Pesquisador Multi-Agente na Web

André Barbosa
Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
+351 91 823 50 68
ei01043@fe.up.pt

Filipe Montenegro
Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
+351 91 630 42 14
ei01104@fe.up.pt

SUMÁRIO
[150 a 200 palavras]

1. INTRODUÇÃO
O sistema multi-agente é constituído por diversos agentes cuja função é pesquisar páginas web de acordo com palavras-chave fornecidas pelo utilizador e do perfil do próprio utilizador (conjunto de preferências ou palavras-chave armazenadas). A pesquisa de informação é efectuada a partir de um endereço inicial, definido pelo utilizador, ou através dos endereços de *n* páginas retornadas pelo google. O agente pesquisador deve analisar não só a(s) página(s) inicial(is), mas também os seus "links" para verificar se as novas páginas contêm as palavras a procurar. Deve existir um limite máximo de profundidade das "sub" páginas a pesquisar.

Deve ser dada ao utilizador a possibilidade de classificar algumas páginas encontradas para guiar o pesquisador na sua tarefa. Essa classificação pode ser um de dois valores: interessa/não_interessa. O algoritmo de análise pode conter sub-algoritmos que considere relevantes para melhor qualificar as páginas encontradas, tais como: realizar a eliminação de palavras desnecessárias e obter palavras derivadas. A qualificação de uma página deriva da consideração de número de palavras-chave encontradas, posição das palavras na página, e do perfil do utilizador.

Neste sistema multi-agente, cada agente realiza a pesquisa num um domínio Web diferente, comunicando entre si os resultados obtidos.

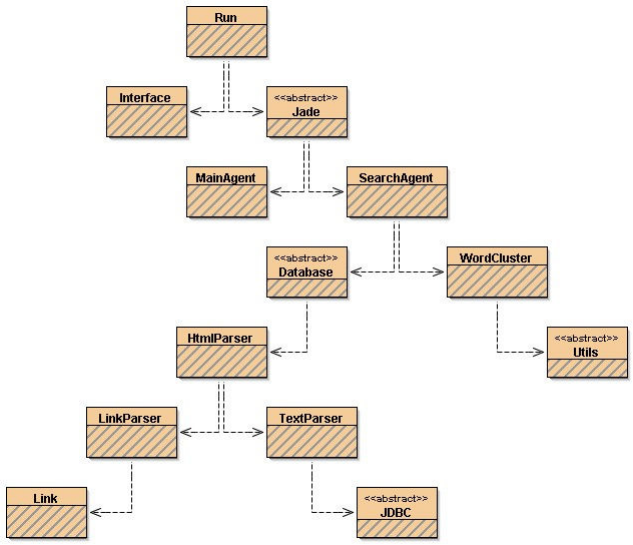
2. DESCRIÇÃO
Nesta secção será descrito de uma maneira geral o trabalho, ou seja, os módulos da arquitectura, os esquemas de representação de conhecimento, as metodologias e os processos de raciocínio a utilizar na análise e na implementação do projecto.

2.1 Divisão e Planeamento do Trabalho
A abordagem ao trabalho encontra-se dividida em diversas partes distintas: *api* do google, *html parser*, *word clustering*, ligação à base de dados e utilização do Jade. Em cada uma das partes é feita uma análise do problema seguida de testes com alguns exemplos atingindo depois uma solução para o problema intermédio. No final da análise individual é necessário proceder à junção de todas as componentes. Nesta fase do projecto já analisamos todas as partes e em quase todas temos já boas soluções para a implementação global do trabalho.

Para melhor compreensão da divisão e planeamento do trabalho, do que fizemos concretamente e do que tencionamos fazer ainda apresentamos a seguir o mapa de *gant*:

[MAPA GANT]

2.2 Diagrama de Classes
O nosso projecto será desenvolvido em *Java* e terá como arquitectura de classes a da figura seguinte:



Como se pode verificar no diagrama a classe executável do nosso projecto á classe *Run*. Esta procede ao estabelecimento das condições necessárias para o funcionamento correcto do programa como a definição de variáveis de ambiente e lança a interface.

Os agentes são definidos nas classes *MainAgent* e *SearchAgent* sendo que o primeiro apenas agrupa os dados resultantes das pesquisas efectuadas pelos outros agentes não utilizando portanto nenhuma outra classe do sistema. Por outro lado a classe SearchAgent interage com a base de dados dos *links* guardados na classe abstracta *Database* e com a classe *WordCluster* para proceder ao agrupamento em *clusters* das palavras encontradas no texto analisado.

A análise duma página para posterior remoção dos links e texto presente é feita nas classes *HtmlParser*, *LinkParser*, *TextParser* e *Link*. Os *links* são guardados, como referido anteriormente, através da classe *Database* numa lista ligada e o texto é mantido numa base de dados *SQL* através da interface *JDBC*.

Para a implementação do algoritmo de agrupamento de palavras semelhantes em *clusters* apenas é necessário o uso da classe *WordCluster* que por sua vez recorre à classe abstracta *Utils* para executar algumas funções especiais desenvolvidas com objectivos muito específicos da aplicação.

Optámos por referenciar no diagrama as classes *Jade* e *JDBC* mas estas não fazem parte do nosso projecto, simplesmente foram referidas para um melhor entendimento do modo de funcionamento do projecto.

2.3 Metodologias e Processos de Raciocínio

Durante a execução do projecto foi necessário recorrer a diversas estratégias para a resolução de problemas que foram surgindo. A descrição dos diversos problemas encontrados e das técnicas adoptadas para a sua resolução são descritos nos tópicos a seguir apresentados.

2.3.1 Detecção de links e texto numa página web

Para a detecção de links numa página web desenvolvemos um parser que procura a tag <a, que em html significa que existe um link, e dentro dessa tag o conteúdo de href é o próprio link. É necessário ter em conta que o campo href pode começar com ./ ou ../ sendo portanto necessário ir ao link inicial e fazer as respectivas alterações para que o link seja válido. Ainda importante é o facto de os links não puderem referenciar ficheiros de extensões inválidas para análise textual como pdf ou exe.

Relativamente à detecção de texto há vários aspectos a ter em conta visto que o texto pode estar contido num parágrafo identificado em html pela tag , numa tabela nomeadamente na tag <td><td> e em todo o conteúdo duma página web que não esteja inserida em nenhuma tag.

Os processos de detecção de links e texto serão sujeitos ainda a alguns melhoramentos no decorrer da implementação já que estes evidenciam algumas falhas. Tentaremos minimizar estas falhas e identificar o máximo número de links correctos possível e a maior quantidade de texto podendo até ser necessário recorrer a um parser html já implementado.

2.3.2 Listagem de palavras válidas/inválidas

Na análise de um texto surgem imensas palavras que não serão utilizadas para o agrupamento das mesmas em clusters. Destas fazem parte por exemplo os artigos e as preposições que ocorrem com muita frequência e não têm nenhum significado associado sendo portanto necessário descarta-las. Artigos como a, o, os e as e preposições como em, por, entre e de serão portanto descartados da pesquisa.

Relativamente a verbos e conjugações verbais neste momento optámos apenas por retirar da análise aqueles que ocorrem com muita frequência como por exemplo é, foi e fica.

Uma maneira simples, neste momento a utilizada, de identificar palavras válidas é o seguinte algoritmo:

- Se palavra no inicio da frase e não pertence à lista de palavras inválidas então é valida.
- Se palavra não pertence à lista de palavras inválidas e é precedida de um artigo ou uma preposição então palavra válida.
- Todas as palavras que numa frase não verifiquem estas condições são consideradas inválidas.

Consideremos o exemplo com as seguintes frases: Lisboa é a capital de Portugal. Portugal fica no sul da Europa. O campeonato da Europa de 2004 foi em Portugal. A Expo-98 foi realizada em Lisboa.

É fácil verificar que, utilizando o algoritmo anterior, obtemos a seguinte lista de palavras válidas: Lisboa, capital, Portugal, sul, Europa, campeonato, 2004, Expo-98. Fica assim provada a eficiência deste algoritmo visto que estas são efectivamente as palavras a serem utilizadas numa futura aglomeração por semelhança. De fora fica a palavra realizada, que apesar de não constar da lista de palavras inválidas foi identificada como tal visto não ser precedida de uma preposição nem de um artigo.

2.3.3 Algoritmo do word clustering

Para procedermos à aplicação do algoritmo de word clustering (agrupamento de palavras semelhantes) num determinado texto é necessário primeiro calcular o número de ocorrências de uma determinada palavra com todas as outras na mesma frase. Este resultado é guardado num vector e cada palavra válida do texto tem um associado.

Ainda do exemplo anterior, a matriz de todos os vectores obtida é a seguinte:

	Lisboa	capital	Portugal	sul	Europa	campeonato	2004	EExpo-98
Lisboa	0	1	1	0	0	0	0	1
capital	1	0	1	0	0	0	0	0
Portugal	1	1	0	1	1	1	1	0
sul	0	0	1	0	1	0	0	0
Europa	0	0	1	1	0	0	0	0
campeonato	0	0	1	0	1	0	1	0
2004	0	0	1	0	1	1	0	0
Expo-98	1	0	0	0	0	0	0	0

O algoritmo de word clustering por nós analisado indica que a similaridade entre duas palavras corresponde ao co-seno normal entre os vectores de ocorrências correspondentes. Assim, para cada par de vectores o co-seno entre eles é dado pela seguinte fórmula:

$$sim(\vec{w_a}, \vec{w_b}) = cos(\vec{w_a}, \vec{w_b}) = \frac{\sum_{j=1}^{m} w_a^j \times w_b^j}{\sqrt{\sum_{j=1}^{m} w_a^{j^2}} \times \sqrt{\sum_{j=1}^{m} w_b^{j^2}}}.$$

Da fórmula é importante dizer que w_a significa o vector da palavra a, w_b o vector da palavra b e o m o número total de palavras, ou seja, o número total de vectores.

Aplicando esta parte do algoritmo no nosso exemplo no par de palavras (Lisboa, capital) obtemos um valor de similaridade igual a 0,408 e no par (2004, Expo-98) o valor obtido foi 0.

A restante parte do algoritmo, ou seja, o agrupamento das palavras semelhantes em clusters ainda não foi muito bem analisado faltando decidir se havemos de usar um algoritmo do tipo hierárquico, não hierárquico ou incremental. Assim, nesta fase não podemos adiantar mais sobre o algoritmo a utilizar.

2.3.4 Características e Funcionamento dos agentes

Decidimos utilizar no projecto dois tipos de agentes diferentes: o *MainAgent* e o *SearchAgent*. Podemos dizer que se trata de uma estrutura cliente-servidor visto que o *MainAgent* está sempre a correr tal como um servidor e vários agentes do tipo *SearchAgent* são lançados para fazer o tratamento de texto respondendo depois para o *MainAgent*.

Os agentes do tipo *SearchAgent* são responsáveis por executar o tratamento do texto contido num determinado *link* e pela execução do algoritmo de *word clustering* sobre esse texto. O resultado é posteriormente enviado ao agente principal (*MainAgent*).

O *MainAgent* tem de receber as respostas de todos os agentes de pesquisa e fazer o agrupamento dos dados recebidos informando depois a aplicação do resultado obtido.

2.4 Representação do Conhecimento

Serão descritas as estruturas utilizadas para guardar os diferentes tipos de dados existentes ao longo da aplicação que vão desde links a agentes. Cada uma foi escolhida de acordo com a finalidade pretendida e tendo em conta outros dados como a memória utilizada e o tempo gasto na leitura e escrita dos dados.

2.4.1 Armazenamento dos links

Após a identificação de links numa página web estes são guardados numa lista ligada como um objecto do tipo Link que contém o link propriamente dito e a prioridade associada. Se a aplicação detectar um link que já existe na lista ligada procede-se à verificação da sua prioridade e, se esta for superior à contida na lista, o link existente é removido e o novo adicionado. Caso o link novo não tenha prioridade superior o programa simplesmente ignora-o.

No final, quando todos os links tiverem sido identificados o programa efectua uma ordenação da lista pela prioridade para uma melhor percepção.

2.4.2 Base de dados para as palavras

Depois de ter realizado alguns testes com ficheiros relativamente grandes verificamos que o sistema não tinha memória suficiente para guardar todas as palavras nem para proceder ao cálculo dos vectores das ocorrências. Assim, as palavras válidas de um determinado texto serão mantidas numa base de dados do tipo *SQL*, o que facilita bastante em termos de memória utilizada.

A tabela da base de dados que mantém as palavras terá apenas um campo do tipo *string* correspondente à palavra. Isto verifica-se já que não é necessário que a tabela contenha um campo do tipo código devido ao facto de todas as entradas terem de ser diferentes.

Para a implementação desta parte do projecto utilizaremos a tecnologia *JDBC*, que permite ligar directamente bases de dados do tipo *SQL* a programas escritos em Java. Não podemos adiantar muito mais sobre esta tecnologia visto ainda não estarmos muito bem informados.

2.4.3 Vectores de Ocorrências

Como já foi descrito anteriormente os vectores de ocorrências correspondem ao número de ocorrências que uma palavra tem com todas as outras na mesma frase.

Para guardarmos os vectores de ocorrências de todas as palavras optámos por criar um *array* de inteiros bidimensional com tamanho m x m sendo m o número total de palavras. Esta solução parece a melhor visto que um *array* é uma estrutura simples, logo com tempo de acesso bastante rápido e curto espaço de ocupação na memória.

3. TRABALHO REALIZADO

[introdução ao trabalho realizado]

3.1 API do Google

O *Google* disponibiliza uma *API* em Java que recebe como argumentos o tipo de operação a executar e o se quer pesquisar, caso de a operação ser a pesquisa. Os resultados da pesquisa são retornados num formato específico contendo várias informações desde o *URL* da página, o título, o sumário entre outros.

Esta *API* já foi testada por nós e até já fizemos algumas alterações nas classes disponibilizadas alterando a saída dos resultados para um ficheiro de texto. Procedemos também à remoção do *URL* de cada resultado disponibilizada pelo Google.

A seguir demonstramos um exemplo de utilização da *API* do *Google* efectuando uma pesquisa à palavra sapo:

```
[
URL  = "http://blogs.sapo.pt/"
Title = "Blogs do <b>SAPO</b>"
Snippet = "Nos Blogs do <b>Sapo</b> pode publicar os seus pensamentos
Directory Category = {SE="", FVN=""}
Directory Title = ""
Summary = ""
Cached Size = "25k"
Related information present = true
Host Name = ""
]
```

3.2 HTML Parser

[mostrar um exemplo da nosso parser html

[falar de outros parsers mais eficientes disponíveis]

[mostrar um exemplo com um desses parsers]

3.3 Word Clustering

[mostrar os resultados em diferentes ficheiros da remoção de palavras válidas de um texto]

[mostrar o resultado da ocorrência das palavras válidas na mesma frase]

[aplicar o algoritmo nos dados obtidos na co-ocorrência e verificar os resultados com posterior comentário aos mesmos]

3.4 Jade

[mostrar um exemplo simples do tutorial]

[falar da comunicação que vai haver no nosso projecto e mostrar um exemplo]

4. REFERÊNCIAS

[1] API do Google

http://www.google.com/apis

[2] HTML Parser
http://htmlparser.sourceforge.net/
http://www.quiotix.com/downloads/html-parser/
[3] Word Clustering
http://pespmc1.vub.ac.be/CLUSTERW.html
http://www.di.ubi.pt/~pln/
http://tahoe.inesc-id.pt/
[4] Jade
http://sharon.cselt.it/projects/jade/
[5] Java
http://java.sun.com