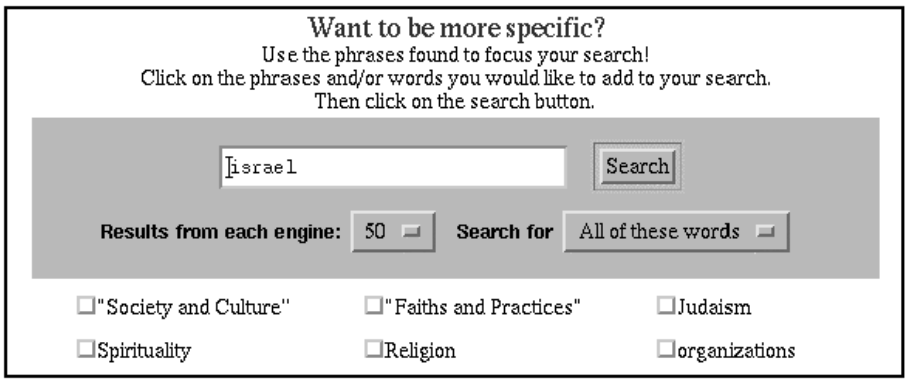


Fig. 3.6: `Grouper I` - Exemplo de refinamento de consulta "israel" usando o primeiro *cluster* [48].

Os autores usaram ainda a informação proveniente dos *logs* do `Grouper` e do `HuskySearch` para novos estudos sobre os documentos visitados e o tempo despendido pelos utilizadores na navegação através dos dois sistemas. Com o auxílio desta informação, eles detectaram alguns problemas. Por exemplo, algumas vezes o sistema construía *clusters* com frases comuns nos documentos, mas que não tinham nenhum significado para o utilizador, tais como *"This was last updated on"*. O número de *clusters* também aumentava bastante com o número de documentos da busca, o que levava os utilizadores a terem de percorrer muitos documentos para encontrarem algo relevante, tornando o *clustering* pouco eficaz.

Para resolver estes problemas, decidiram então criar o `Grouper II`, o qual apresentou três novos componentes:

1. Índice Dinâmico - permite que os utilizadores vejam as frases dos *clusters* e eliminem as irrelevantes, antes de se formarem os *clusters* base. Este índice é gerado a partir de um sub-conjunto da árvore de sufixos do *STC*.

2. Múltiplas vistas - permite os utilizadores verem os resultados através do índice dinâmico, *clusters* ou lista ordenada.

3. Navegação interactiva e hierárquica - interface baseada no sistema `Scatter/Gather`, onde os utilizadores podem seleccionar um ou mais documentos ou *clusters* de interesse, definindo assim sub-conjuntos dos documentos. Isto permite uma apresentação hierárquica, em vez de se ter apenas muitos *clusters* para navegação.

A Figura 3.7 mostra a interface de índice dinâmico, de uma busca retornada para o termo *"clinton"*, em 1999.

Informação relativa aos motores `MetaCrawler`, `HuskySearch` e `Grouper` pode ser encontrada na Web[5], embora estes serviços já tenham sido desactivados para o público geral.

[5] http://www.cs.washington.edu/research/clustering

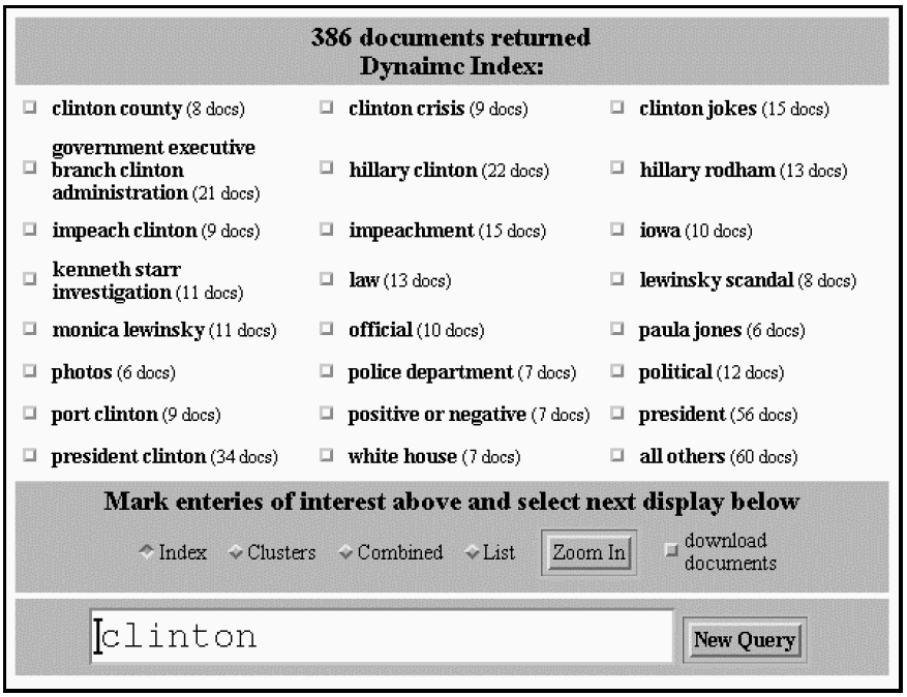


Fig. 3.7: `Grouper II` - Interface de Índice Dinâmico [48].

3.3.3 A *framework* Carrot

A primeira implementação do algoritmo *Lingo* foi realizada na *framework* `Carrot` [26, 30, 28, 27]. Os autores tiveram como inspiração os trabalhos efectuados no sistema `Grouper`. `Carrot` é um motor do tipo *Meta-Search*, de código aberto, que facilita a implementação, estudo e desenvolvimento de busca e processamento de páginas Web, bem como de novos algoritmos de *clustering* para a Web.

A arquitectura é modular, e é composta por quatro tipos essenciais de componentes:

1. Entrada (*input*) - o papel principal deste componente é obter e gerar dados para processamento, baseados numa busca de termos, geralmente introduzida por um utilizador. Exemplos destes componentes incluem *wrappers* para motores de busca, conjuntos de textos, ou mesmo componentes que geram texto aleatório. São por isso componentes que produzem informação para posterior processamento.

2. Filtro (*filter*) - estes componentes transformam uns dados de uma determinada forma. Exemplos destes componentes incluem segmentação de texto, radicalização de palavras, extracção de propriedades, *clustering* ou classificação. O algoritmo *Lingo*, por exemplo, é implementado no `Carrot` sob a forma de componente de filtro. A primeira implementação do algoritmo *STC* em código aberto foi também efectuada no `Carrot`.

3. Saída (*output*) - são os componentes finais na cadeia de processamento e são responsáveis por consumir a informação dos componentes anteriores de alguma forma. Geralmente apresentam os dados finais ao utilizador, mas também podem ser usados para efeitos de optimizações ou de ajustes.

4. Controlador (*controller*) - responsável por combinar os três componentes anteriores numa cadeia de processamento: os dados entram no componente de Entrada, são processados por componentes de

Filtro, e são finalmente consumidos num componente de saída. Um utilizador apenas interage com este componente.

Os componentes são independentes, e podem ser combinados de modo a construir um fluxo de processamento, como se mostra na Figura 3.8.

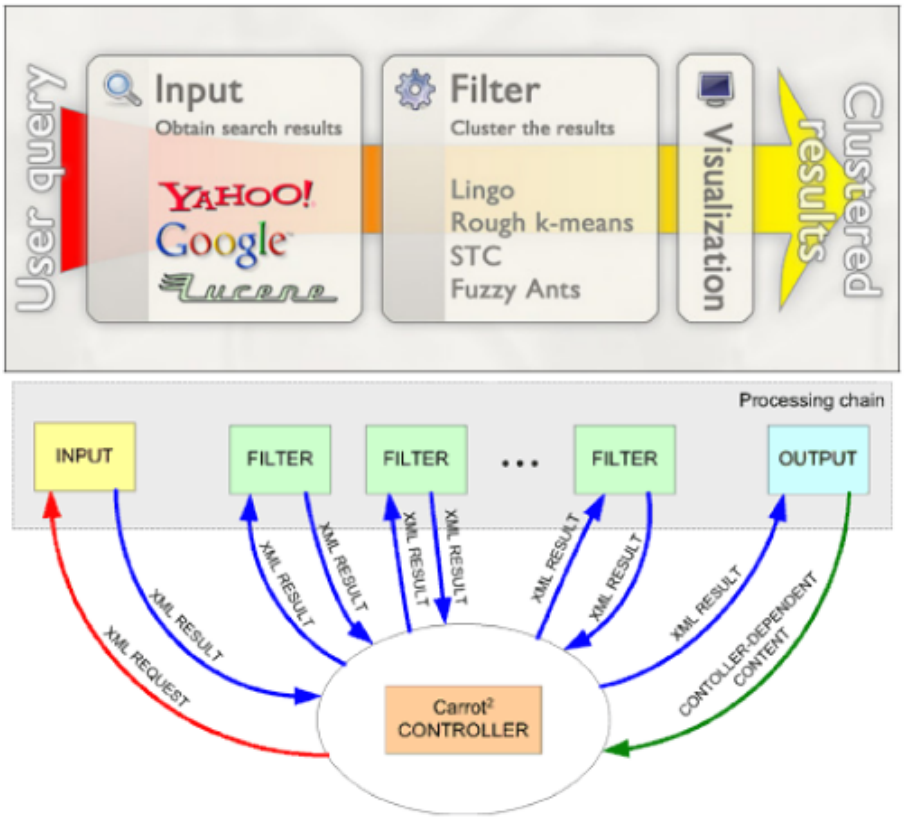


Fig. 3.8: Carrot^2: Cadeia de processamento [26, 30].

Os requisitos iniciais do Carrot^2 incluíam a necessidade de módulos, coisa que não existia no primeiro Carrot. Outros requisitos eram a flexibilidade e o processamento eficiente dos dados. Como estes dois requisitos são difíceis de alcançar mutuamente, Para garantir simultâneamente a flexibilidade e um processamento eficiente dos dados, o Carrot^2 possui duas arquitecturas de comunicação entre os componentes, como ilustrado na Figura 3.9.

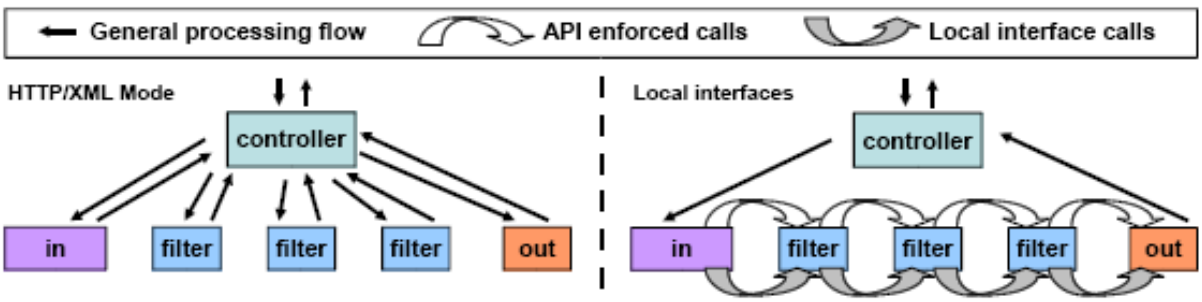


Fig. 3.9: Carrot^2: Duas formas de comunicação entre os componentes [29].

A primeira arquitectura é baseada em XML. Neste caso, os componentes comunicam entre si usando

apenas o protocolo HTTP, trocando mensagens em XML. A comunicação é mediada pelo componente controlador, que conhece a ordem dos componentes na cadeia de processamento. As vantagens desta arquitectura são: (1) os componentes podem ser distribuídos entre várias máquinas; (2) podem também ser escritos em várias linguagens de programação, desde que estas sejam capazes de lidar com o protocolo HTTP e com documentos em XML; (3) e não precisam de saber onde os dados são produzidos. Finalmente, a configuração é centralizada no componente controlador, o que torna possível a construção de um sistema capaz de balancear os pedidos e tolerante a faltas.

A segunda arquitectura é baseada em interfaces locais. Neste caso, a comunicação entre os componentes é feita através de chamadas API directas. No entanto, a *framework* nada conhece sobre estas chamadas, visto que os componentes são combinados de modo a formar a cadeia de processamento já em tempo de execução.

As chamadas API são divididas em dois tipos: sistema e aplicação. As chamadas API de sistema são definidas no núcleo do `Carrot`^{2}. Incluem, por exemplo, métodos para gestão do ciclo de vida dos componentes, que todos têm de implementar. As chamadas de API de aplicação entre os componentes são desconhecidas durante a compilação, pelo que os componentes em tempo de execução têm estabelecer um acordo inicial (*handshake*) antes de determinar e usar os métodos compatíveis entre si. Isto é feito dinamicamente através do componente controlador, o qual conhece a ordem dos componentes na cadeia. Cada componente declara as suas capacidades e as capacidades que espera do componente antecessor. A cadeia final só é formada quando todos os componentes tiverem feito o acordo com o seu antecessor e sucessor.

As vantagens desta arquitectura são: (1) melhor desempenho; (2) reutilização de objectos e memória, pois há partilha de estruturas entre componentes; (3) os dados podem ser passados entre os objectos à medida que vão sendo produzidos. Finalmente, os tipos de dados entre os componentes são mais flexíveis, e podem ter maior complexidade.

Actualmente está disponível uma versão de demonstração do `Carrot`^{2}[6], juntamente com o seu código fonte. Adicionalmente ao `Carrot`^{2}, foram concebidas aplicações para visualizar, testar e experimentar em tempo real os efeitos das alterações nos parâmetros de configuração dos vários algoritmos de *clustering*. Um exemplo é a aplicação `Carrot Workbench`[7], que consiste numa aplicação separada do `Carrot`^{2} mas que usa as mesmas APIs, e permite efectuar buscas na Web e na máquina local. Nesta aplicação o utilizador pode escolher as fontes a usar, suportando motores de busca convencionais para buscas na Web, e também aplicações locais de indexação de documentos, tais como o `Google Desktop`[8] e `Lucene`[9], o que permite efectuar *clustering* sobre documentos indexados localmente.

Na construção do protótipo apresentado neste trabalho foi usada a versão 3.0 da *framework* `Carrot` [18].

Nesta versão, foi introduzido o conceito de "atributo". Um atributo é uma variável que afecta o comportamento de certos componentes do `Carrot` (no caso de um atributo de entrada), ou transporta o resultado de processamento de um componente (no caso de um atributo de saída). No `Carrot`, os atributos podem ter

[6] http://demo.carrot2.org/demo-stable/main

[7] http://project.carrot2.org/download-workbench-win32.html

[8] http://desktop.google.com

[9] http://lucene.apache.org

diferentes escopos:

1. Escopo de instância - neste escopo, o atributo é inicializado uma vez, no momento da criação de um componente. O componente pode então ser reutilizado para responder a vários pedidos, sem que o valor do atributo se altere.

2. Escopo de pedido - neste caso o atributo é inicializado para cada pedido, e o seu valor é obtido e colocado no contexto de execução, para cada passo. Os atributos estão associados a uma variável Java nas classes dos respectivos componentes, e têm por isso um tipo, valor por omissão, e informação adicional que afecta o seu comportamento e ligação (*binding*) em tempo de execução.

Na versão 3.0, os componentes são igualmente geridos por um componente "controlador". O controlador é responsável pela criação de componentes, ligação do valor dos atributos de entrada, recolha do valor dos atributos de saída, e limpeza de ambiente no final da execução. Também é responsável pela gestão multi-tarefa dos componentes. A Figura 3.10(a) mostra o ciclo de vida de um componente, para um pedido simples. No entanto, geralmente são necessários mais componentes para responder a um pedido. Nestes casos, o controlador mantém um mapa de atributos para cada pedido, de forma análoga a um contexto de execução numa aplicação Web. Este mapa de atributos é usado por cada componente ao longo do fluxo de processamento, como se pode ver na Figura 3.10(b).

3.3.4 CarrotSearch e Grokker

Após o `Carrot`^{2}, os seus dois autores principais, Dawid Weiss e Stanislaw Osinski, criaram ainda uma versão comercial denominada `Carrot Search`[10], na qual foi implementado o algoritmo *Lingo3G*. O *Lingo3G* foi também implementado no motor de busca comercial `Grokker`[11]. Este motor, além de mostrar os *clusters* sob a forma clássica de texto, tal como no `Carrot`^{2} ou muitos outros, a sua interface tem ainda a vista *"Map View"* a qual mostra os *clusters* graficamente sob a forma de círculos, ou sob a forma de quadrados.

A Figura 3.11 mostra os *clusters* formados para a busca *"jaguar"*. Quanto maior for o círculo ou quadrado que representa um *cluster*, maior é a quantidade de documentos que esse *cluster* contém, sendo os documentos individuais representados por uma folha de papel. Os resultados da busca podem ainda ser exportados para ficheiros HTML, TXT ou XML.

3.3.5 Vivísimo e Clusty

O motor de busca `Vivísimo` foi desenvolvido na Universidade *Carnegie Mellon*, em 2000 [44]. Foi considerado o Estado da Arte e um dos principais serviços que efectua *clustering* sobre os *snippets* de páginas Web [10]. O seu sucesso deveu-se em parte à qualidade dos seus *clusters* e à satisfação dos seus utilizadores, tendo recebido o prémio do melhor *Meta-Searcher* nos anos de 2001 a 2003, prémio atribuído

[10] http://demo.carrot-search.com/carrot2-webapp/main

[11] http://www.grokker.com

anualmente pela revista *on-line* SearchEngineWatch[12]. Em 2005, Vivísimo ganhou o contrato para a implementação de buscas no FirstGov[13], o portal oficial do Governo dos Estados Unidos[14].

Sendo um motor comercial, o algoritmo de *clustering* implementado não é de código aberto, sendo por isso desconhecido. No entanto, sabe-se que é eficiente e é baseado num algoritmo heurístico para agrupar documentos textuais, e numa velha ideia de Inteligência Artificial: um bom *cluster* - ou grupo de documentos - é aquele que fornece uma boa e clara descrição do seu conteúdo. Assim, em vez de se formarem *clusters* e depois se tentar descrever o seu conteúdo, o algoritmo primeiramente forma os *clusters* com boas descrições, e só depois são adicionados os documentos, tal como acontece no *Lingo* [19].

Da análise dos seus *clusters*, é ainda possível dizer que são parcialmente sobrepostos, e que as suas hierarquias possuem dois níveis [10]. Não se sabe se o algoritmo faz uso de "conhecimento" externo, através de outros serviços, ou se os seus dados de entrada são apenas os documentos das buscas. Actualmente o Vivísimo original parece ter sido desactivado, para dar o lugar ao seu sucessor - o Clusty[15], o qual é baseado na mesma tecnologia. A empresa Vivísimo desenvolve e fornece serviços para buscas na Web e em ambientes corporativos.

3.3.6 Outros Motores de Busca

Além dos motores já citados, muitos outros merecem referência. Se alguns tentam inovar pela qualidade dos algoritmos usados, melhorando por exemplo a qualidade das descrições dos *clusters*, outros apostam nas funcionalidades ou na interface com o utilizador, tentando representar os *clusters* de forma gráfica. Exemplos destas interfaces podem ser vistas no KartOO[16], que mostra *clusters* sob a forma de nuvens, no AllPlus[17] que mostra *clusters* em grafos, ou no TouchGraph[18], que permite uma representação visual das ligações entre as várias páginas Web, sobre os resultados obtidos do Google.

A Tabela A.1 do Apêndice A faz uma comparação de alguns dos motores de busca actuais, segundo as seguintes características:

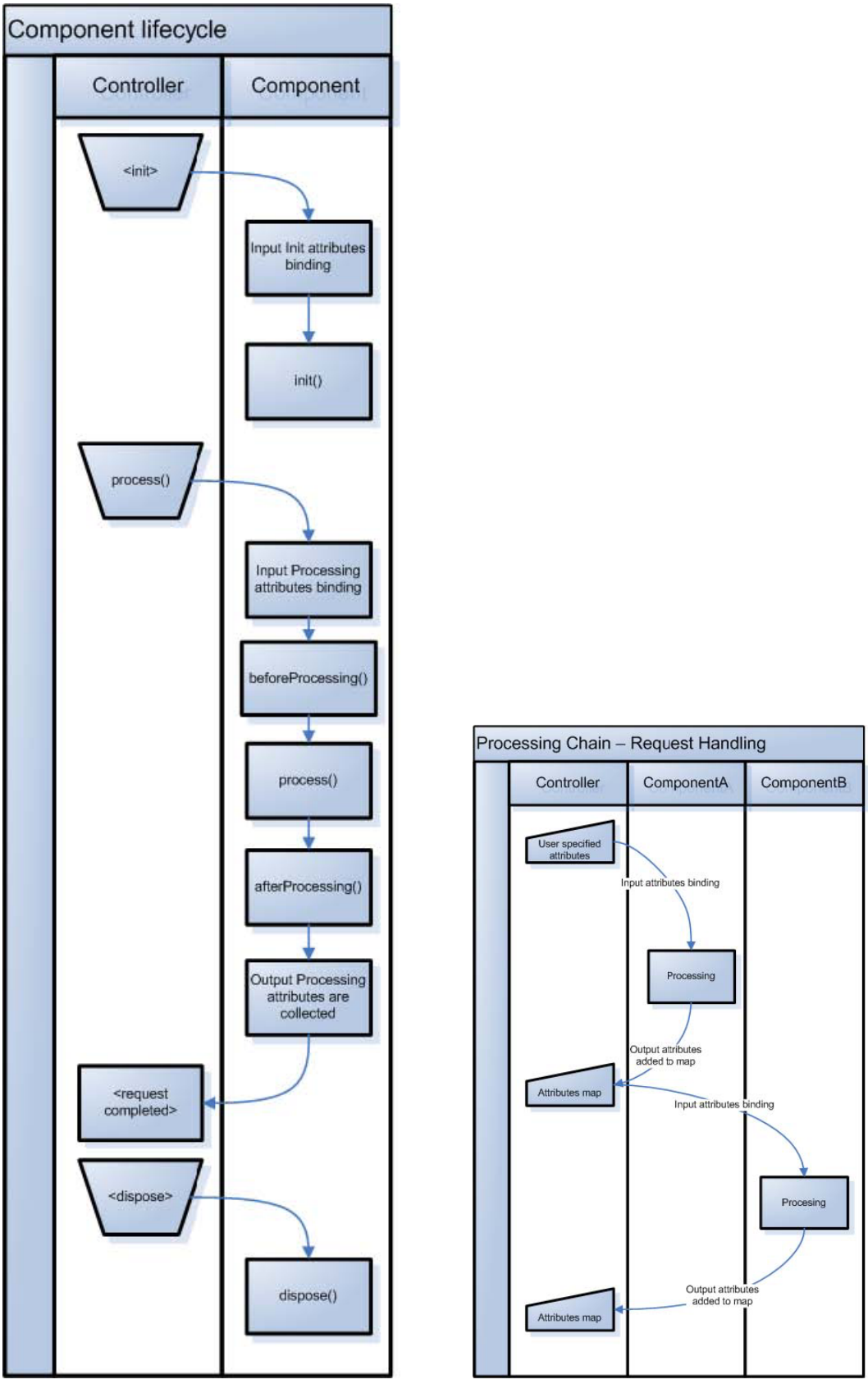
1. Se o motor é ou não *Meta-Searcher* e, em caso positivo, quais são os motores convencionais que são consultados, servindo de fonte;

2. Se os resultados das várias fontes são combinados ou mostrados de forma independente;

3. Se o motor efectua ou não *clustering* sobre os resultados e, em caso positivo, de que forma são apresentados os *clusters* (texto ou gráficos). Nalguns casos, é também indicado o algoritmo de *clustering* implementado e se os *clusters* são ou não hierárquicos;

4. Finalmente são indicados outros tipos de busca suportados, os quais podem ser temáticos, por fonte *Meta-Search*, por domínio do URL, ou por linguagem, país ou continente.

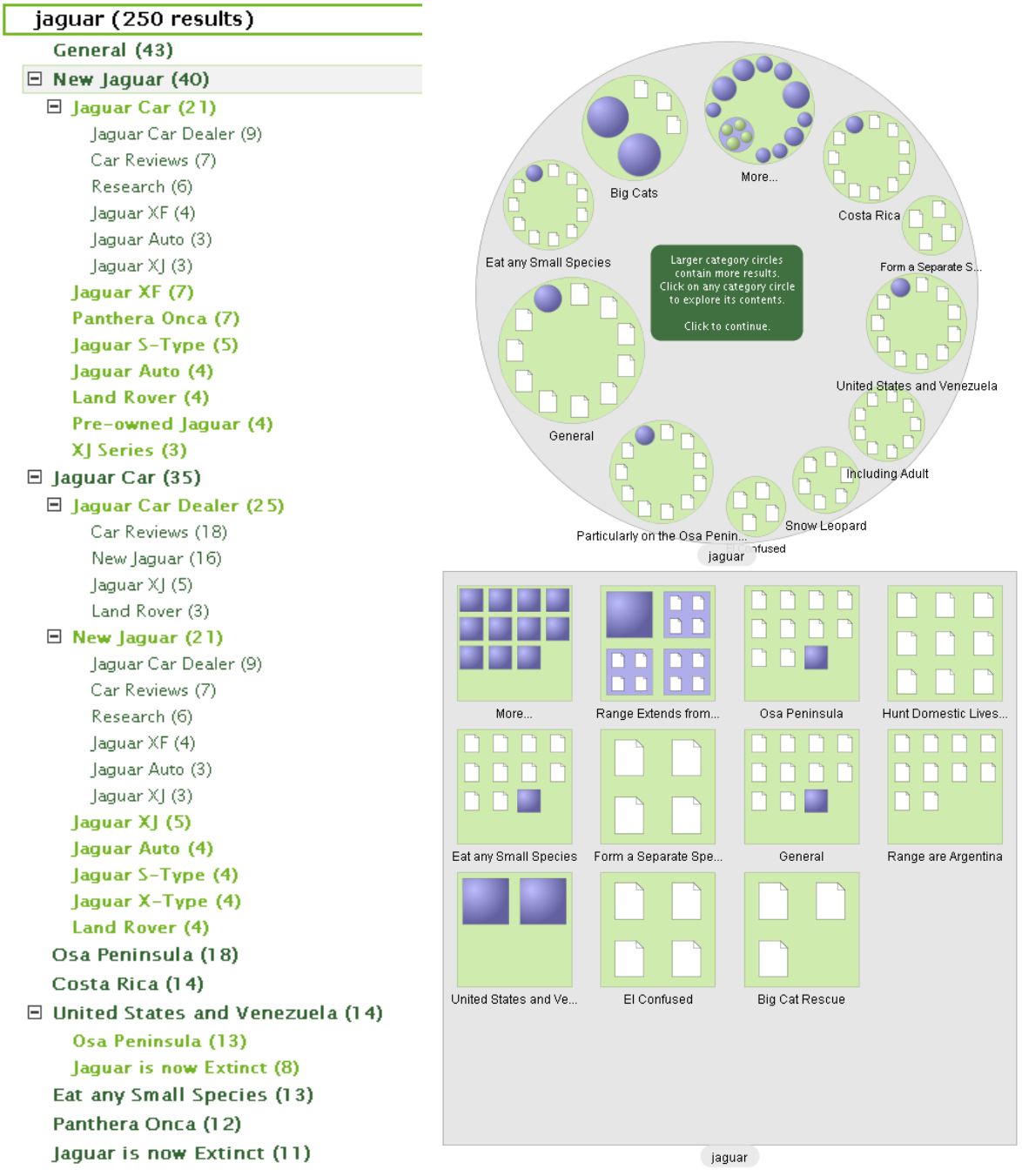
[12] http://www.searchenginewatch.com
[13] http://FirstGov.gov
[14] http://en.wikipedia.org/wiki/Vivisimo
[15] http://clusty.com
[16] http://www.kartoo.com
[17] http://www.allplus.com
[18] http://www.touchgraph.com



(a) Interacção entre controlador e componente

(b) Fluxo de processamento com mapa de atributos

Fig. 3.10: Carrot v3: Controlador e componentes [18].



(a) *clusters* como Árvore de Texto

(b) *clusters* em *"Map View"*

Fig. 3.11: `Grokker`: Visualização dos *clusters* hierárquicos.

Capítulo 4

Arquitectura da Solução

Após se analisar o caso da Web Portuguesa, verificou-se que esta ainda está atrasada no que diz respeito aos motores de busca, sendo mais frequente encontrar portais (*directórios*), que apenas oferecem listas de tópicos classificados manualmente. Os motores de busca Portugueses são poucos, e os motores *Meta-Search* são inexistentes. Com base nestes pressupostos, a solução implementada no âmbito desta Tese consistiu na alteração da *framework* `Carrot` de modo a funcionar no panorama Português. Desta forma, foi criado um *meta-searcher* Português, com a capacidade de *clustering* de resultados, devidamente parametrizado para lidar com as particularidades específicas da língua Portuguesa.

4.1 Alterações na *framework* Carrot

A solução desenvolvida inclui não só a criação de um *Meta-Searcher* Português, com capacidade de *clustering* de resultados, como também a implementação das modificações e parametrizações necessárias para tornar um tal sistema capaz de lidar com as especificações da nossa língua. Na construção do protótipo apresentado neste trabalho, foi usada a versão 3.0 da *framework* `Carrot`, por ser a versão mais actual disponível na altura [18]. Em seguida descrevem-se as alterações efectuadas à *framework* `Carrot`.

4.1.1 Componentes de Entrada

Nos componentes de entrada foram modificadas algumas fontes (motores de busca) já existentes (`GoogleAJAX`, `Yahoo!`, `MSN`, `Wikipedia`) no `Carrot` para procura apenas em páginas Portuguesas ou escritas em Português (neste caso incluindo o Português do Brasil). Foram também implementadas novas fontes correspondentes a motores de busca Portugueses, tais como o `Sapo`[1], `NetIndex`[2] e `Tumba!`[3]. São usadas APIs de pesquisa (*web-services*) dos motores. Por exemplo, no caso do `GoogleAJAX` é usada a API *Google AJAX*, e no caso do `Yahoo!` é usada a API *Yahoo Boss*. Na falta destas APIs, foram lidas e processadas as páginas HTML dos resultados, como é o caso das pesquisas `Google`, `Sapo`, `NetIndex` e `Tumba!`, as quais também constituem as quatro fontes da Meta-Pesquisa "Web". Cada fonte tem a sua própria configuração, visto que depende

[1] http://www.sapo.pt
[2] http://www.netindex.pt
[3] http://www.tumba.pt

das capacidades ou limitações impostas pelos motores de busca ou APIs respectivos. A tabela 4.1 enumera algumas destas configurações.

Fonte	API	Pesquisas em Simultâneo	Resultados por Pesquisa	Máximo de Resultados
Google / IST	HTML	10	100	1000
GoogleAJAX	AJAX	10	8	64
Sapo	HTML	4	20	1000
SapoJSON	JSON	10	50	1000
NetIndex	HTML	4	10	1000
Tumba!	HTML	4	50	1000
MSN	Live	10	50	1000
Yahoo / Wikipedia	BOSS	5	50	1000
Web	Várias	22	180	4000

Tabela 4.1: Configuração das diversas fontes de Pesquisa

Por exemplo, a fonte `GoogleAJAX` permite efectuar em simultâneo 10 pesquisas, em que cada uma retorna no máximo 8 resultados, até um limite máximo de 64 resultados. Como outro exemplo, no caso da fonte `Sapo` são necessários 50 pedidos de 20 resultados cada, para se obter um total de 1000 resultados. A Figura C.1 do Apêndice C mostra a hierarquia de classes Java da implementação das diversas fontes usadas no protótipo.

4.1.2 Pré-Processamento

No pré-processamento foram usadas listas com palavras frequentes (*stopwords*) Portuguesas (do Português de Portugal e do Brasil). Foram editadas e combinadas várias listas, incluindo algumas da já mencionada colecção *CHAVE*, que se encontram disponíveis na Web[4]. A actual lista usada no protótipo tem cerca de 600 palavras, as quais se enumeram no Apêndice B. Foram aplicados algoritmos de radicalização (*stemming*) preparados para a língua Portuguesa, sendo adaptados ao protótipo os algoritmos radicalizadores *Portuguese Porter Stemmer* e *RSLP* (Removedor de Sufixos da Língua Portuguesa) [25]. Informação adicional sobre os dois radicalizadores de Português pode ser encontrada na Web[5], bem como a API *PTStemmer*[6] usada no protótipo.

4.1.3 Processamento

Na etapa de processamento foram usados os algoritmos *STC* e *Lingo* existentes na versão 3.0 da *framework* `Carrot`. O algoritmo *TRC* [19] foi adaptado da versão 2.x para a versão 3.0 da *framework*. Os algoritmos encontram-se descritos no Capítulo 3.2. Adicionalmente, estes algoritmos foram devidamente integrados com alguns dos radicalizadores de palavras disponíveis para a língua Portuguesa (*Snowball PT*, *Porter PT*

[4] http://linguateca.di.uminho.pt/Paulo/stopwords
[5] http://informationr.net/ir/12-3/paper315.html
[6] http://code.google.com/p/ptstemmer

e *RSLP*) e também com as listas de palavras frequentes. Ainda neste passo, e com base em resultados empíricos, foram feitos alguns ajustes aos vários parâmetros dos algoritmos usados, de modo a obter os melhores valores para a língua Portuguesa. No Capítulo 5.5, na página 45, são mostrados os resultados da avaliação experimental.

4.1.4 Visualização e Interface

Nos componentes de Visualização e Interface, foi traduzida a interface do `Carrot` para Português e foram adicionadas novas opções, tais como a possibilidade de escolher o radicalizador de palavras e de usar ou não a lista de palavras frequentes. A Figura 4.1 ilustra a interface do protótipo.

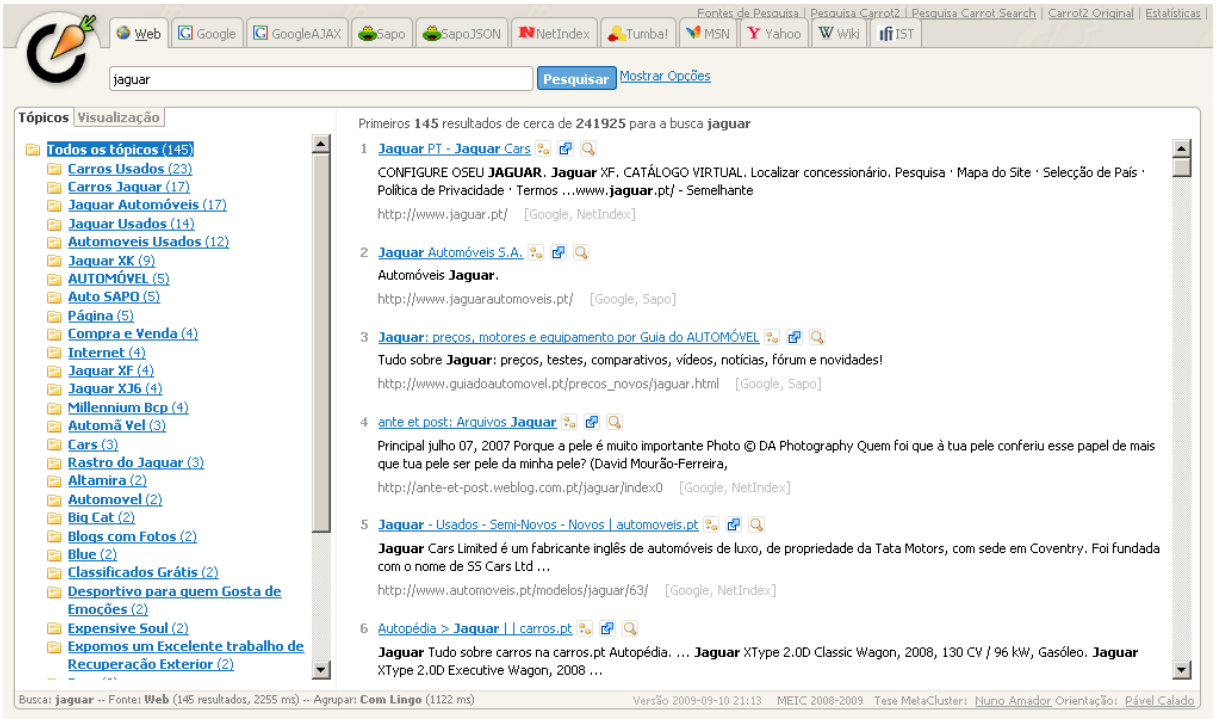


Fig. 4.1: Interface do Protótipo.

Importa notar que a *framework* `Carrot` não tem implementado de origem um algoritmo de ordenação e fusão (*ranking* e *merging*) dos resultados, limitando-se a apresentá-los tal como vêm dos motores de busca individuais (o `Carrot` usa o Meta-Motor `eTools`, o qual efectua a Meta-Pesquisa, e já fornece a lista de resultados tratada). No entanto, com a implementação da Meta-Pesquisa "Web", houve a necessidade de combinar e ordenar os resultados retornados pelas diversas fontes da Meta-Pesquisa. De forma a resolver este problema, teve de ser implementado um algoritmo que fosse capaz de desempenhar esta tarefa, e que em seguida se descreve.

4.2 Ordenação e fusão de Resultados

Os sistemas de recuperação de informação geralmente apresentam os resultados ordenados pela sua relevância estimada, calculada através de métricas que indicam o grau de semelhança dos resultados relativamente

aos termos usados para efectuar a pesquisa. Esta ordem de relevância é calculada internamente em cada motor de busca individual, muitas vezes recorrendo à combinação [37], ou através de algoritmos mais complexos, como é o caso do *PageRank* do `Google` [31]. No caso concreto da fonte "Web", houve necessidade de ter um algoritmo que fosse capaz de ordenar os resultados e juntar resultados duplicados num só, quando são provenientes de várias fontes. Desta forma, para a pesquisa "Web" pretendia-se um algoritmo que fosse capaz de:

1. Fusão (*merging*) - fundir e agrupar num só resultado todos com o mesmo URL, quando estes são obtidos de várias fontes. É mantida uma lista de fontes por resultado, com todas as fontes que forneceram esse resultado. Ainda durante a fusão, é efectuada a comparação dos títulos e das descrições obtidos das várias fontes para o mesmo resultado, na tentativa de escolher o que tiver um "melhor" título/descrição. No protótipo desenvolvido, são simplesmente escolhidas as maiores frases, respectivamente para o título e para a descrição, que foram obtidos de todas as fontes;

2. Ordenação (*ranking*) - foi implementado um algoritmo simples que ordena os resultados de acordo com os valores calculados pela equação (4.1)

$$s(r) = \sum_{f=1}^{F_r} f * N * 100 - \sum_{f=1}^{F} P_{r,f} * W_f \tag{4.1a}$$

$$= \frac{F_r(F_r+1)}{2} * N * 100 - \sum_{f=1}^{F} P_{r,f} * W_f \tag{4.1b}$$

$$= F_r(F_r+1) * N * 50 - \sum_{f=1}^{F} P_{r,f} * W_f \tag{4.1c}$$

   onde $s(r)$ é a pontuação de um resultado na lista de resultados "Web"; F_r é o número de fontes que forneceram o resultado; N é a soma total dos resultados obtidos de todas as fontes individuais (sem *merging*); F é o número de fontes Meta-Pesquisa "Web"; $P_{r,f}$ é a posição do resultado r na lista de resultados da fonte f; finalmente, W_f é o peso da fonte f, em que menores valores terão mais peso e vice-versa.
   Verificam-se ainda as condições:

   $1 \leq P_{r,f} \leq N = \sum_{f=1}^{F} max(P_{r,f})$
   $1 \leq F_r \leq F = 4$, que correspondem às fontes `Google`, `Sapo`, `NetIndex` e `Tumba!`.
   $0 \leq W_f \leq 100$, neste momento as 4 fontes têm o mesmo peso, que é 100.

Os resultados na lista da pesquisa são ordenados por ordem decrescente de $s(r)$, ficando o maior valor no topo da lista. A ideia é ter os resultados ordenados decrescentemente pelo número de fontes dos quais foram obtidos e, em caso de empate, é tida em conta a posição nas fontes individuais, mas permitindo ter um peso atribuído a cada fonte.

Capítulo 5

Avaliação Experimental e Resultados

Neste capítulo, mostram-se os resultados experimentais obtidos com o protótipo desenvolvido.

5.1 Método de Testes

Para a avaliação experimental, a interface foi alterada de modo a ocultar as várias opções e parâmetros dos algoritmos, sob a forma de 20 tipos diferentes de botões "Pesquisar", de forma a não influenciar os utilizadores quanto aos parâmetros usados. Adicionalmente, foram incluídas duas ligações, uma para a avaliação da lista de resultados, e outra para a avaliação dos tópicos formados. Para estas ligações, são geradas duas páginas dinâmicas sob a forma de inquéritos. As Figuras F.1 e F.2, no Apêndice F, ilustram as duas páginas dinâmicas geradas para a busca “bola”, usando o radicalizador *Porter PT* e o algoritmo *Lingo*. A Figura 5.1 ilustra a interface de avaliação.

Seguidamente, foram levados a cabo testes com utilizadores em que foi pedido a cada um que fizesse uma qualquer pesquisa do seu interesse e, em seguida, avaliasse: (1) cada resultado retornado como relevante ou não; (2) cada tópico (*cluster*) criado como relevante ou não, e como tendo uma descrição explicativa ou não do seu conteúdo.

Os resultados dos inquéritos são submetidos pelos utilizadores, e ficam guardados em ficheiros de texto, onde posteriormente são tratados recorrendo a *scripts* usando comandos em Linux *BASH shell* e a ferramenta `GNUPlot`. Desta forma, os *scripts* implementados permitem regenerar automaticamente os gráficos sempre que houver mais dados dos inquéritos.

Os resultados retornados pelos motores de busca foram avaliados, calculando a sua precisão, como definida na equação (5.1)

$$P = \frac{|A \cap B|}{|B|} \tag{5.1}$$

em que A é o conjunto de resultados relevantes e B é o conjunto de resultados retornados. Foi medida a precisão média para os primeiros 5, 10 e 20 resultados no topo da lista. No caso dos tópicos, foi medida a precisão média para os 5, 10 e 20 maiores tópicos (com maior número de documentos), excluindo o

Fig. 5.1: Interface do protótipo modificada para a avaliação.

tópico “Outros Tópicos”. Pelo menos 10 utilizadores preencheram e submeteram inquéritos, para os quais se obtiveram as estatísticas presentes na Tabela 5.1.

	Avaliação de Resultados	Avaliação de tópicos (*clusters*)	Totais
Nº de Inquéritos	37	53	90
Nº de *queries* distintas	33	35	37

Tabela 5.1: Estatísticas dos inquéritos

Nas secções seguintes mostram-se os resultados obtidos nos inquéritos. As barras verticais apresentam os valores médios de precisão. No topo de cada gráfico, estão o número de amostras consideradas para o cálculo do valor. Adicionalmente, os valores marcados com “ * ” referem-se aos valores por omissão dos algoritmos do `Carrot`.

5.2 Precisão dos Resultados

A Figura 5.2 ilustra a precisão média obtida a partir dos dados recolhidos dos inquéritos de avaliação de resultados.

Uma breve análise do gráfico revela que, como esperado, a precisão decresce com o aumento do número de resultados avaliados. No entanto, esta mantém-se acima dos 60% para os 10 primeiros resultados, que são normalmente os únicos vistos pelos utilizadores.

Convém notar que apenas foi permitido aos utilizadores usarem a fonte relativa à meta-pesquisa "Web",

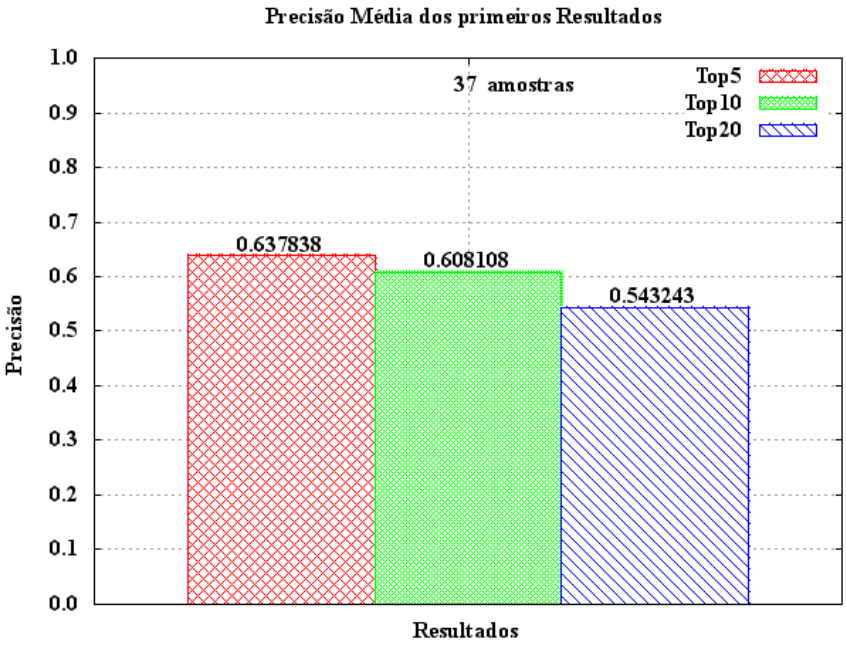


Fig. 5.2: Precisão média dos primeiros 5, 10 e 20 resultados.

pois esta efectua a fusão e a ordenação de resultados, usando a equação 4.1 descrita na secção 4.2. Desta forma, evitou-se que os utilizadores avaliassem o mesmo resultado mais do que uma vez, e ao mesmo tempo avaliassem a lista de resultados ordenada pela já mencionada equação 4.1.

5.3 Precisão dos Algoritmos nos Tópicos

A Figura 5.3 ilustra a precisão obtida em cada um dos algoritmos *TRC*, *STC* e *Lingo*. Os valores foram obtidos com a utilização do algoritmo de radicalização *RSLP*.

Neste caso o algoritmo *STC* revelou uma precisão superior ao *Lingo*. Isto poderá dever-se ao facto do *STC* retornar maior número de tópicos com apenas um termo, os quais se revelaram para os utilizadores mais precisos que os tópicos de vários termos (frases) formados pelo *Lingo*. O algoritmo *TRC* recebeu a pior precisão devido ao facto de atribuir descrições aos tópicos que frequentemente nada têm a ver com os documentos que os compõem.

5.4 Precisão dos Radicalizadores

A Figura 5.4 representa a precisão obtida para os radicalizadores *Snowball PT*, *Porter PT* e *RSLP* de Orengo. Nenhum significa que nenhum radicalizador foi usado. Os valores foram obtidos com a utilização do algoritmo de *clustering Lingo*.

O gráfico revela que não usar radicalizador permite obter maior precisão para os 5 e 10 maiores tópicos. A utilização do radicalizador torna as palavras mais ambíguas, o que leva a uma esperada diminuição da precisão. Estes resultados indicam que, para a língua portuguesa, é vantajoso não usar radicalizadores, se

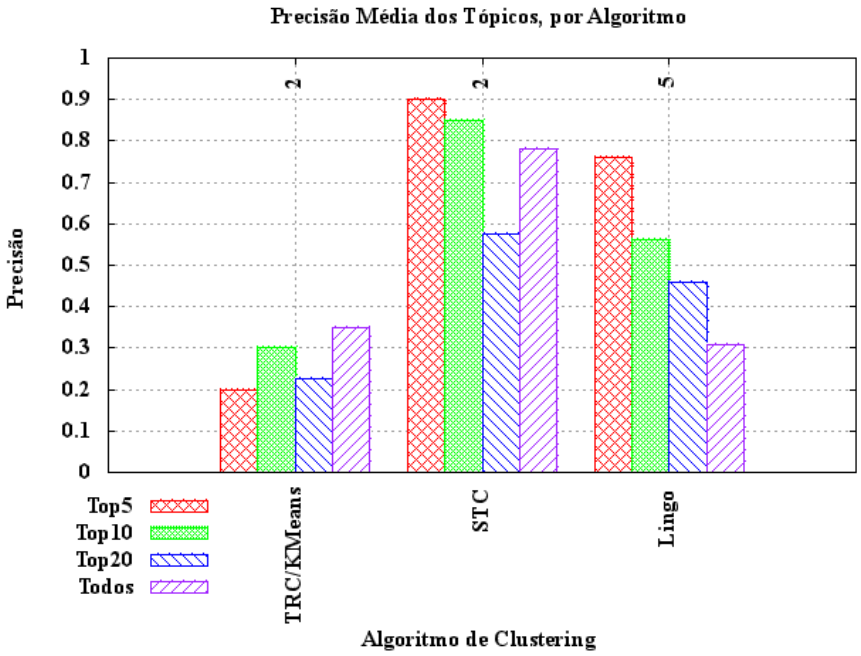


Fig. 5.3: Precisão média dos Tópicos, por algoritmo de *clustering*.

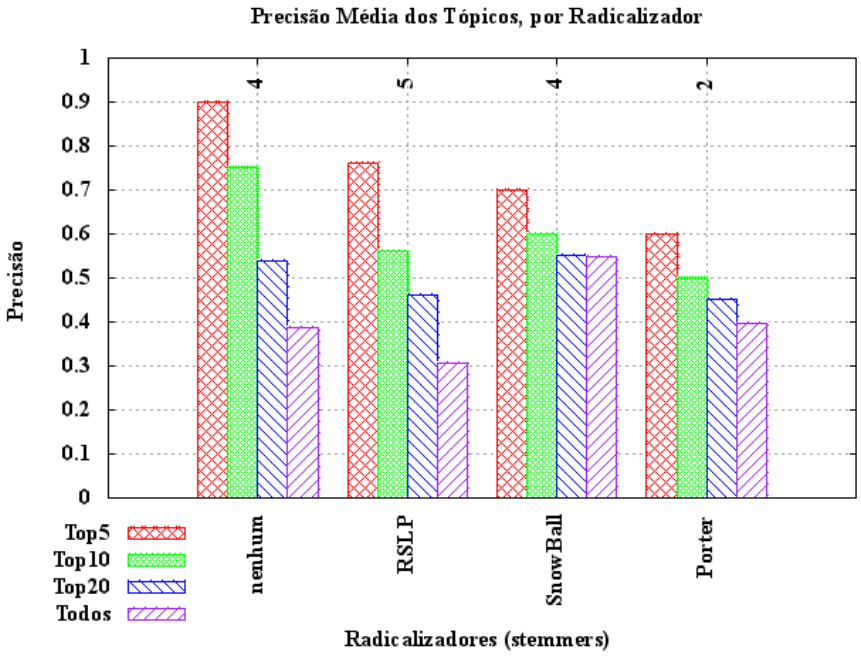


Fig. 5.4: Precisão média dos Tópicos, por radicalizador.

pretendemos obter alta precisão nos primeiros tópicos.

5.5 Precisão de alguns parâmetros dos algoritmos

Uma vez que a maioria dos algoritmos apresentados foi desenvolvido e testado na língua inglesa, é interessante perceber se estes podem ser configurados para funcionar na língua portuguesa. Nesta secção mostram-se os resultados obtidos para a precisão dos algoritmos, quando se variam os valores de alguns dos seus parâmetros.

5.5.1 Algoritmo *TRC/K-Means - Tolerance Rough Clustering*

Neste algoritmo, foi testado o parâmetro *Membership Threshold*, percentagem que define o "quanto" um documento deve ser relacionado com um *cluster* de modo a passar a pertencer a esse *cluster*. Os valores experimentados foram 0.2, 0.3 e 0.4, ilustrados na Figura 5.5.

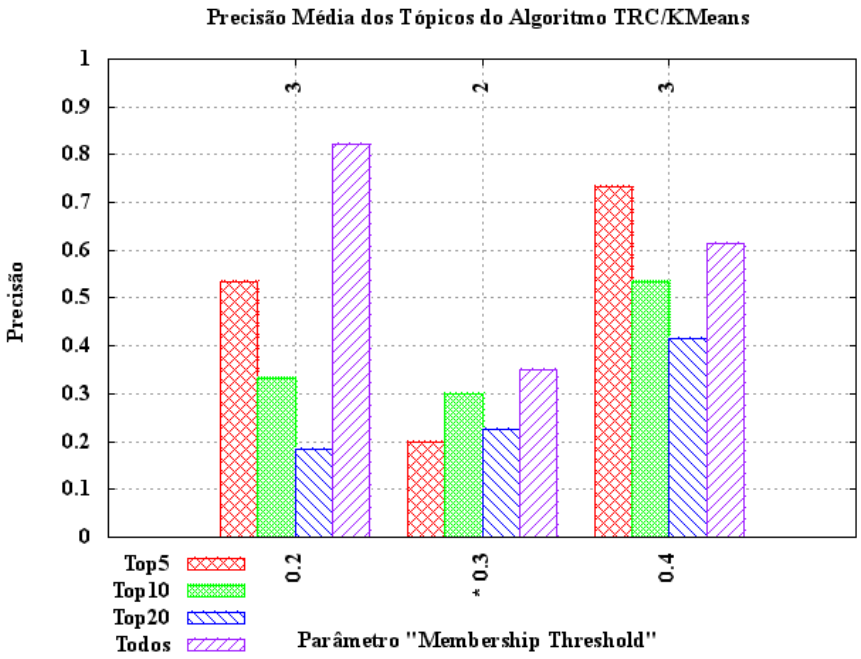


Fig. 5.5: Precisão média dos Tópicos do algoritmo *TRC* ("*membership threshold*").

O valor por omissão do `Carrot` (0.3), foi o que obteve a pior precisão em relação aos outros dois valores experimentados. Tal confirma que os parâmetros adequados para a língua inglesa podem não ser os melhores para o português. Os melhores resultados no topo da lista de páginas retornadas foram obtidos com o valor 0.4, enquanto o valor 0.2 teve uma melhor precisão no conjunto total de resultados.

5.5.2 Algoritmo *STC - Suffix Tree Clustering*

Para o algoritmo *STC* foram testados os parâmetros *Document Count Boost*, *Optimal Phrase Length Dev*, *Single Term Boost* e *Merge Threshold*. O parâmetro *Document Count Boost* aumenta a pontuação de um *cluster* base com o número de documentos que pertence a esse *cluster*. Os valores experimentados foram 1.0, 1.5, 2.0 e estão ilustrados na Figura 5.6.

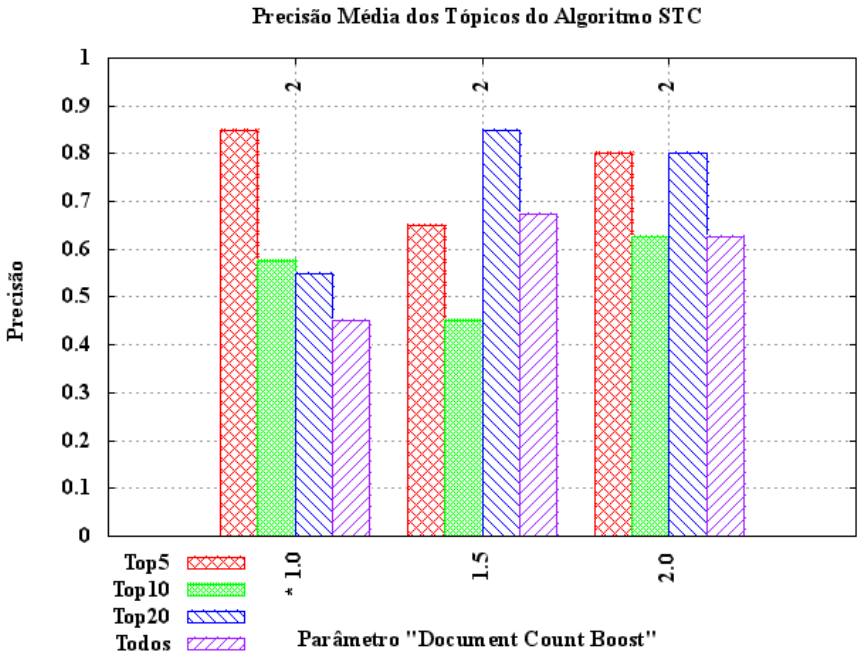


Fig. 5.6: Precisão média dos Tópicos do *STC* (“*Document Count Boost*”).

O parâmetro *Optimal Phrase Length Dev* contribui com parte da pontuação de um *cluster* base, fazendo esta parte depender da variação do tamanho da frase da descrição desse *cluster*. Os valores experimentados foram 1.0, 1.5, 2.0, e estão ilustrados na Figura 5.7.

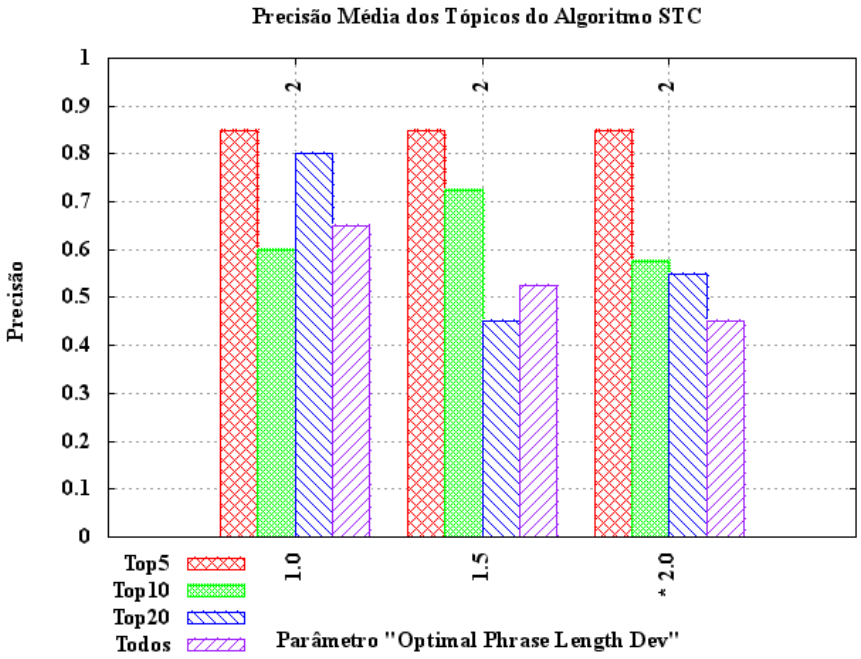


Fig. 5.7: Precisão média dos Tópicos do *STC* (“*Optimal Phrase Length Dev*”).

Foram obtidos valores de precisão muito semelhantes para os três valores experimentados, embora o maior valor (2.0) ser mais estável à medida que aumenta o número de tópicos. Dada a pouca quantidade

de amostras (apenas uma) em dois dos valores não é possível tirar conclusões em relação à precisão do parâmetro. No entanto, os valores obtidos parecem ser muito semelhantes, indicando que o parâmetro não tem uma importância significativa. O parâmetro *Single Term Boost* também influencia a pontuação de um *cluster* base. Se for maior que zero, o valor é atribuído à pontuação dos *clusters* base que têm descrições de apenas um termo, independentemente da pontuação já calculada para esses *clusters*. Desta forma é possível fazer com que *clusters* base com apenas um termo na sua descrição tenham maior ou menor "peso" em relação a *clusters* com vários termos. Os valores experimentados foram 0.5, 1.0, 2.0 e estão ilustrados na Figura 5.8.

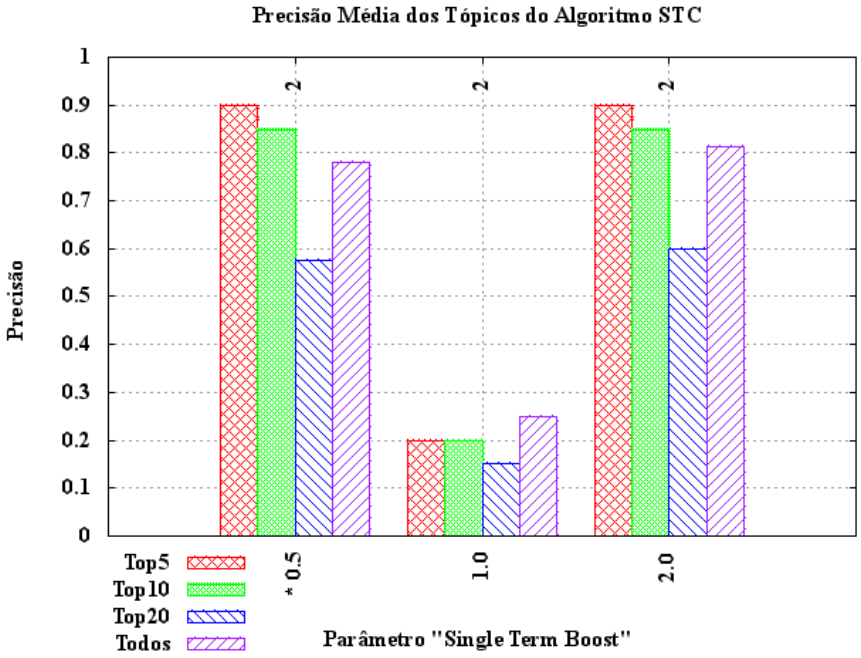


Fig. 5.8: Precisão média dos Tópicos do *STC* ("*Single Term Boost*").

Os valores 0.5 e 2.0 mostraram valores de precisão iguais, e muito superiores aos obtidos com o valor 1.0 do parâmetro. O parâmetro *Merge Threshold* corresponde à percentagem de documentos comuns entre dois *clusters* para que estes sejam fundidos num só *cluster*. Baixos valores resultam em fusões mais agressivas, que podem originar mais documentos irrelevantes nos *clusters*. Valores altos minimizam a fusão de *clusters*, que podem resultar em *clusters* muito semelhantes ou mesmo duplicados. Os valores experimentados foram 0.5, 0.6, 0.8 e estão ilustrados na Figura 5.9.

Os três valores de teste revelaram uma precisão média semelhante, de onde se conclui que o valor do parâmetro pouco influência a precisão.

5.5.3 Algoritmo *Lingo*

Para o algoritmo *Lingo* foram testados os parâmetros *Cluster Merging Threshold* e *Title Words Boost*. O parâmetro *Cluster Merging Threshold* do algoritmo *Lingo* tem um comportamento semelhante ao parâmetro *Merging Threshold* do algoritmo *STC*. Os valores experimentados foram 0.1, 0.7 e 0.9 e estão ilustrados na Figura 5.10.

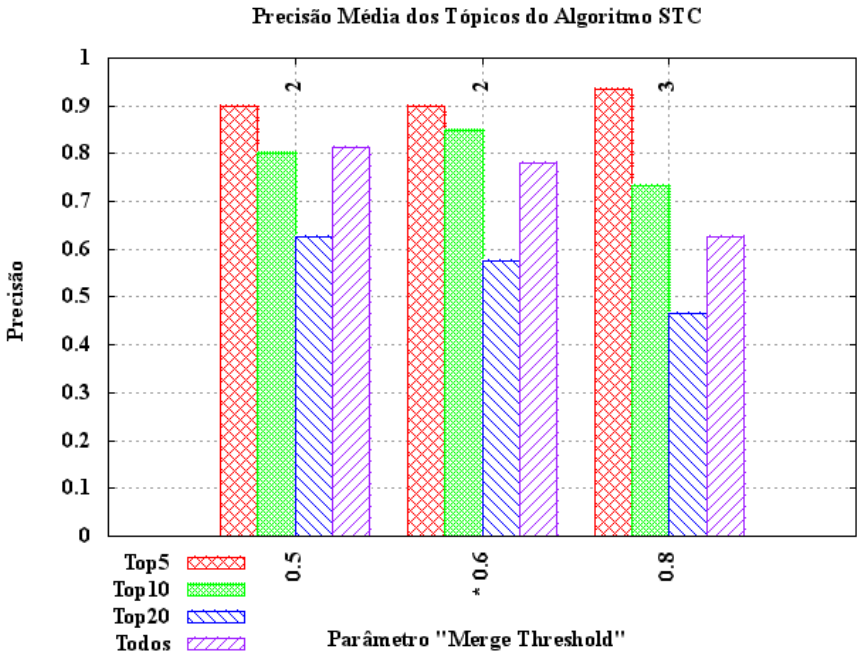


Fig. 5.9: Precisão média dos Tópicos do *STC* ("*Merge Threshold*").

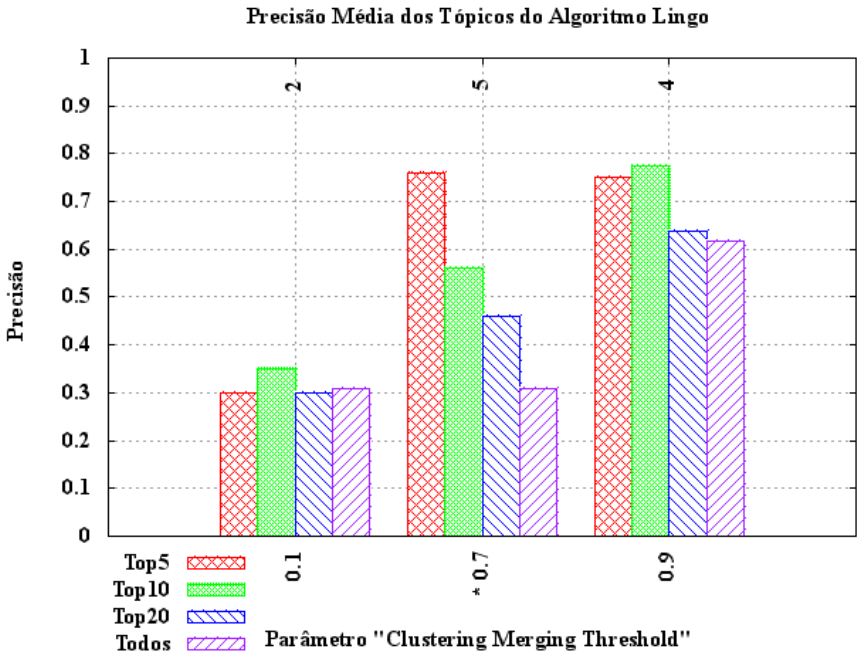


Fig. 5.10: Precisão média dos Tópicos do *Lingo* ("*Clustering Merging Threshold*").

Para diferentes valores, o parâmetro *Clustering Merging Threshold* do *Lingo* não revelou a mesma estabilidade de precisão que foi obtida com o parâmetro *Merging Threshold* do *STC*. Maiores valores do parâmetro permitem de facto obter maiores precisões, mas minimizam a fusão de tópicos, resultando em tópicos muito semelhantes ou mesmo duplicados. O parâmetro *Title Words Boost* permite definir o peso relativo dos termos que aparecem no título em relação aos mesmos termos que aparecem na descrição de um resultado.

Os valores experimentados foram 0.5, 1.0 e 2.0 e estão ilustrados na Figura 5.11.

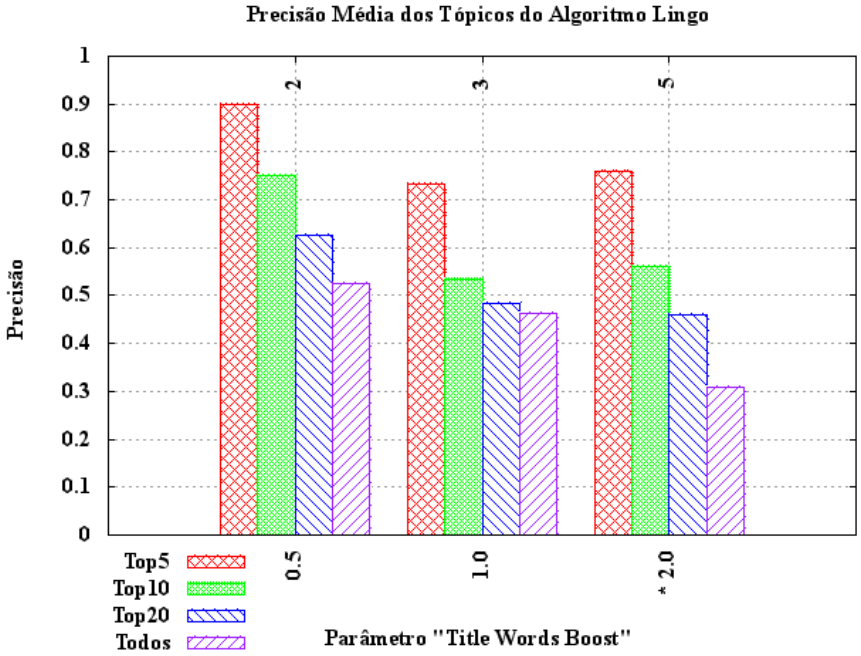


Fig. 5.11: Precisão média dos Tópicos do *Lingo* (“*Title Words Boost*”).

A precisão obtida foi muito semelhante para os valores experimentados. No entanto, atribuir menor peso (0.5) aos termos dos títulos das páginas revelou uma precisão superior. Tal será devido ao facto de muitas páginas Web não terem simplesmente título, ou este ter pouco a ver com o conteúdo da página.

5.6 Qualidade das descrições dos Tópicos

Nesta secção mostra-se a percentagem de boas descrições atribuídas pelos utilizadores, em função do algoritmo (Figura 5.12) e em função do radicalizador (Figura 5.14). Esta avaliação é importante porque, um bom tópico (com documentos relevantes), poderá ser descartado e ignorado pelo utilizador caso a sua descrição seja de fraca qualidade.

Ambos os algoritmos *STC* e *Lingo* obtiveram melhor percentagem de boas descrições por parte dos utilizadores, relativamente ao algoritmo *TRC*. Para melhor ilustrar o desempenho dos algoritmos, a Figura 5.13 mostra os tópicos formados a partir de 121 resultados, obtidos com a busca “massa”. No exemplo dado, o radicalizador usado foi o *Porter PT*.

Claramente, existe uma superioridade na qualidade das descrições produzidas pelos algoritmos *STC* e *Lingo*, quando comparadas com o algoritmo *TRC*.

A Figura 5.14 mostra a precisão obtida para os diferentes radicalizadores, usando apenas o algoritmo *Lingo*. É interessante observar que os radicalizadores *Porter PT* e *Snowball PT* obtêm os melhores resultados. Aparentemente a redução a radicais efectuada pelo *RSLP* prejudica a qualidade das descrições. Do mesmo modo, quando não é usado radicalizador, mais *clusters* são criados, com menos documentos, o que aumenta a sua precisão, mas diminui a qualidade das descrições.

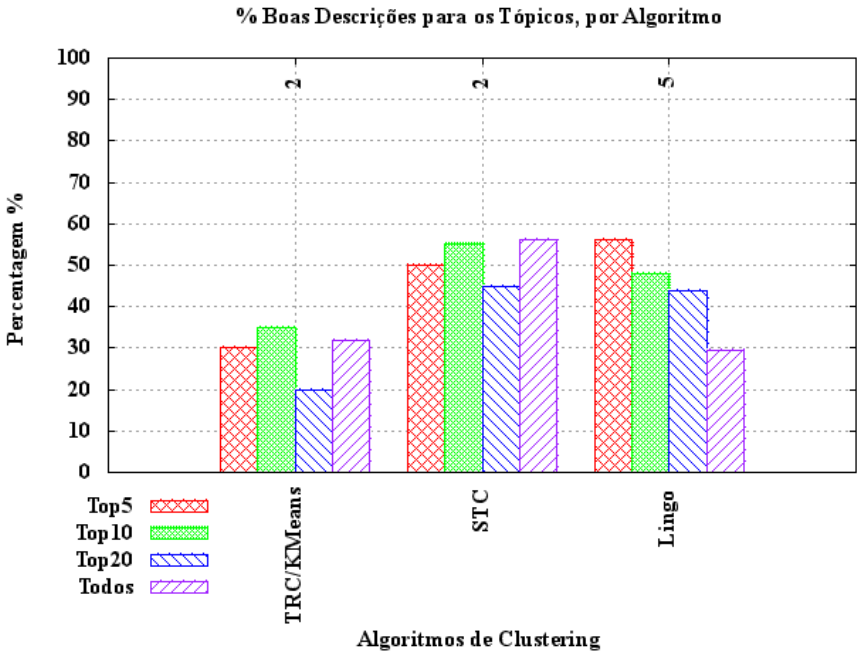


Fig. 5.12: Percentagem de boas descrições dos Tópicos, por algoritmo de *Clustering*.

A Figura 5.15 mostra a influência dos diferentes radicalizadores nas descrições dos tópicos, para os mesmos 160 resultados da busca "bola". O algoritmo usado foi o *Lingo*. Podemos observar que a utilização de um radicalizador leva à redução do número de tópicos, uma vez que, quando consideramos só a raiz das palavras, há uma maior tendência para alguns *clusters* se unirem num só. Por exemplo, dois temas como "jogadores da bola" e "jogo da bola" que, sem radicalização seriam considerados distintos, serão considerados o mesmo se for considerado apenas o radical das palavras. Assim, o radicalizador "nenhum" formou tópicos mais específicos, enquanto que o radicalizador *Snowball* formou tópicos mais abrangentes.

5.7 Eficiência dos Algoritmos

Para a avaliação da eficiência dos três algoritmos de *clustering TRC*, *STC* e *Lingo*, foi escolhida para o teste apenas uma fonte correspondendo a um motor de busca convencional, tendo neste caso sido escolhido o `Sapo`. Em seguida foram efectuadas cinco buscas com termos diferentes, para cada número de resultados pretendidos e suportados como opção na interface, ou seja, para 50, 100, 150, 200, 400 e 1000 resultados. Adicionalmente, para cada busca e número de resultados, escolheu-se um dos três algoritmos de *clustering* e mediu-se o tempo da duração de execução do algoritmo, excluindo portanto o tempo de pesquisa e *download* dos resultados da Web. Durante as medições, não foram usados algoritmos de radicalização. A cache de resultados implementada no `Carrot` foi igualmente desactivada, de forma a não influenciar os resultados.

Os valores foram obtidos em ambiente fechado, com apenas um utilizador/avaliador. Foi usado um computador pessoal com um processador Intel Core2 Quad Q9450 a 2.66 GHz (core Yorkfield, 45nm, stepping 7, revision C1) com 2 GigaBytes de RAM DDR3 a 1333 MHz, a correr em Linux uBuntu 8.04, numa máquina virtual VMWare 6.5 configurada com 1 GigaByte de RAM e para usar apenas um núcleo. A máquina virtual

Fig. 5.13: Tópicos formados pelos algoritmos *TRC*, *STC* e *Lingo*, para a busca “massa”.

Java usada é a J2RE 1.6.0 update 12.

A Figura 5.16 mostra o tempo médio (em mili-segundos) obtido para os cinco termos de busca quando se varia o algoritmo e o número de resultados.

Como se pode ver na Figura 5.16, o algoritmo *STC* demonstra, efectivamente, um comportamento linear com o aumento do número de resultados.

Para o algoritmo *Lingo*, foram usadas as bibliotecas nativas optimizadas `BLAS` (*Basic Linear Algebra Subprograms*) e `LAPACK` (*Linear Algebra Package*) para cálculo vectorial e matricial. As rotinas usadas nestas bibliotecas existem há cerca de 30 anos, tendo sido portadas para várias plataformas e linguagens, incluindo o Java, e são disponibilizadas juntamente com a *framework* `Carrot`. Segundo os autores do `Carrot`, o uso destas bibliotecas permite um aumento de até 400% de eficiência em relação ao cálculo matricial realizado inteiramente em Java. O uso destas bibliotecas para a decomposição *SVD* permite ao *Lingo* obter tempos de execução lineares e semelhantes aos obtidos pelo *STC*.

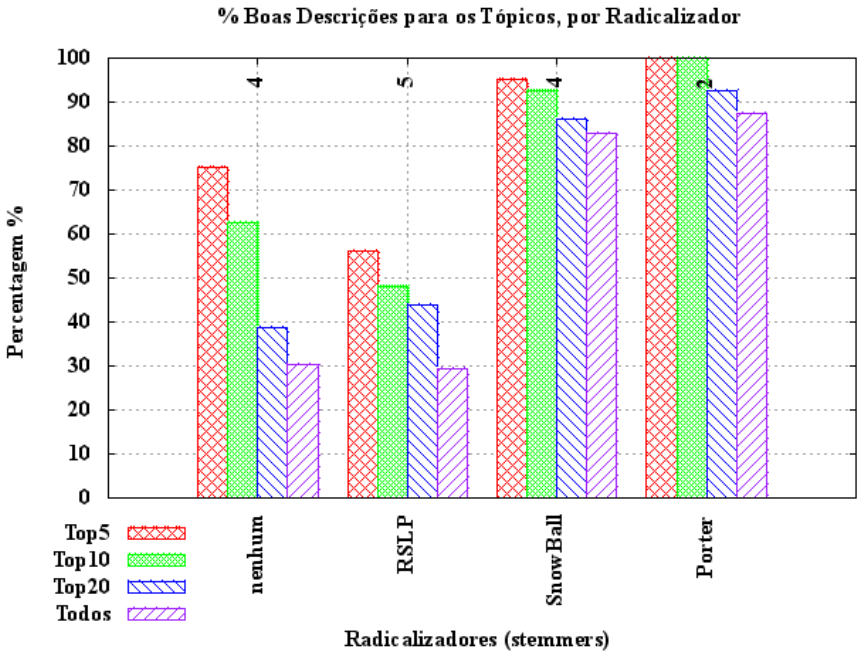


Fig. 5.14: Percentagem de boas descrições dos Tópicos, por Radicalizador.

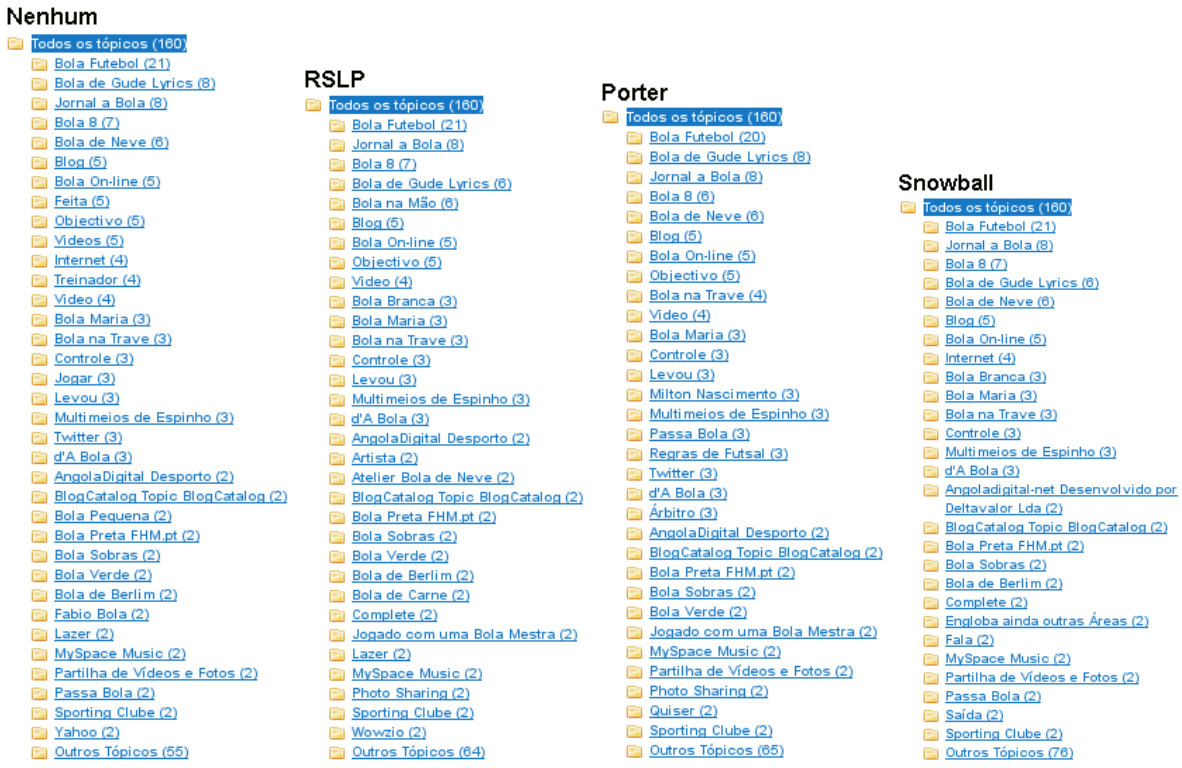


Fig. 5.15: Efeito do radicalizador nos Tópicos: nenhum, *RSLP*, *Snowball PT* e *Porter PT*.

O algoritmo *TRC*, devido ao seu comportamento quadrático, demonstra que apenas poderá ser usado para um baixo número de resultados de entrada. No *TRC*, os valores de 60000ms obtidos para 1000 resultados foram tão elevados que, quando comparados com os 1870ms do *STC* e 2760ms do *Lingo* obtidos para o mesmo número de resultados, afectaria a escala do gráfico da Figura 5.16, optando-se assim por não incluir

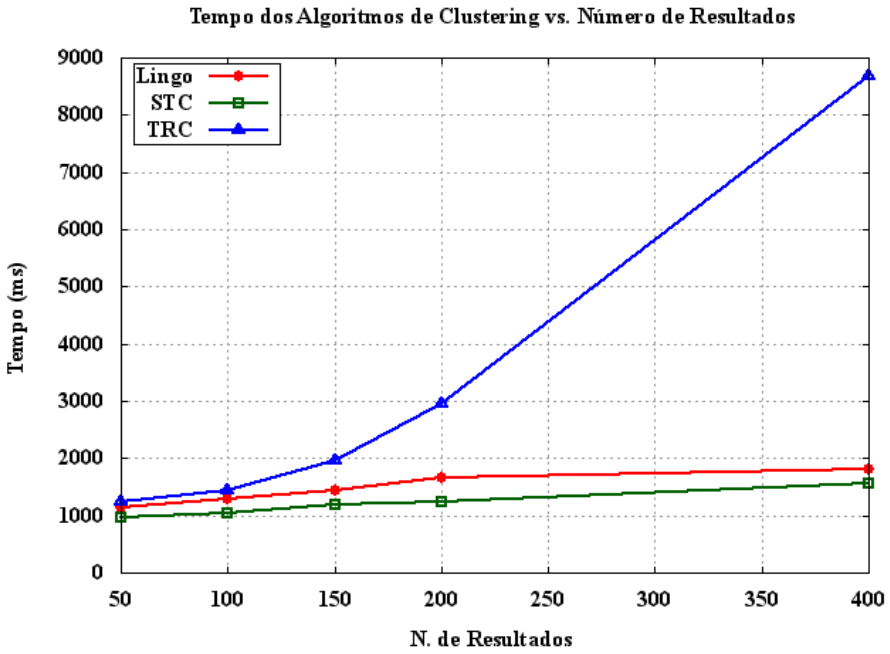


Fig. 5.16: Tempos de Clustering em função do Número de Resultados.

estes valores.

5.8 Avaliação da Meta-Pesquisa

O presente trabalho não poderia ser concluído sem que, de alguma forma, fossem efectuadas experiências e fosse incluída uma avaliação que permitisse demonstrar a eficácia, as vantagens e as desvantagens de se usar Meta-Pesquisa. É isso que se trata neste capítulo.

5.8.1 Sobreposição do Conjunto dos Resultados

Para a avaliação do conjunto de resultados da Meta-Pesquisa, foram efectuadas buscas para alguns termos, usando exclusivamente à meta-pesquisa "Web", e foram guardados os dados recorrendo novamente aos inquéritos de avaliação dos resultados. O objectivo consistiu em obter os primeiros 100 resultados de cada um dos 3 motores de busca convencionais: `Google`, `Sapo` e `NetIndex`, e avaliar até que ponto cada resultado é único. A fonte `Tumba!` ficou de fora porque, na altura da avaliação, se encontrava indisponível.

Um resultado é único quando o seu URL do documento é único no conjunto de resultados. De forma a simplificar a avaliação, parte-se do seguinte pressuposto: se um mesmo resultado existir nos três motores, este será retornado pelo conjunto dos primeiros cem. Isto não é totalmente verdade, pois os algoritmos de ordenação (*ranking*) de cada motor, podem fazer que o mesmo resultado se encontre no top 100 para um motor e não para outro. O resultado seria então dado como presente no primeiro motor e em falta no segundo. Assumindo o erro causado por esta aproximação, foi definida uma função normalizada, servindo de indicador para a quantidade de resultados duplicados retornados pelas várias fontes individuais.

A função 5.2 indica a eficácia da meta-pesquisa, e é definida como

$$f = \frac{m - n}{t - n} \tag{5.2}$$

com:

$$n = min(|G|, |S|, |N|)$$
$$m = |G \cup S \cup N|$$
$$t = |G| + |S| + |N|$$
$$n <= m <= t$$
$$t - n \neq 0$$

em que t é o total de resultados sem fusão, e m é o total de resultados com fusão, retornados pela Meta-Pesquisa "Web". O valor n é o mínimo de resultados obtidos de todos os motores, e que será usado para a normalização. O conjunto G é formado pelos resultados retornados pelo `Google`, e $|G|$ é o número total de resultados retornados; De forma semelhante, $|S|$ e $|N|$ são os totais de resultados retornados, respectivamente, pelas fontes `Sapo` e `NetIndex`.

Na função 5.2, normalizada para valores entre 0 e 1, o valor 0 acontece quando $m = n$, e indica que todos os resultados sofreram fusão e foram retornados por todas as fontes. O valor 1 acontece quando $m = t$, e indica que não houve nenhuma fusão de resultados das várias fontes, o que equivale a dizer que os resultados são todos diferentes, correspondendo por isso ao valor ideal pretendido para a meta-pesquisa.

Obtiveram-se assim os resultados apresentados na tabela 5.2.

Query	$\|G\|$	$\|S\|$	$\|N\|$	$\|G \cap S\|$	$\|G \cap N\|$	$\|S \cap N\|$	n	m	t	f
benfica	100	75	90	13	2	2	75	250	265	0.921053
bola	101	99	57	18	1	0	57	238	257	0.905
brasil	103	98	100	7	0	0	98	294	301	0.965517
cadeiras	100	100	100	11	0	0	100	289	300	0.945
centro	105	100	100	28	2	6	100	271	305	0.834146
funções	101	100	100	17	0	0	100	284	301	0.915423
jaguar	102	96	100	7	4	1	96	286	298	0.940594
justiça	101	97	100	15	2	5	97	277	298	0.895522
rosa	102	99	90	10	2	1	90	279	291	0.940299
sócrates	99	99	80	10	1	2	80	266	278	0.939394

Tabela 5.2: Resultados da Meta-Pesquisa

Como se pode observar pela tabela, para todos os termos de pesquisa realizados, obteve-se uma eficácia da Meta-Pesquisa sempre superior a 83%, correspondendo este pior valor à pesquisa “centro”. Nesta pesquisa, podemos ver que o `Google` e o `Sapo` retornaram os mesmos 28 resultados, o `Google` e o `NetIndex` retornaram os mesmos 2 resultados, e o `Sapo` e `NetIndex` os mesmos 6 resultados. No extremo oposto, a melhor eficácia (de 96,5%) foi obtida com a pesquisa “brasil”, para a qual apenas o `Google` e o `Sapo`

mostraram em comum 7 resultados, num total de $t = 301$, e que foram fundidos nos $m = 294$ resultados retornados pela Meta-Pesquisa "Web". Para a pesquisa "cadeiras" a eficácia da Meta-Pesquisa seria de 0% caso todos os $t = 300$ resultados fossem fundidos em $m = n = 100$ resultados. Resumindo, se usarmos a função 5.2 para calcular a eficácia média para as pesquisas da tabela 5.2, constatamos que esta é de 92%.

Tudo isto permite concluir que a Meta-Pesquisa é bastante eficaz ao retornar resultados distintos entre as várias fontes.

5.8.2 Tempos da Meta-Pesquisa vs Pesquisa

Nesta secção mostram-se os tempos obtidos com a recuperação dos resultados provenientes dos motores de busca. Os testes foram realizados nas mesmas condições descritas na secção 5.8.1. A ligação à Internet é realizada através de ADSL a 2500kbps de *downstream* e 900kbps de *upstream*. A fonte "Web" compreende os motores `Google`, `Sapo` e `NetIndex`. Mais uma vez a fonte `Tumba!` ficou de fora do teste por se encontrar indisponível.

A Figura 5.17 ilustra os tempos (em milisegundos) de recuperação de resultados pelo protótipo, para a fonte de pesquisa `Sapo` e para a fonte de Meta-Pesquisa "Web", quando se pedem 50, 100, 150, 200, 400 e 1000 resultados.

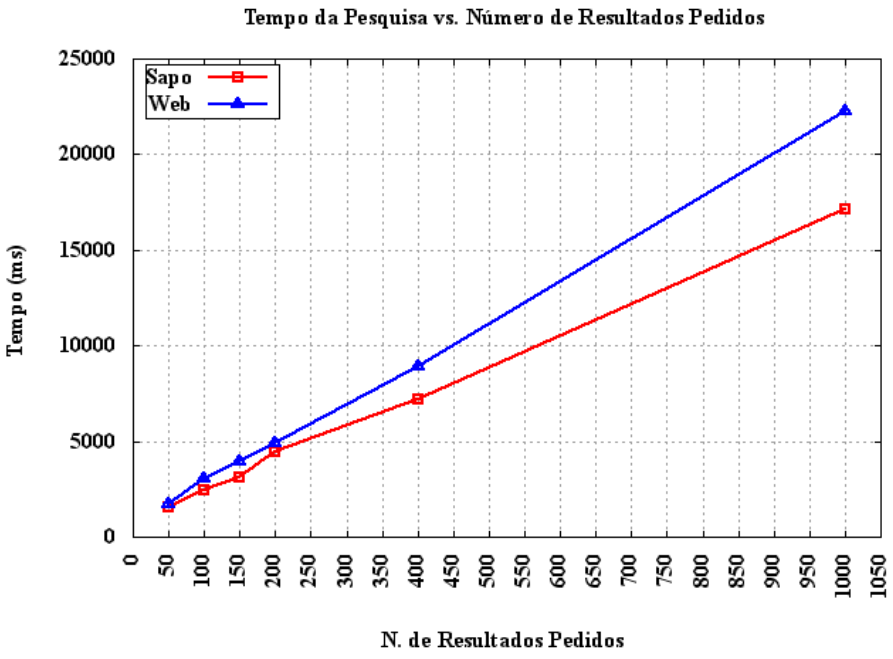


Fig. 5.17: Tempos de Pesquisa em função do Número de Resultados Pedidos.

Como era de esperar, obtém-se um tempo de resposta linear, que aumenta com o número de resultados. No entanto, constata-se que os tempos obtidos pela Meta-Pesquisa "Web" não são o triplo (por termos 3 fontes) dos obtidos para a pesquisa `Sapo`. Tal deve-se ao facto de se estar a usar paralelismo nos pedidos, com as configurações já apresentadas na Tabela 4.1, no Capítulo 4.1.1, na página 37. Isto demonstra que, com a Meta-Pesquisa, estamos também virtualmente a *ganhar tempo* para obter o mesmo número de resultados. Partindo deste pressuposto, fez-se um novo gráfico que, permite visualizar o tempo em

função dos resultados realmente *retornados*, em vez dos resultados *pedidos*. Este gráfico é apresentado na Figura 5.18.

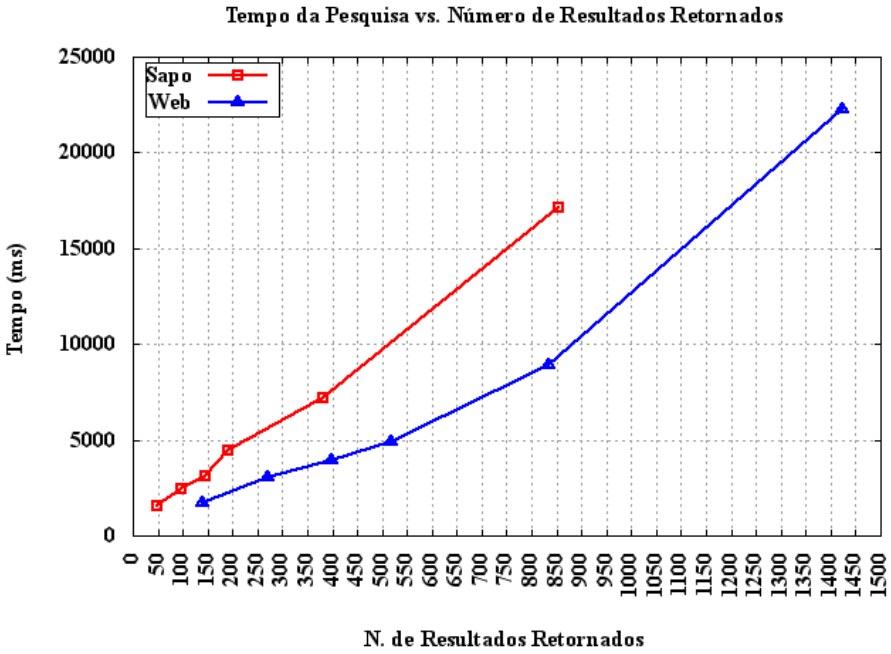


Fig. 5.18: Tempos de Pesquisa em função do Número de Resultados Retornados.

A diferença é que, no caso da Meta-Pesquisa "Web", quando pedimos 50 resultados, estamos a *pedir* 50 resultados *por fonte*, resultando em cerca de 150 resultados *retornados* para o conjunto das três fontes. Da análise da Figura 5.18, é mesmo possível concluir que, para o mesmo número de resultados retornados, a Meta-Pesquisa "Web" precisa apenas de cerca de metade do tempo da pesquisa `Sapo`. Com o paralelismo nas pesquisas, a Meta-Pesquisa permite aproveitar melhor a largura de banda disponível em relação a apenas uma pesquisa a um motor convencional. Os tempos da Meta-Pesquisa só não foram ainda melhores, porque provavelmente atingimos a largura de banda máxima disponível na ligação.

Capítulo 6

Conclusões e Trabalho Futuro

O aumento contínuo e exponencial da quantidade de informação disponível na Web coloca novos e desafiantes problemas aos motores de busca. Um dos problemas imediatos, é que, um motor de busca por si só é incapaz de indexar toda esta enorme quantidade de informação. Em parte, a solução consiste na utilização não de um, mas de vários motores de busca, na esperança que, com a combinação das várias bases de dados de cada um, estejamos a aumentar a cobertura da Web. No entanto, para o utilizador seria penoso lidar com todos os sistemas sempre que pretender realizar uma pesquisa. Assim nasceram os motores de busca *Meta-Search*. Estes oferecem uma interface única e interpõem-se entre o utilizador e os motores de busca individuais, e são responsáveis por obter os termos de pesquisa do utilizador e enviá-la no formato apropriado para cada motor individual. Em seguida, os resultados retornados pelos vários motores são combinados, eliminando os duplicados, são ordenados e apresentados ao utilizador por uma certa ordem de relevância. Neste trabalho são descritas as arquitecturas típicas dos motores de busca convencionais e de Meta-Pesquisa, bem como os vários componentes que os constituem.

Por outro lado, os motores de busca individuais retornam muitos resultados. A grande maioria dos utilizadores apenas olha para os primeiros resultados, tornando imperativo que estes sejam realmente os que lhe sejam relevantes. As pesquisas dos utilizadores são também muitas vezes ambíguas. Por exemplo, uma pesquisa por "jaguar" pode ser entendida como a marca de automóveis, ou como o animal felino. Desta forma, seria útil se conseguíssemos descobrir os vários tópicos semânticos, e separar e atribuir os resultados aos vários tópicos encontrados. É aqui que os algoritmos de *clustering* revelam a sua importância, pois constituem uma forma de aprendizagem não supervisionada, e sem conhecimento prévio dos dados presentes nos resultados e sem a intervenção humana, tentam separar a informação em grupos (*clusters*), atribuindo a cada grupo os documentos que lhe poderão pertencer. Neste trabalho é feita uma introdução aos algoritmos de *clustering* clássicos, e aos algoritmos de *clustering* desenvolvidos especificamente para lidar com texto e documentos e páginas Web. Em seguida são descritos alguns dos sistemas onde esses algoritmos foram implementados e avaliados.

No caso da Web Portuguesa, existem poucos motores de busca com indexador próprio, e os motores de busca *Meta-Search* são inexistentes, havendo apenas directórios com páginas previamente classificadas à mão.

Com base nestes pressupostos, a proposta da solução descrita nesta Tese consiste na alteração da *framework* `Carrot` [30], de modo a funcionar no panorama Português. Esta proposta inclui não só a criação de um *Meta-Searcher* Português, com capacidade de *clustering* de resultados, como uma análise detalhada das modificações e parametrizações necessárias para tornar um tal sistema capaz de lidar com as especificações da nossa língua. Nos componentes de entrada, foram alterados alguns dos componentes fonte, e também desenvolvidos novos, de modo a fornecerem resultados em Português (de Portugal e do Brasil). No pré-processamento foram usadas listas de palavras frequentes em Português, algumas na sua variante Portuguesa e Brasileira, e foram usados algoritmos de radicalização específicos para a língua Portuguesa. Nos componentes de processamento foram testados alguns dos parâmetros dos algoritmos de *clustering* usados: *TRC*, *STC* e *Lingo* [19, 47, 26], de forma a fornecerem os melhores tópicos para a língua Portuguesa. Na visualização, a interface foi igualmente traduzida para Português.

Foram realizados inquéritos a vários utilizadores, os quais avaliaram a relevância dos resultados e dos tópicos formados, bem como a qualidade descritiva destes últimos. Esta avaliação teve por base vinte tipos diferentes de buscas, previamente estudadas e configuradas, as quais foram apresentadas para pesquisa aos utilizadores sem que estes tivessem conhecimento da sua configuração. Em relação à relevância dos resultados, os inquéritos mostraram que o protótipo consegue atingir um valor superior a 60% de precisão para os primeiros 10 resultados. Os resultados dos inquéritos também mostram que, em certos casos, quando alguns parâmetros dos algoritmos de *clustering* são alterados para determinados valores, existe um aumento da relevância e da qualidade atribuídos pelos utilizadores aos tópicos. Podemos então concluir que os melhores valores dos parâmetros obtidos para a língua Inglesa não são necessariamente os melhores valores para a língua Portuguesa.

Os testes também mostram que os algoritmos *Lingo* e *STC* oferecem não só uma melhor qualidade descritiva dos tópicos em relação ao algoritmo *TRC*, como também os seus tempos de execução se terem revelado lineares, em comparação com o comportamento quadrático do *TRC*. Foram igualmente estudados os efeitos dos vários algoritmos de radicalização de palavras na formação dos tópicos. Os testes realizados indicam que, para a língua Portuguesa, é vantajoso não usar radicalizadores, se pretendermos obter alta precisão nos primeiros 10 resultados. Para obter uma boa precisão a partir dos 10 primeiros resultados, o uso de radicalizador passa a ser aconselhado.

Outro tipo de testes incidiu sobre o conjunto de resultados retornados pela Meta-Pesquisa "Web". Os testes mostram que usar Meta-Pesquisa constitui claramente uma vantagem, visto que apenas cerca de 8% dos resultados foram retornados por dois ou três motores de busca, e os restantes 92% serem constituídos por resultados totalmente distintos. Outra grande vantagem da Meta-Pesquisa diz respeito ao tempo necessário para obter o mesmo número de resultados quando em comparação com um motor de busca individual. Neste caso, os testes mostram que, para obter três vezes mais resultados, a Meta-Pesquisa precisou apenas de metade do tempo, atribuindo-se este mérito à paralelização das pesquisas e a uma boa largura de banda.

6.1 Contribuições

As principais contribuições deste trabalho foram as seguintes:

1. Implementação de um motor de Meta-Pesquisa preparado para a língua Portuguesa, e com capacidade de *clustering* dos resultados. Existe a tentativa de obter exclusivamente resultados em Português (de Portugal e do Brasil).

2. Utilização de radicalizadores de palavras preparados para o Português, num sistema de recuperação de informação. No protótipo foram implementados três radicalizadores diferentes para a língua Portuguesa, a saber: *Porter*, *Snowball* (implementação particular do *Porter*) e *RSLP*.

3. Elaboração de uma lista com palavras frequentes em Português (de Portugal e do Brasil). Estas listas foram elaboradas com base nalgumas colecções, incluindo a colecção *CHAVE* elaborada e mantida pela Linguateca Portuguesa, e que se encontra descrita na Secção 3.1.1.

4. *Parsers* HTML para alguns motores de pesquisa Portugueses: `Sapo`, `NetIndex` e `Tumba!`.

5. Algoritmo simples de fusão e ordenação de resultados. O algoritmo foi implementado na Meta-Pesquisa "Web", e permite a atribuição de "pesos" a cada fonte, os quais influenciam a posição dos resultados na lista de resultados.

6. Adaptação de algoritmos de *clustering* de código aberto aplicados à língua Portuguesa. Foram usados e testados os algoritmos de *clustering STC* e *Lingo* que são fornecidos com a versão 3.x da *framework* `Carrot`.

7. Migração e adaptação do algoritmo *TRC* da versão 2.x para a versão 3.x da *framework* `Carrot`. Foram igualmente realizados testes a este algoritmo. Analogamente, foi também iniciada a adaptação do algoritmo *FuzzyAnts* [40], mas no entanto não foi concluída a tempo de ser incluída na interface e nos testes.

6.2 Trabalho Futuro

Presentemente, ainda existe algum trabalho a desenvolver no protótipo testado. Este trabalho passa pela tentativa de resolução de alguns problemas já encontrados, tais como a ocorrência de tópicos duplicados, sendo necessária a normalização da acentuação, e a adaptação às várias formas da língua Portuguesa (Portugal e Brasil). Por exemplo, para o caso da acentuação, tópicos duplicados tais como "Gráficos de Funções" vs "Graficos de Funcoes" e, para o caso das várias variantes (Portuguesa e Brasileira) da mesma palavra, "Bolos de Baptizado" vs "Bolos de Batizado" serem fundidos num só. Outro problema com solução mais imediata será a remoção de tópicos compostos apenas por valores numéricos, o que presentemente já é realizado no *Lingo* (no qual pode mesmo ser parametrizável por expressões regulares), mas em falta no *TRC* e *STC*. Além disso, podem ainda ser adicionadas mais funcionalidades, tais como a implementação de mais fontes de resultados, e relativas a conjuntos específicos de informação (sítios académicos, *blogs*, notícias,

etc.). Poderão, eventualmente, também serem adicionados mais algoritmos de *clustering* e de radicalização, e estudando o seu comportamento.

Finalmente, será necessário complementar o trabalho realizado com mais testes, e com a participação de mais utilizadores na resposta aos inquéritos existentes e a novos inquéritos. Na avaliação da Meta-Pesquisa, será interessante verificar a eficácia quando se varia o número de resultados. À partida a eficácia deverá diminuir à medida que aumenta o número de resultados obtidos dos motores de busca.

Bibliografia

[1] Leonidas Akritidis, George Voutsakelis, Dimitrios Katsaros, and Panayiotis Bozanis. Quadsearch: A novel metasearch engine. In *PCI'07: Proceedings of the 11th Panhellenic Conference on Informatics*, pages 453–466, Patras Hellas, Greece, May 2007.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[4] Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[5] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[6] Monica Desai and Amanda Spink. An algorithm to cluster documents based on relevance. *Information Processing and Management*, 41(5):1035–1049, September 2005.

[7] Sándor Dominich. *The Modern Algebra of Information Retrieval*. Springer Berlin, 1 edition, 2008.

[8] Douglass R. Cutting and David R. Karger and Jan O. Pedersen and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *SIGIR'92: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, Copenhagen, Denmark, June 1992.

[9] Filippo Geraci. *Fast Clustering for Web Information Retrieval*. PhD thesis, Università degli Studi di Siena, Facoltá di Ingegneria, Dipartimento di Ingegneria dell'informazione, 2008.

[10] Filippo Geraci, Marco Pellegrini, Marco Maggini, and Fabrizio Sebastiani. Cluster generation and cluster labelling for web snippets: A fast and accurate hierarchical solution. In Fabio Crestani, Paolo Ferragina, and Mark Sanderson, editors, *SPIRE'06: 13th International Conference on String Processing and Information Retrieval*, volume 4209 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2006.

[11] Eric J. Glover, Steve Lawrence, William P. Birmingham, and Lee C. Giles. Architecture of a metasearch engine that supports user information needs. In *CIKM'99: 8th International Conference on Information and Knowledge Management*, pages 210–216, Kansas City, MO, November 1999. ACM Press.

[12] Antonino Gullì. *On Two Web IR Boosting Tools: Clustering and Ranking*. PhD thesis, Università degli Studi di Pisa, May 2006.

[13] Antonino Gullì and Alessio Signorini. Building an open source meta-search engine. In *WWW'05: Proceedings of the 14th International World Wide Web Conference: Special interest tracks and posters*, pages 1004–1005, New York, USA, 2005. ACM Press.

[14] Marti Hearst. Some thoughts on tagging. In *MIT HCI Human-Computer Interaction Seminar Series Spring '07*, Berkeley, April 2007. UC Berkeley School of Information, UC Berkeley.

[15] Marti A. Hearst and Jan O. Pedersen. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *SIGIR'96: Proceedings of the 19th Annual International Conference on Research and Development in Information Retrieval*, pages 76–84, New York, NY, USA, 1996. ACM Press.

[16] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Upper Sadle River, New Jersey, 1988.

[17] Sophoin Khy, Yoshiharu Ishikawa, and Hiroyuki Kitagawa. A novelty-based clustering method for on-line documents. *World Wide Web*, 11(1):1–37, 2008.

[18] Urszula Krukar. Design and implementation of a text processing application in the eclipse rich client framework. Master's thesis, Poznañ University of Technology, Faculty of Computing Science and Management, Institute of Computing Science, Department of Computing Science, Poland, 2008.

[19] Ngo Chi Lang. A tolerance rough set approach to clustering web search results. Master's thesis, Faculty of Mathematics, Informatics and Mechanics, Warsaw University, December 2003.

[20] Yoelle S. Maarek, Ronald Fagin, and Dan Pelleg. Ephemeral document clustering for web applications. Technical report, IBM Research Report RJ 10186, 2000.

[21] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability*, volume Volume I: Statistics, pages 281–297, 1967.

[22] Bruno Martins. Inter-document similarity in web searches. Master's thesis, University of Lisbon, Faculty of Sciences, October 2004. Also available as Technical Report DI/FCUL TR 4-11.

[23] Weiyi Meng, Clement T. Yu, and King L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.

[24] Mahamed G. Omran, Andries P. Engelbrecht, and Ayed A. Salman. An overview of clustering methods. *Intell. Data Anal.*, 11(6):583–605, 2007.

[25] Viviane M. Orengo and Christian R. Huyck. A stemming algorithm for the portuguese language. In *SPIRE'08: Procedings of the 8th International Conference on String Processing and Information Retrieval*, pages 186–193, November 2001.

[26] Stanislaw Osinski. An algorithm for clustering of web search results. Master's thesis, Department of Computing Science, Poznan University of Technology, Poland, 2003.

[27] Stanislaw Osinski, Jerzy Stefanowski, and Dawid Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Advances in Soft Computing, Intelligent Information Processing and Web Mining, in IIPWM'04: Proceedings of the 12th International Conference on Intelligent Information Systems*, pages 359–368, Zakopane, Poland, May 2004.

[28] Stanislaw Osinski and Dawid Weiss. Conceptual clustering using lingo algorithm: Evaluation on open directory project data. In *Advances in Soft Computing, Intelligent Information Processing and Web Mining, in IIPWM'04: Proceedings of the 12th International Conference on Intelligent Information Systems*, pages 369–378, Zakopane, Poland, May 2004.

[29] Stanislaw Osinski and Dawid Weiss. Carrot^2: Design of a flexible and efficient web information retrieval framework. In *AWIC*, pages 439–444, 2005.

[30] Stanislaw Osinski and Dawid Weiss. Clustering search results with carrot^2. In *Presentation for the Bioinformatics research group*, Dresden, January 2007. Technical University of Dresden.

[31] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[32] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[33] Magnus Rosell. Introduction to information retrieval and text clustering. August 2006.

[34] Frédéric Roulland, Aaron N. Kaplan, Stefania Castellani, Claude Roux, Antonietta Grasso, Karin Pettersson, and Jacki O'Neill. Query reformulation and refinement using nlp-based sentence clustering. In *ECIR'07: 29th European Conference in Information Retrieval*, pages 210–221, 2007.

[35] Walker W. Royce. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, pages 1–9, August 1970. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, 328–338.

[36] Nachiketa Sahoo, Jamie Callan, Ramayya Krishnan, George T. Duncan, and Rema Padman. Incremental hierarchical clustering of text documents. In Philip S. Yu, Vassilis J. Tsotras, Edward A. Fox, and Bing Liu, editors, *CIKM'06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 357–366, New York, USA, 2006. ACM Press.

[37] Gerard Salton. *Automatic Text Processing - The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[38] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the Association for Computing Machinery (ACM)*, 18(11):613–620, 1975.

[39] Samuel Sambasivam and Nick Theodosopoulos. Advanced data clustering methods of mining web documents. *Issues in Informing Science and Information Technology*, 3:563–580, 2006.

[40] Steven Schockaert. Het clusteren van zoekresultaten met behulp van vaagmieren (clustering of search results using fuzzy ants). Master's thesis, University of Ghent, Flanders, Belgium, May 2004.

[41] Cornelis J. van Rijsbergen. *Information Retrieval*. Butterworths, London, Butterworths, London, 2nd edition, 1979.

[42] Kenneth L. Vester and Moses C. Martiny. Information retrieval in document spaces using clustering. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, Kongens Lyngby, Denmark, 2005.

[43] Ellen M. Voorhees. *Implementing agglomerative hierarchical clustering algorithms for use in document retrieval*, volume Information Processing and Management, 22. 1986.

[44] Dawid Weiss. A clustering interface for web search results in polish and english. Master's thesis, Poznañ University of Technology, Poland, June 2001.

[45] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999.

[46] Michal Wróblewski. Hierarchiczny algorytm grupowania stron www wykorzystujacy model przestrzeni wektorowej. Master's thesis, Institute of Informatics, University of Technology, Poznan, Poland, July 2003.

[47] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Research and Development in Information Retrieval*, pages 46–54, 1998.

[48] Oren Zamir and Oren Etzioni. Grouper: a dynamic clustering interface to web search results. In *WWW'99: Proceedings of the 8th International World Wide Web Conference*, volume 31, 11-16, pages 1361–1374, Toronto, Canada, May 1999.

[49] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *KDD'97: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290, California, USA, 1997.

[50] Oren Eli Zamir. *Clustering web documents: a phrase-based method for grouping search engine results*. PhD thesis, University of Washington, 1999.

Apêndices

A Comparações de alguns Motores de Busca

Tabela A.1: Comparações de alguns Motores de Busca

Motor de Busca	M	C	Resultados	Fontes Meta-Search	Clustering	Tipo Clustering	Outras Buscas
AllPlus www.allplus.com	S	S	Combinados, árvore de Texto ou Grafo de Clusters	Ask, Google, Live, Yahoo!	Sim	Hierárquico, Grafo	News, Images, Video, Blogs
BizNetic www.biznetic.com	S	S	Combinados, árvore de Texto	Sim	Sim	Hierárquico	News, Medical, Blogs, Images, USA, por País, por Domínio
CarrotSearch www.carrot-search.com	S	S	Combinados, árvore de Texto	eTools, Google, Yahoo! News, Wikipedia, PubMed, outros	seleccionável entre: FuzzyAnts, HAOG+FI, HAOG+STC, K-Means, STC	Hierárquico e Não-Hierárquico	News, ODP, Jobs, Medical, por Fonte, por Domínio
Carrot2 demo.carrot2.org	S	S	Combinados, árvore de Texto	eTools	Lingo3G	Hierárquico	News, ODP, Jobs, Medical, por Fonte, por Domínio
Clusty www.clusty.com	S	S	Independentes & Combinados, árvore de Texto	AP, Ask, Gigablast, Live, ODP, Yahoo! News, WiseNut, Wikipedia, New York Times, Associated Press, Reuters, Picsearch, BizRate, Blogdigger, BlogPulse, Feedster, Technorati, Indeed, Shopzilla	Vivissimo / Clustering 2.0	Hierárquico	News, Images, Blogs, Shops, Jobs, Gov, Labs, por Fonte, por Domínio, por Linguagem
Cuil www.cuil.com	N	S	Combinados, Categorias	-	Sim	Não-Hierárquico, Drilldown	Video
Dogpile www.dogpile.com	S	S	Combinados, Texto	Google, Yahoo!, Ask, Live	Sim	Não-Hierárquico	Images, audio, video, news, Yellow & White Pages, por Domínio, por Linguagem
eTools www.etools.ch	S	S	Independentes & Combinados, Texto	Altavista, Ask, Atsearch New, Bluewin, Cuil New, Entireweb, Google, Live, Lycos, Search, Seekport, Webofant, Wikia, Yahoo!	Sim	Não-Hierárquico	por País, por Linguagem
FuzzFind www.fuzzfind.com	S	S	Ponderação Combinada, Texto	Google, Live, Yahoo!, Del.icio.us	Sim	Não-Hierárquico	-
Grokker www.grokker.com	S	S	Independentes & Combinados, árvore de Texto ou "Map View", Exportáveis	Yahoo!, Wikipedia, Amazon Books	Lingo3G	Hierárquico	Books
iBoogie www.iboogie.com	S	S	Combinados, árvore de Texto	MSN, AllTheWeb, Netscape, Ask, WiseNut, AltaVista, Yahoo!, Overture, About, Exalead, outros	Clusterizer	Hierárquico	Images, News, Blog, Games, Gov, Publishing, Libraries, Medical, Military, Science, Sports, Tech, Travel
Info info.com	S	S	Combinados, Texto	Google, Yahoo!, MSN, Ask, About	Sim	Não-Hierárquico	Research, News, Images, Video, Health, Shop, Classifieds, Flights, Hotels, Movies, Audio, Yellow & White Pages, Webmail, por Linguagem
iPureSearch www.ipuresearch.com	S	S	Combinados, árvore de Texto	Amazon, eBay, YouTube, MySpace, CraigsList, Wikipedia, ODP	Sim	Hierárquico	Blogs, Forums, Sports, Kids, Domains, Images, Audio, Video, News, Dictionary, eBooks, Health, por País
iZito www.izito.com	S	S	Combinados, árvore de Texto	Google, AOL, Ask, Yahoo!, MSN, Altavista, EntireWeb	Sim	Hierárquico	Video, MP3, Images, por Domínio
IzQuick www.ixquick.com	S	N	Combinados	Achei, Aonde, Altavista, AllTheWeb, Exalead, Qkport, Ask, Gigablast, Wikipedia, Bebo, MSN, Winzy, CNN, NBC, Yahoo!, EntireWeb, ODP	-	-	Video, images, International Phone Book, por Fonte, por Linguagem, por País
Jux2 www.jux2.com	S	S	Combinados, comparação entre fontes	Yahoo!, MSN, Google	Sim	Não-Hierárquico	Comparação de Resultados, por Domínio
Kartoo www.kartoo.com	S	S	Combinados, Gráfico	?	Sim	Não-Hierárquico, Gráfico	Images, Videos, Wikipedia
Mamma www.mamma.com	S	N	Combinados	Ask, About, Entireweb, Business.com, Gigablast, Wisenut, ODP	-	-	Video, Yellow & White Pages, por Fonte

continua na próxima página...

Motor de Busca	M	C	Resultados	Fontes Meta-Search	Clustering	Tipo Clustering	Outras Buscas
MetaCrawler www.metacrawler.com	S	S	Combinados	Google, Yahoo!, MSN, Ask Jeeves, About, MIVA, LookSmart, outros	Sim	Não-Hierárquico	Images, audio, video, news, Yellow & White Pages, por Domínio, por Linguagem
mnemo www.mnemo.org	S	N	Combinados	Synonyms, Translations, Tags, Neighbours	-	-	Images, Digg, Del.icio.us, Youtube
PolyMeta www.polymeta.com	S	S	Combinados	Ask, Yahoo!, Cuil, Google, MSN, Exalead, AllTheWeb, Gigablast	Sim	Hierárquico	News, Images, Blogs, Video, por Fonte
Qksearch www.qksearch.com	S	S	Independentes & Combinados	Yahoo!, Exalead, Google, AOL, Fast, MSN, Altavista	Sim	Hierárquico	Images, MP3, News, Kids, Jobs, Auctions, Foruns, Medical, USGov, por Fonte, por País
Quintura www.quintura.com	?	S	?	?	Sim	Keyword Drilldown	Images, Video, Amazon
Scour www.scour.com	S	N	Combinados, ordenação por fonte	Google, Yahoo!, MSN	-	-	Images, Video
Search www.search.com	S	S	Combinados, fontes selecionáveis	Google, Ask, MSN, ODP	Sim	Não-Hierárquico	Images, Video, People, Shopping, Music, News, Games, por Linguagem
Soovle soovle.com	S	S	Independentes	Wikipedia, Google, Amazon, Yahoo!, Ask, Youtube, Answers.com	Sim	Não-Hierárquico	por Fonte
SortFix www.sortfix.com	S	S	Independentes	Google, Yahoo!, dmoz	Sim	Power Words	por Fonte
TouchGraph www.touchgraph.com	S	S	Demo para apenas Google, Grafo de Clusters	Google	Ligações de páginas	Grafos	-
Viewzi www.viewzi.com	S	?	Independentes & Combinados	Yahoo!, Google, MSN, Ask, Amazon, eBay, Target, WalMart	?	?	Shopping, Videos, Images, News, Books, Music, Recipe, Celebrity, Technology
WebFetch www.webfetchpro.co.uk	S	N	Combinados	Google, Yahoo!, Live, Ask	-	-	Images, Video, News
WebMenu liveportal.webmenu.com	N	S	Lista Única	-	Sim	Hierárquico	USA, News, Images, Video, Blogs, Jobs, Wikipedia, MP3/Audio, Shopping, Auctions, Medical Info, Gov Info (US), Forum, People (US), Kids, por País, por Domínio
WideExplorer www.widexplorer.com	S	N	Independentes	Google, MSN, YouTube, Google News, Wikipedia, Yahoo! Answers, Scribd	-	-	de cada Fonte, por Fonte
ZapMeta www.zapmeta.com	S	S	Combinados	Google, Altavista, AOL, Yahoo!, Live Search, MSN, EntireWeb	Sim	Hierárquico	Images, Video, MP3, por País
Zuula www.zuula.com	S	N	Independentes	Google, Yahoo!, Live, GigaBlast, Exalead, Alexa, EntireWeb, Mahalo, Wikia, Visvo, Mojeek	-	-	Images, Video, News, Blog, Jobs, por Domínio, por Fonte

Legenda: M=Meta-Search; C=Clustering

B Lista de Palavras Frequentes Portuguesas

a à acerca acordo adeus afirma afirmou agora ainda algo algumas alguns ali altura além ambos ano anos antes ao aos apenas apesar apoio apontar após aquela aquelas aquele aqueles aqui aquilo área as às assim associação através atrás até aumento aí baixo banco bastante bem bilhões biliões boa boas bom bons brasil brasileira brasileiro brasília breve cada caminho campanha candidato capital casa caso catorze causa cedo cento central centro cerca certamente certeza cidade cima cinco cinema coisa com comissão como comprido congresso conhecido conselho conta contos contra corrente cultura custa cá câmara da daquela daquelas daquele daqueles dar das de debaixo decisão demais dentro depois deputado desde desligado dessa dessas desse desses desta destas deste destes deve devem deverá dez dezanove dezasseis dezassete dezoito dia diante dias dinheiro direção direcção direita direito diretor director disse diz dizem dizer do dois dos doze duas durante dá dão dúvida e é economia econômica económica ela elas ele eleições eles em embora empresa empresas enquanto entanto entre então equipa equipe era és especial essa essas esse esses esta estado estados estar estará estas estava este estes esteve estive estivemos estivémos estiveram estiveste estivestes estou está estás estão eu eua europa europeia exemplo facto falta fará fato favor faz fazeis fazem fazemos fazer fazes fazia faço federal fez fhc ficou filho filme fim final foi folha fomos for fora foram forma foste fostes frente fui geral governo grande grandes grupo guerra havia história hoje homem hora horas há inflação iniciar inicio internacional início ir irá isso isto janeiro jogo juros justiça já lado lei lhe ligado lisboa livro local logo longe lugar lá maior maioria maiorias mais mal mas me meio melhor menor menos mercado meses mesma mesmo meu meus mil milhões miliões minha minhas ministro ministério momento muito muitos mulher mundo máximo média mês música na nacional nada naquela naquelas naquele naqueles nas nem nenhuma nessa nessas nesse nesses nesta nestas neste nestes no noite nome nos nossa nossas nosso nossos nova novas nove novo novos num numa nunca nao não nível nós número o obra obras obrigada obrigado oitava oitavo oito onde ontem onze os ou outra outras outro outros p para parece parte partido partir passado paucas país países pegar pela pelas pelo pelos perto período pesquisa pessoas plano pode podem poder poderá podia polícia política ponto pontos por porque porquê porto portugal portuguesa portugueses português posição possivelmente posso possível pouca pouco poucos povo prazo presidente preço preços primeira primeiras primeiro primeiros problema problemas processo produtos produção programa projecto projeto promeiro própria próprias próprio próprios próxima próximas próximo próximos ps psd pt pude puderam pôde põe põem público quais qual qualquer quando quanto quarta quarto quase quatro que quem quer quereis querem queres quero questão quieto quinta quinto quinze quais quáis quê r real recursos região relação reportagem república rio sabe sabem saber saúde se segunda segundo segurança sei seis seja sem semana sempre sendo sentido ser seria será serão sete sector setor seu seus sexta sexto sido sim sistema situação sob sobre social sociedade sois somente somos sou sp sua suas sucursal sul são sétima sétimo só tal talvez também tanta tantas tanto tarde te tel tem temos tempo tendes tenho tens tentar tentaram tente tentei ter terceira terceiro terá teu teus teve tinha tipo tive tivémos tivemos tiveram tiveste tivestes toda todas todo todos trabalhar trabalho treze três tu tua tuas tudo tão têm último últimos um uma umas uns us usa usar vai vaia vais valor veja vem vens ver verdade verdadeiro vez vezes viagem vida vinda vindo vinte você vocês vos vossa vossas vosso vossos vários vão vêm vós zero zona

C Classes Java das Fontes de Documentos

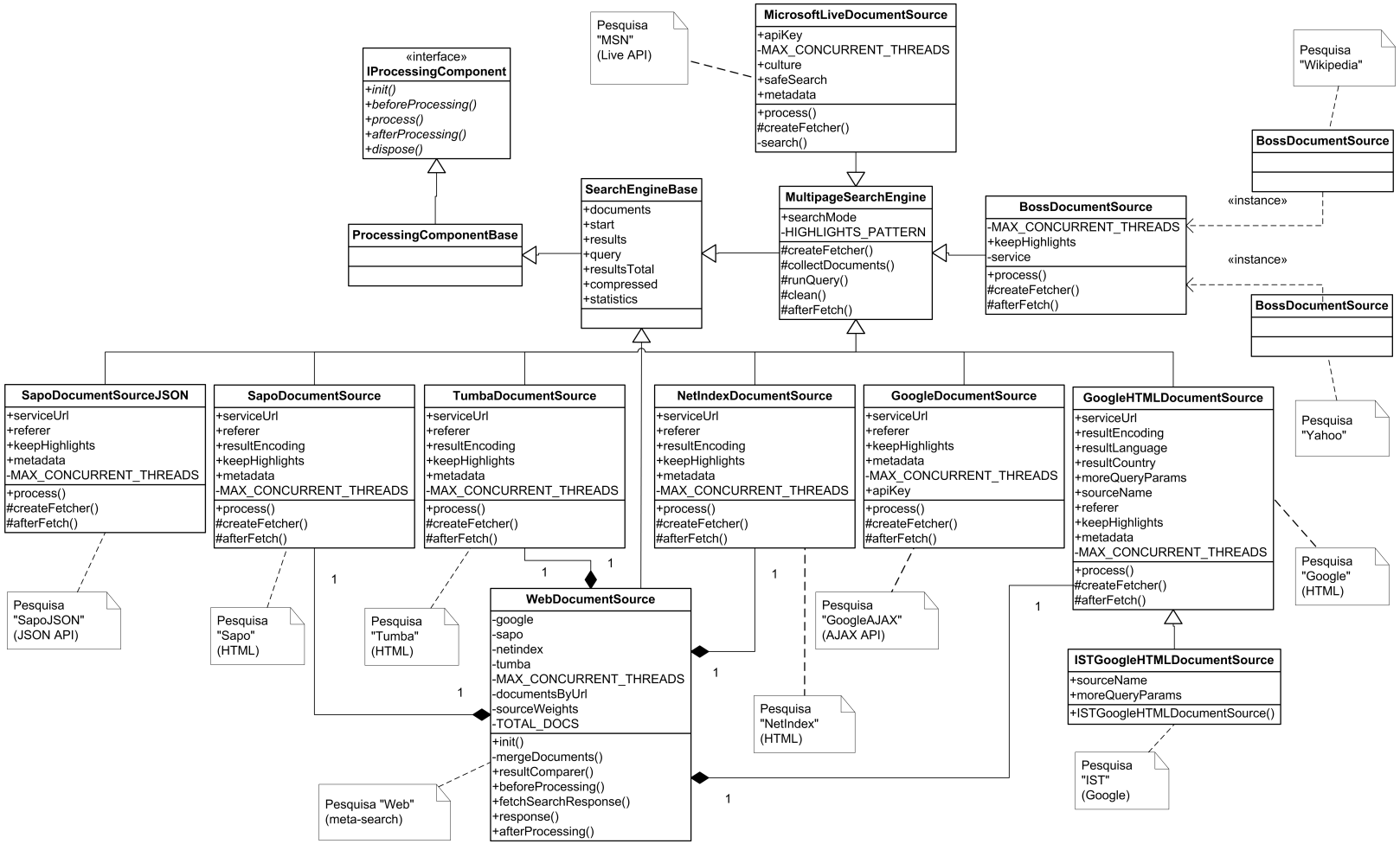
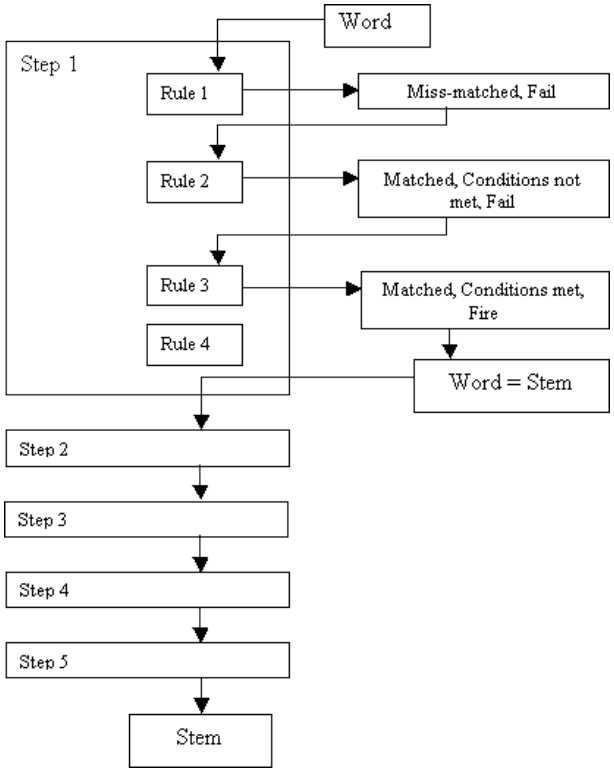
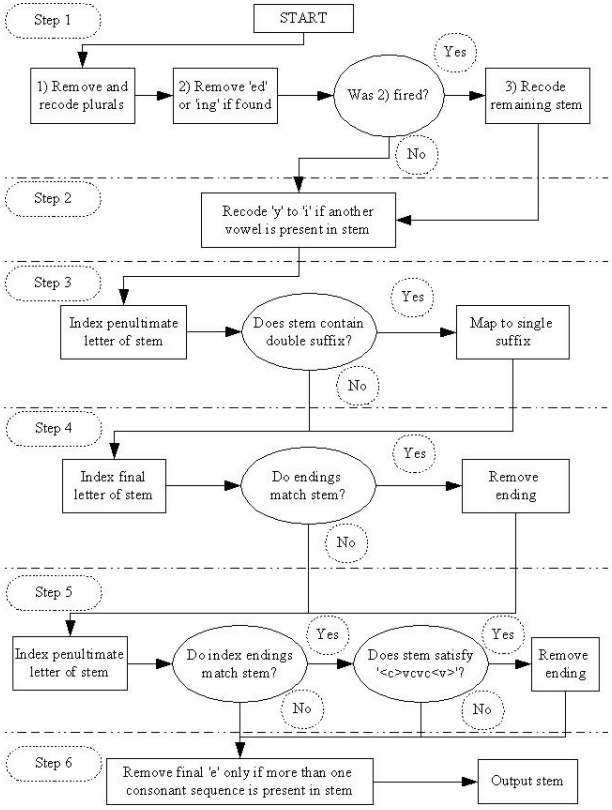


Fig. C.1: Classes Java das Fontes de Documentos.

D Algoritmo *Porter* de Radicalização de Palavras



(a) Fluxo: Regras e condições.



(b) Exemplo para a língua Inglesa.

Fig. D.1: Algoritmo de Radicalização *Porter* [32].

E Algoritmo *RSLP* de Radicalização de Palavras

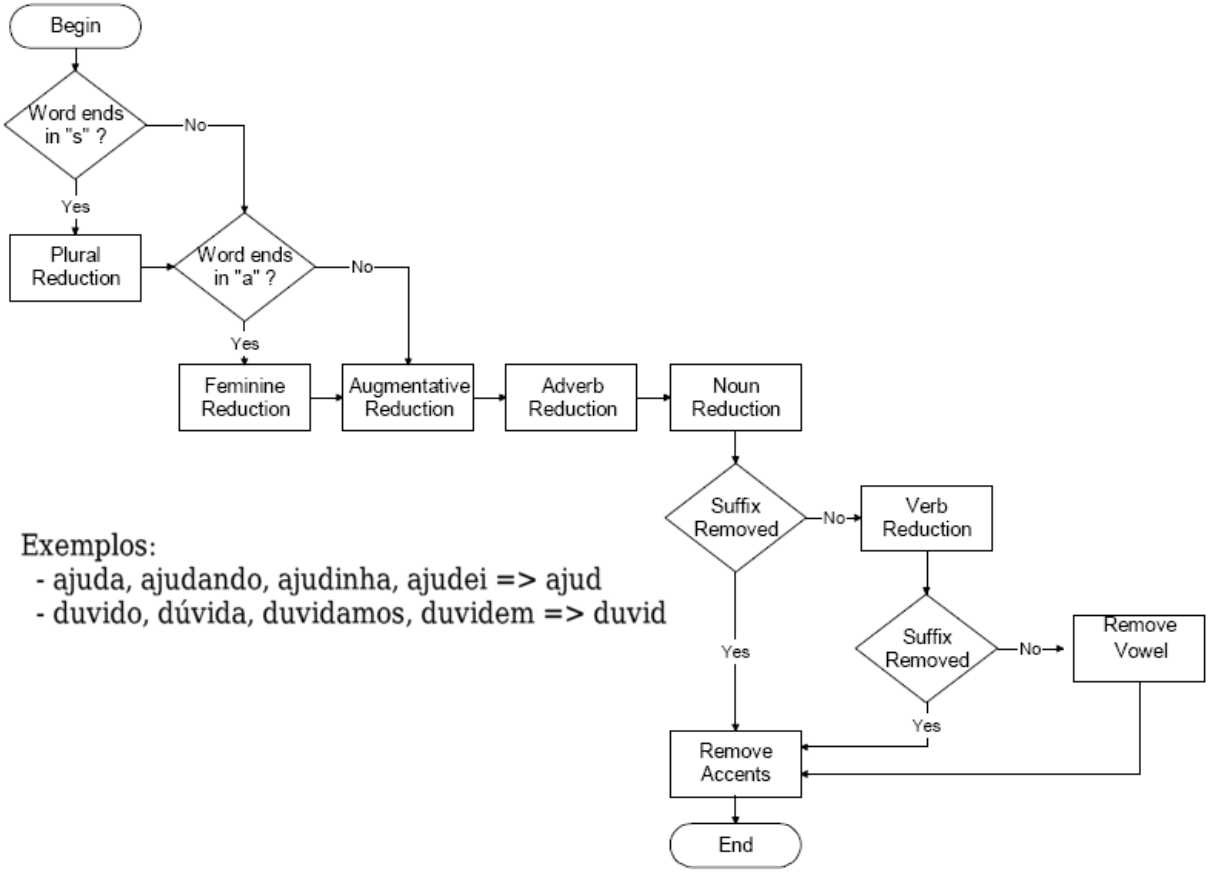


Fig. E.1: Algoritmo de Radicalização *RSLP* [25].

F Inquéritos de Avaliação

Instruções:
Com base na página da interface principal, por favor responda às seguintes questões.
Em caso de dúvida, posicione o rato sobre a questão por uns momentos, até aparecer um texto de ajuda.
Note que todas as respostas são opcionais, mas o objectivo é mesmo responder! Obrigado!

Avaliação dos 20 primeiros Resultados obtidos, num total de 237		
N.	Relevante?	Resultado
#1	◯Sim ◯Não	**A Bola** *19 Set 2009 ... Jogo d'A BOLA .... A BOLA Sociedade Vicra Desportiva. TEN HEROES. You decide. Who will be the 2009 GOLDEN FOOT winner ...www.abola.pt/ - 1 hora atrás - Em cache - Semelhante* [Google, Sapo]
#2	◯Sim ◯Não	**Bola Branca - Renascença - Música e Informação Dia a Dia** *18 Set 2009 ... Opinião; Blogs em Destaque; Frases e Gafes; Prémios Bola Branca. Ribeiro Cristóvão: Doping no Ciclismo. Também tu, Nuno Ribeiro? ...* [Google, Sapo]
#3	◯Sim ◯Não	**O aclubista vai à bola - Ensaio Geral** *Identifique-se Ensaio Geral Edi£o revista e aumentada. Licenciado em cinema e a cursar psicologia. Sou o Ricardo Josã©, e esta ã© a minha pã¡gina. Inicio Autor* [Google, NetIndex]
#4	◯Sim ◯Não	**Jornal a bola** *O jornal A Bola é um jornal desportivo,futebol, atletismo, automobilismo e engloba ainda outras áreas: passatempos, videos, publicações diversas, fórum, chat .* [Google, Sapo]
#5	◯Sim ◯Não	**As mulheres da bola** *As mulheres da bola. Um blog dedicado ao lado mais cor-de-rosa da vida dos jogadores de futebol. Queres conhecer as mulheres dos principais jogadores? ...* [Google, Sapo]
#6	◯Sim ◯Não	**Bola de Neve** *Bola de Neve é um projecto editorial baseado na experiência que consiste em apoiar os educadores e professores.* [Google, Sapo]
#7	◯Sim ◯Não	**Fabrico Próprio » Bola de Berlim** *É só para dizer que já vi uma bola de berlim recheada de chocolate. Confesso que não gosto de bolas de berlim por isso não sei se serão boas :) ...* [Google, Sapo]
#8	◯Sim ◯Não	**O Mistério da Bola de Fogo :: The Mysterious Ball of Fire** *Site oficial da sessão de planetário em exibição no Centro Multimeios de Espinho .* [Google, Sapo]
#9	◯Sim ◯Não	**O que é a Eco-bola** *A Eco-bola (Öko-ball) é um dispositivo que permite reduzir a utilização de detergentes na lavagem de roupa entre 80 a 100%. A Eco-bola é feita de polipropileno ecológico e ...* [Google, Sapo]
# 10	◯Sim ◯Não	**A verdadeira história da ida de CR7 para Madrid - Sapo Videos** *5 para a Meia Noite Á Lei da Bola Beat Generation Carne pra canhão Contemporâneos Gato ... Menos de 1 ano depois do Porto roubar Rodríguez ao Benfica, volta a atacar e rouba o ...* [Google, Sapo]
# 11	◯Sim ◯Não	**FC Porto PlanetaPortugal - Lucho tocou na bola 72 segundos** *Ainda tentou repetir a façanha por um par de vezes, mas houve sempre algo que falhou. No total, Lucho endossou a bola a companheiros por 33 vezes. ...* [Google, NetIndex]
# 12	◯Sim ◯Não	**Bola Azul - Expresso.pt** *Todas as notícias que marcam a actualidade ao minuto, no seu jornal Expresso.* [Google, Sapo]
# 13	◯Sim ◯Não	**Bola de Goalball (ref. 4804)** *Formato do ficheiro: PDF/Adobe Acrobat - VisualizarO seu browser pode não ter um leitor de ficheiros PDF. O Google recomenda que veja o site em versão de texto.Bola standard da International Blind Sports Association, usada para a prática de ... Bola azul, audível, com sistema de três guizos internos, ...* [Google, Sapo]
# 14	◯Sim ◯Não	**JornalismoPortoNet Weblog: As novidades do site de A Bola** *JornalismoPortoNet Weblog Licenciatura em Ciências da Comunicação - Jornalismo, Assessoria, Multimédia da Universidade do Porto Weblog de Jornalismo Online / Ciberjornalismo* [Google, Sapo]
# 15	◯Sim ◯Não	**Últimas Notícias » A Bola » junho 7, 2009** *Últimas notícias das principais fontes de informação em Portugal com notícias nacionais e internacionais.* [Google, Sapo]
# 16	◯Sim ◯Não	**Bolinhos 3D: Bola Futebol** *skip to main	skip to sidebar Bolinhos 3D Bolos especiais para dias especiais e porque acredito que todos os bolos devem ser tã£o saborosos quanto a sua aparã^ncia sugere... Bola Futebol* [NetIndex]
# 17	◯Sim ◯Não	**Geometria: Parã¡bola** *Por aqui colocamos alguns problemas, construã§ãµes e animaã§ãµes de geometria que nos foram sugeridos pelo estudo ou por necessidades do ensino - bã¡sico e secundã¡rio.* [NetIndex]
# 18	◯Sim ◯Não	**A Bola - abola.pt - abola.pt** *Jogo d'A BOLA .... PublicidadeSugestõesTotolotariasProgramação TVBota de ouro ESM · RSS · PDA Definir como Home Page. © A BOLA Sociedade Vicra Desportiva.* [Google]
# 19	◯Sim ◯Não	**Atelier Bola de Neve - Criação de boneco em fimo personalizados** *Bonecos / Esculturas em Fimo Personalizados feitos a mão - ideal para prendas originais, casamento, aniversário e muito mais* [Sapo]
# 20	◯Sim ◯Não	**A Bola é minha!** *A Bola é minha! é o jornal desportivo mais humorado de Portugal. Notícias, imagens, anedotas, videos, e sons do Sporting, Benfica, Porto, Boavista, ...* [Google]

O seu nome (opcional):

Outros comentários:

Submeter | Limpar

Fig. F.1: Avaliação de Resultados para a busca “bola”.

Instruções:
Com base na página da interface principal, por favor responda às seguintes questões.
Em caso de dúvida, posicione o rato sobre a questão por uns momentos, até aparecer um texto de ajuda.
Note que todas as respostas são opcionais, mas o objectivo é mesmo responder! Obrigado!

Avaliação dos Tópicos formados para os resultados obtidos			
N.	Relevante?	Explicativo?	Tópico
#1	◯ Sim ◯ Não	◯ Sim ◯ Não	**Bola Futebol** [*Contém 26 Documentos*]
#2	◯ Sim ◯ Não	◯ Sim ◯ Não	**Jornal a Bola** [*Contém 23 Documentos*]
#3	◯ Sim ◯ Não	◯ Sim ◯ Não	**Jogadores** [*Contém 9 Documentos*]
#4	◯ Sim ◯ Não	◯ Sim ◯ Não	**Jornal Desportivo** [*Contém 9 Documentos*]
#5	◯ Sim ◯ Não	◯ Sim ◯ Não	**Blog** [*Contém 8 Documentos*]
#6	◯ Sim ◯ Não	◯ Sim ◯ Não	**Jogos de Jogos** [*Contém 8 Documentos*]
#30	◯ Sim ◯ Não	◯ Sim ◯ Não	**Ref** [*Contém 3 Documentos*]
#31	◯ Sim ◯ Não	◯ Sim ◯ Não	**Revista de Imprensa** [*Contém 3 Documentos*]
#32	◯ Sim ◯ Não	◯ Sim ◯ Não	**Â a Bola On-lineâ** [*Contém 3 Documentos*]
#33	◯ Sim ◯ Não	◯ Sim ◯ Não	**Blogaveiro Blogaveiro Pretende-se ponto de Encontro** [*Contém 2 Documentos*]
#34	◯ Sim ◯ Não	◯ Sim ◯ Não	**Bola Pequena** [*Contém 2 Documentos*]
#35	◯ Sim ◯ Não	◯ Sim ◯ Não	**Bola com Livros** [*Contém 2 Documentos*]
#36	◯ Sim ◯ Não	◯ Sim ◯ Não	**Bola de Rugby** [*Contém 2 Documentos*]
#37	◯ Sim ◯ Não	◯ Sim ◯ Não	**Entra na Baliza** [*Contém 2 Documentos*]
#38	◯ Sim ◯ Não	◯ Sim ◯ Não	**Jesus de Benfica** [*Contém 2 Documentos*]
#39	◯ Sim ◯ Não	◯ Sim ◯ Não	**Liga Sagres** [*Contém 2 Documentos*]
#40	◯ Sim ◯ Não	◯ Sim ◯ Não	**Madeira** [*Contém 2 Documentos*]
#41	◯ Sim ◯ Não	◯ Sim ◯ Não	**Malucos que Prefere Jogar pelo Chão** [*Contém 2 Documentos*]
#42	◯ Sim ◯ Não	◯ Sim ◯ Não	**Notícias que Marcam a Actualidade ao Minuto** [*Contém 2 Documentos*]
#43	◯ Sim ◯ Não	◯ Sim ◯ Não	**Posicionamento das Bolas 1.3** [*Contém 2 Documentos*]
#44	◯ Sim ◯ Não	◯ Sim ◯ Não	**Vídeo Realizado** [*Contém 2 Documentos*]
#45	◯ Sim ◯ Não	◯ Sim ◯ Não	**Vídeos e Fotos** [*Contém 2 Documentos*]
#0	◯ Sim ◯ Não	◯ Sim ◯ Não	**Outros Tópicos** [*Contém 85 Documentos*]

O seu nome (opcional):

Outros comentários:

Submeter Limpar

Fig. F.2: Avaliação de Tópicos para a busca “bola”, usando os algoritmos *Porter PT* e *Lingo*.