Osvaldo C. de Matos Júnior

Identificando Bases de Dados Específicas na Web

Feira de Santana - BA, Brasil

07 de abril de 2008

Osvaldo C. de Matos Júnior

Identificando Bases de Dados Específicas na Web

Monografia apresentada como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação pela Universidade Estadual de Feira de Santana.

Orientador:
João Batista Rocha Júnior

Co-orientador:
Ângelo Conrado Loula

Departamento de Exatas
Universidade Estadual de Feira de Santana

Feira de Santana - BA, Brasil

07 de abril de 2008

Resumo

Estudos recentes apontam que a maior e mais valiosa porção da informação disponível na Internet não é acessível a buscadores convencionais, esta informação está armazenada em bases de dados invisíveis que só podem ser acessadas através de formulários de consulta em sites de conteúdo especializado. Este trabalho propõe uma solucão para identificação automática destas bases na Web através de uma crawler especializado, capaz de navegar pela *Web* de forma inteligente e eficiente.

Abstract

Recent studies point that the largest and more valuable portion of the information available on the Internet can not be easily accessed by conventional search engines. This hidden information lies behind the search fields of specialized-content Websites. This work presents a heuristic to automatically identify this databases on the *Web* through a specialized *Web* Crawler, which is capable of navigating the *Web* in an intelligent and efficient manner.

Agradecimentos

Agradeço à minha mãe Noélia Carneiro da Silva Matos e ao meu pai Osvaldo Carneiro de Matos pela vida e por está aqui neste momento. À minha querida irmã Kilma Carneiro de Silva Matos, irmão Leandro Carneiro da Silva Matos e meu avô Odiniloel Ferreira da Silva (*in memorian*) por acreditarem na minha capacidade e ter me dado força pra conseguir seguir em frente. Agradeço à minha namorada Vera Leilane pela paciência e por ter me acompanhado nestes difíceis cinco últimos anos. Agradeço novamente a estes, aos amigos de Riachão do Jacuípe e aos colegas do curso de Engenharia de Computação da UEFS, todos vocês conseguiram me dar o apoio moral e emocional necessário para conseguir transpor as barreiras impostas pela vida.

Agradeço também aos meus mestres que me auxiliaram ao longos destes anos a alcançar o aprendizado com bastante dedicação, não desistindo mesmo naqueles momentos onde parecia impossível prosseguir. Em especial ao Prof. João Batista, pela tolerância, atenção e motivação, que foi determinante na minha formação acadêmica e profissional, comportando-se como professor, amigo, pai ou patrão, o qual sempre respeitei e devo grandes favores. Ao Prof. Roberto Bittencourt que sempre acreditou em nosso potencial, nos deu auto-estima e nos motivou durante estes anos de faculdade a lutar por nossos sonhos. A Ângelo Loula por ter me conduzido nas orientações finais deste trabalho.

Também agradeço ao pessoal do Goshme: Daniel Murta, Rafael Costa, Rodrigo Barreto e Gustavo Maia; pela oportunidade na empresa e por acreditarem que, mesmo ainda sendo um estudante, conseguiria atender às necessidades de pesquisa e desenvolvimento dentro da empresa.

A todos estes e todos os outros que contribuíram direto ou indiretamente na minha vida e, consequentemente, ajudaram de alguma forma na realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

"*Seja mais sábio do que outras pessoas se puder,*
mas não diga isso a elas."
Lord Chesterfield

1 Introdução

Desde o seu surgimento, em 1957, a Internet vem crescendo rapidamente, principalmente após o surgimento do WWW (*World Wide Web*), ou simplesmente *Web* (teia), composto por documentos textuais, imagens, sons e outros, que mantém conexões (*hyperlinks*) com outros sites ou páginas. O WWW permitiu um maior compartilhamento de informações entre seus usuários, estimados em mais de 1 bilhão em 2007 segundo IWS (2007).

Para auxiliar os usuários na localização da informação desejada, surgiram sites especializados em localizar informações em outros sites na *Web*, divididos em dois tipos: sistemas de diretórios e motores de busca. Os sistemas de diretórios, ou catálogos, possuem as informações organizadas e classificadas em categorias temáticas com a ajuda de pessoas (BRANSKI, 2000). No entanto, a ocorrência do termo é verificada apenas no título e descrição dos itens cadastrados, desprezando todo o texto presente nas páginas HTML apontadas. Já os motores de busca (máquinas de busca, engenhos de busca, buscadores gerais, ou simplesmente buscadores) possuem seu índice - estrutura de dados com relacionamento entre termos e URLs, criado automaticamente, capturando as informações contidas no conteúdo de um endereço da *Web* visitado com auxilio de programas chamados *crawlers*. *Crawlers* são programas que cruzam a *Web* coletando ou atualizando páginas para um servidor central para que sejam indexadas (BAEZA-YATES; RIBEIRO-NETO, 1999), realimentando o processo com novos endereços coletados.

O conjunto das páginas *Web* que são apontadas por *hyperlinks* presentes em outras páginas define a **Web superficial** (*Surface Web*) (BERGMAN, 2001), estimada em 170 terabytes de informação em 2002 (LYMAN et al., 2003). É nesta porção da *Web* que os buscadores convencionais costumam coletar seus documentos, visitando os links da teia para formar sua base de dados, necessária à consulta dos usuários. Estes buscadores tentam agregar o maior número de páginas das mais diversas áreas de conhecimento, por isso são chamados **buscadores gerais**. Em 2005, foi estimado que a *Web* superficial possuísse, no mínimo, 11.5 bilhões de páginas (GULLI; SIGNORINI, 2005), e o buscador

mais popular, o Google, tivesse um índice com mais de 8 bilhões de páginas (GULLI; SIGNORINI, 2005).

Outra porção da *Web*, conhecida como **Web profunda** (*Deep Web*, *Hidden Web* ou *Invisible Web*) (CHANG et al., 2003), possui seu conteúdo escondido atrás de interfaces de consulta, comumente formulários HTML (RAGHAVAN; GARCIA-MOLINA, 2001). Em julho de 2000, estimou-se que a *Web* profunda era 400 a 550 vezes maior que a *Web* superficial (BERGMAN, 2001). Os documentos que fazem parte da *Web* profunda são acessados apenas através das páginas geradas dinamicamente, em resposta às consultas dos usuários.

É possível explorar a *Web* profunda através dos **buscadores verticais**, sites de busca da *Web* que possuem coleções de documentos que abordam um tópico específico, constituindo verdadeiras **bases de dados especializadas** (GOSHME, 2006). Sites assim costumam ter informação de maior qualidade, a exemplo do HealthLine (2007), que recupera apenas documentos sobre saúde, e seu conteúdo faz parte da *Web* profunda. Os pontos de entrada para estas bases de dados especializadas (buscadores verticais) são as interfaces de consulta (formulários de busca), encontradas na *Web* superficial. Os *crawlers* dos tradicionais engenhos de busca seguem os *hyperlinks* da *Web* superficial, ignoram os formulários HTML e não adentram estas bases, deixando uma gigantesca quantidade de informação escapar. É possível recuperar as informações escondidas da *Web* profunda, mas antes é preciso localizar as interfaces de consulta e então aprender a interagir com elas, para que seja possível extrair seu conteúdo.

Visitar todas as páginas da *Web* superficial à procura de todas estas interfaces de busca seria o ideal, mas é uma tarefa praticamente impossível. A *Web* está constantemente em mudança: novos conteúdos são adicionados enquanto os antigos são removidos e modificados (BARBOSA; FREIRE, 2005), dificultando o processo de detecção automática destas interfaces. Não são todas as páginas HTML que possuem interfaces de consulta, estudos de 2003 estimam que existem 307.000 sites cujo conteúdo faz parte da *Web* profunda, com 4.2 interfaces de consulta por site (CHANG et al., 2003). Ou seja, bilhões de páginas devem ser visitadas à procura de milhares de interfaces de consulta, "é como procurar uma agulha no palheiro"(BARBOSA; FREIRE, 2005).

Não são todos os formulários HTML presentes nas páginas *Web* que representam interfaces de busca, é preciso dintingui-los dentre os milhões existentes. Para descobrir automaticamente se um formulário é de busca é preciso agir como um ser humano: preencher seus campos e submetê-lo, analisando o resultado devolvido pelo servidor destino. Uma

outra opção seria construir uma heurística, capaz de inferir se um formulário é ou não de busca analisando apenas o código HTML formulário e da página que o contém, não submetendo uma requisição sequer ao servidor destino.

Neste trabalho, será apresentada uma ferramenta que navega pela *Web* superficial procurando as entradas para a *Web* profunda focado no idioma inglês. Esta ferramenta será capaz de visitar links de páginas HTML, coletar os formulários HTML e distinguir os formulários de busca dos demais formulários. Também será apresentada a estratégia de navegação na *Web* mostrando as técnicas usadas durante a coleta e seleção dos links a serem visitados, visando encontrar em menos tempo as entradas para as bases de dados da *Web* profunda.

1.1 Objetivos

GERAL:

- Propor uma arquitetura de um *crawler* focado em localizar automaticamente as bases de dados da *Web* profunda.

ESPECÍFICO:

- Construir um classificador capaz de distinguir as interfaces de consulta dos demais formulários HTML.
- Desenvolver uma solução para coletar os links que apresentam maior possibilidade de levar à bases de dados da *Web* profunda.
- Desenvolver uma ferramenta capaz de navegar pela *Web* superficial procurando as entradas para a *Web* profunda.
- Integrar a solução desenvovida na máquina de busca para *Web* profunda Goshme, adicionando as novas bases de dados encontradas.

1.2 Organização da Monografia

Esta monografia é constituída de cinco capítulos dos quais este é o primeiro. No Capítulo 2 serão apresentados os principais estudos na área de recuperação da informação na *Web* profunda que contribuíram no desenvolvimento deste trabalho, além da arquitetura

e funcionamento de uma máquina de busca para a *Web* profunda. O Capítulo 3 apresenta a solução desenvolvida: como foi abordado o problema, quais as estratégias adotadas para sua resolução, mostrando os principais erros encontrados e a arquitetura da ferramenta desenvolvida. O Capítulo 4 apresenta o estudo de caso da implantação da solução em uma máquina de busca real para a *Web* profunda, o ambiente computacional utilizado nos experimentos e uma avaliação de desempenho da ferramenta implementada. O Capítulo 5 apresenta as principais conclusões sobre o trabalho e os próximos trabalhos que deverão ser realizados.

2 Trabalhos Relacionados

Encontrar uma informação específica na *Web* tem se tornado cada vez mais difícil: a diversidade, grande volume e alta taxa de crescimento dificultam a recuperação da informação relevante. Logo, máquinas de busca surgiram com o intuito de auxiliar o usuário a encontrar mais rapidamente as informações que o interessam. Elas visitam os documentos na *Web* e armazenam localmente informações que auxiliem na recuperação destes documentos.

As máquinas de busca convencionais recuperam os documentos da *Web* superficial (COPE; CRASWELL; HAWKING, 2003). Já as **máquinas de busca para a Web profunda** localizam conjuntos de documentos escondidos atrás de formulários de consulta, sendo necessário descobrir onde estas bases de dados estão localizadas na *Web* e depois interagir com elas para que possa extrair seu conteúdo.

A Seção 2.1 apresenta soluções de arquitetura utilizadas na construção de máquinas de busca para a *Web* profunda, explicando seus componentes e as principais diferenças existentes para uma máquina de busca convencional. Em seguida, a Seção 2.2 apresenta alguns estudos acerca da localização das bases dados da *Web* profunda e a Seção 3.3, por fim, as técnicas de navegação e coleta de links utilizadas durante a localização destas bases.

2.1 Máquinas de Busca para Web Profunda

A arquitetura típica de uma máquina de busca convencional é formada por: *crawlers*, *indexador*, *índice invertido*, *processador de consultas*, e *interface de consulta* para os usuários (SARAIVA, 2001; BAEZA-YATES; RIBEIRO-NETO, 1999) (ver Figura 1). Os *crawlers* cruzam a *Web* coletando páginas para serem armazenadas localmente. Após realizado o *download* das páginas, o *indexador* analisa as páginas extraindo os termos encontrados e relacionando-os com esta, gerando, ao final, o *índice invertido*, um tipo de índice que possui todos os termos conhecidos, os documentos nos quais foram encontrados e evidências

encontradas nos termos (e.g. negrito, itálico, título do documento) para cálculo de *ranking* (i.e. ordenar os documentos por relevância). O *índice* será utilizado pelo *processador de consultas* para recuperar os documentos que possuem as palavras-chave buscadas pelo usuário na *interface de consulta*, realizando todos os cálculos necessários para eleger os melhores documentos, de acordo com as evidências coletadas.

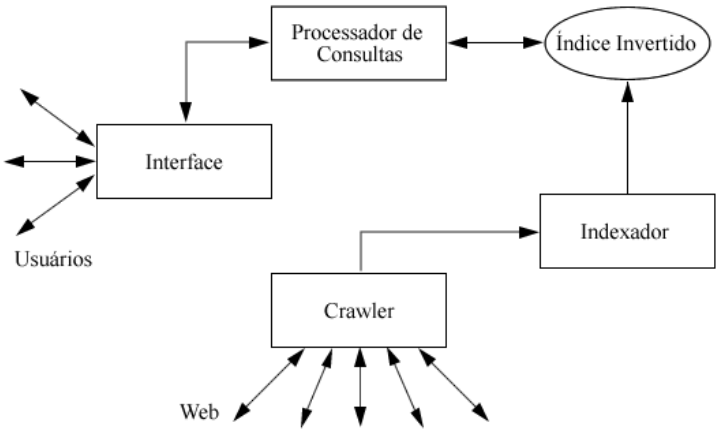


Figura 1: Arquitetura típica de uma máquina de busca. Adaptado de Baeza-Yates e Ribeiro-Neto (1999)

Máquinas de busca para a *Web* profunda também fazem uso de *crawlers*, *indexador*, *índice invertido*, *processador de consultas* e *interface de consulta*. A diferença é que seus *crawlers* são mais elaborados, capazes de identificar os formulários de busca nas páginas HTML e aprender a interagir com os mesmos, extraindo o conteúdo escondido. Os documentos retornados nas páginas de respostas serão utilizados pelo *indexador* para gerar o índice, que informará ao *processador de consultas* quais os documentos escondidos possuem a busca do usuário. Este processo não ocorre necessariamente nesta ordem e podem apresentar outras etapas.

O HiWE (*Hidden Web Exposer*) (RAGHAVAN; GARCIA-MOLINA, 2001), foi uma das primeiras iniciativas na descoberta e recuperação de conteúdo da *Web* profunda. Uma arquitetura genérica foi proposta para que o *crawler* conseguisse interagir com os formulários de busca de forma similar a um ser humano, testando automaticamente as combinações de valores e submetendo-as a consulta, caracterizando uma navegação autônoma. À medida que os formulários eram encontrados, eram submetidos à consulta e seu índice era criado a partir dos resultados das páginas obtidas. O problema é muitas consultas eram submetidas em todos os formulários encontrados, e muitos sequer eram de busca, levando muito tempo até se descobrir.

Em outro trabalho, o SmartCrawl (FONTES; SILVA, 2004), o *crawler* navega pela *Web* procurando páginas que contém formulários HTML, comprimindo e armazenando estas páginas para futura análise. Posteriormente, o componente de indexação descomprime estas páginas, detecta as informações de cada formulário e, utilizando um conjunto de palavras previamente selecionadas, recupera o conteúdo escondido, realizando a indexação simultaneamente. Se um mesmo formulário for encontrado em páginas distintas, todas elas serão indexadas, mas somente uma representará o formulário.

Em Goshme (2006) é apresentado o funcionamento da máquina de busca para *Web* profunda Goshme, que utiliza o *Engine Finder* para localizar na *Web* superficial as bases de dados especializadas e em seguida aprende a interagir com os buscadores e acessar seus documentos. O *Content Extractor* extrai das páginas de resposta apenas o resultado, separando os links dos documentos que serão visitados e indexados, do restante da apresentação da página. Diferente dos buscadores convencionais, o Goshme não traz para seu usuário os melhores documentos. No momento da consulta são eleitos internamente as bases de dados mais relevantes à consulta do usuário, cada uma contendo coleções de documentos. Como resultado, o Goshme retorna para seus usuários uma página de resposta com links que os levam diretamente à página de resposta do buscador vertical, e não à página que contém o formulário de busca, evitando que o usuário necessite preencher novamente os campos e realizar uma nova consulta, além de resolver problemas como a complexidade de determinados formulários (e.g. múltiplos campos de texto) e os diferentes métodos de submissão (GET ou POST). A nova arquitetura do Goshme, incorporando as contribuições deste trabalho, é apresenta na Seção 4.1.

2.2 Identificando Bases de Dados na Web profunda

Para ter acesso ao conteúdo escondido nas bases de dados da *Web* profunda, é necessário identificar seus pontos de entrada, localizados na *Web* superficial (Figura 2). Estas bases de dados estão esparsamente distribuídas na *Web* superficial, portanto é preciso uma estratégia eficiente para localizar os formulários de busca, que servem como pontos de entrada para estas bases de dados (BARBOSA; FREIRE, 2007).

Um estudo feito por Chang et al. (2003) sobre a localização das entradas para as bases de dados em um site da *Web* profunda, observando-se a quantidade de interfaces de consulta, bases de dados distintas e a profundidade (quantidade de saltos realizados entre a página corrente e a raiz do site) das páginas em que foram encontradas. Inicialmente,

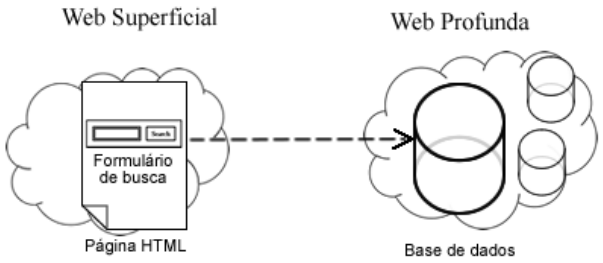


Figura 2: Exemplo de formulário de busca na *Web* superficial acessando base de dados da *Web* profunda

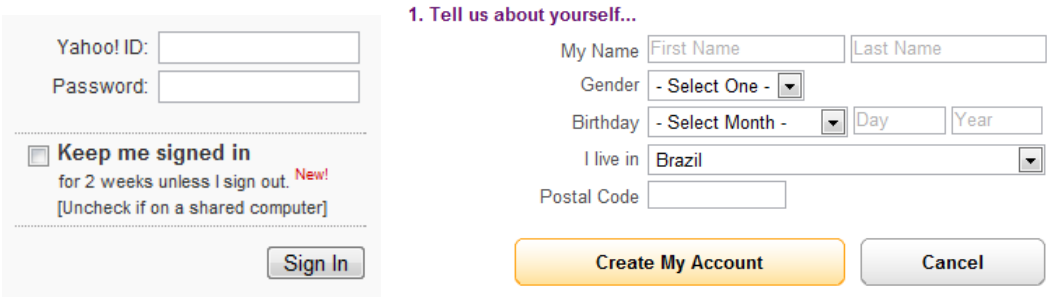
analisando-se um conjunto randômico de 100.000 IPs, foram encontrados apenas 281 servidores *Web*, os quais foram visitados exaustivamente até a profundidade 10. Ao final, 24 sites de acesso *Web* profunda foram detectados, um total de 129 interfaces de consulta que davam acesso a 34 bases de dados distintas, com 94% destas bases localizadas até a profundidade 3. Um teste mais aprimorado foi realizado: foram visitadas todas as páginas até a profundidade 3 de 2.256 servidores *Web* e encontrou 126 sites da *Web* profunda com 406 interfaces de busca e 190 bases de dados. Ao final do trabalho, concluiu-se que existem 1.5 bases de dados para cada site e 2.8 interfaces de consulta para cada uma destas bases.

Estas interfaces são, em sua maioria, formulários HTML (a exemplo do apresentado na Figura 3), que sempre apresentam pelo menos um campo texto (para que usuário possa entrar com a consulta) e geralmente apresentam botões de submissão, que podem ser do tipo *submit*, *image* ou *button*. No entanto, alguns formulários não correspondem a entradas para a *Web* profunda, por exemplo, formulários de cadastro de usuário, *login*, inscrição em listas de e-mail, *newsletter*, entre outros (BARBOSA; FREIRE, 2005) (ver Figura 4).


```
<form name="form1" action="search">
 <input name="query" value="" type="text" />
 <input name="submit" value="Search" type="submit" />
</form>
```

Figura 3: Formulário de busca típico e seu código HTML

Na detecção de formulários de busca, são utilizados processos automáticos, e, consequentemente, das bases de dados *online*. O *Form-Focused Crawler* (FFC) de Barbosa e Freire (2005), otimizado para localizar formulários de busca, utilizou um classificador de formulários, o *Form Classifier*, com algoritmo de árvore de decisão C4.5. Este classificador



(a) Formulário para autenticação de usuários (YAHOO, 2008)

(b) Formulário para cadastro de usuários. Adaptado de Yahoo (2008)

Figura 4: Exemplos de formulários HTML que não devem ser classificados como buscadores

deveria descartar formulários HTML considerados não-buscadores, aprendendo com seus padrões estruturais: número de *tags* do tipo *hidden*, *radios*, de botões, campos de texto, método de submissão utilizado (POST ou GET), etc.; considerando estas características como bons indicadores para determinar se o formulário é ou não de consulta. Mas os resultados não foram satisfatórios, apenas 16% dos formulários recuperados pelo FFC eram de fato relevantes, recuperando grande número de *login*, inscrição em listas de e-mails, acesso a e-mail e outros. Quando as páginas possuíam bases de diferentes domínios, este percentual caiu para 6.5%, recuperando formulários de aluguéis de carro e reserva em hotéis, nos sites que eram sobre pesquisa de passagem aérea, por exemplo.

Barbosa e Freire (2007) apresentaram o *Hierarchical Form Identification* (HIFI), uma nova estratégia para obter com maior precisão os formulários de busca que representam as bases de dados de um domínio específico, utilizando uma combinação de dois classificadores. O primeiro classificador, o *Generic Form Classifier* (GFC), funciona como um filtro, identificando os formulários de busca entre todos os formulários encontrados na *Web*, com base nas características estruturais do formulário e utilizando o classificador C4.5. O segundo classificador, o *Domain-Specific Form Classifier* (DSFC), identifica, entre os formulários de busca, formulários que são relevantes a um determinado domínio, analisando o conteúdo textual do formulário através de um classificador SVM (*Support Vector Machine*) treinado por SMO (*Sequential Minimal Optimization*). Dentre os domínios analisados (passagem aérea, carros, livros, hotel, trabalho, cinema, música e aluguel), o

GFC apresentou *recall*[1] variando entre 90% a 98% e especificidade[2] de 18% a 78%.

2.3 Seleção de Links

Visitar todo o conteúdo da *Web* superficial é praticamente impossível devido ao seu crescimento e geração de conteúdo dinâmico (DILIGENTI et al., 2000). Este desafio esbarra numa infinidade de problemas, como a capacidade de gerenciar tamanha quantidade de informação e nas limitações computacionais existentes de largura de banda e armazenamento necessários para guardar centenas de terabytes de informação. A abordagem menos eficiente seria visitar e coletar todos os links encontrados infinitamente. Esta abordagem exaustiva consome uma grande quantidade de armazenamento e largura de banda de Internet, à medida que milhares de documentos irrelevantes são recuperados.

A implementação de estratégias de navegação nos *crawlers* minimizam o desperdício computacional. *Focused Crawlers* buscam encontrar apenas a porção da web que pertencem a um tópico específico, recuperando o maior número de páginas relevantes à medida que a menor quantidade de documentos irrelevantes são recuperados(DILIGENTI et al., 2000). Diligenti et al. (2000) propõe o *Context Focused Crawler* (CFC), que utiliza um classificador de páginas com algoritmo Naive Bayes, para guiar seu *crawler* durante a busca. Um conjunto inicial de páginas HTML de diferentes áreas de interesse (e.g. saúde, direito, entretenimento) é utilizado para definir as classes do classificador. Quando o *crawler* entra em execução, as páginas visitadas são submetidas ao classificador a fim de descobrir a qual destas classes pertence. Serão coletados apenas os links das páginas pertencentes à mesma classe da página origem (i.e. página onde o link foi encontrado), atribuindo-se um escore. De acordo com o escore atribuído, os links coletados são inseridos em diferentes filas e os próximos links a serem visitados pelo *crawler* obedecem a prioridade das filas, aborgem conhecida como *best-first*.

O *Form-Focused Crawler*, proposto por Barbosa e Freire (2005), utilizou um classificador de páginas, o *Page Classifier*, com algoritmo Naive Bayes, treinado a partir da taxonomia do Dmoz (2007), para descobrir a probabilidade de uma página qualquer pertencer ao tópico em questão ao longo da taxonomia (e.g. arte, cinema, viagens). Este mesmo trabalho utilizava um classificador para *links*, o *Link Classifier*, também usando o algoritmo Naive Bayes, treinado para identificar links que potencialmente levam a páginas

[1] *Recall*=TP/(TP + FN), onde TP corresponde à classificação correta e FN à classificação incorreta da classe desejada.

[2] *Specificity*=TN/(TN + FP), onde TN corresponde à classificação correta e FN à classificação incorreta da classe indesejada.

que possuem formulários de busca, mesmo que o benefício não seja imediato, utilizando o texto-âncora, URL e texto na proximidade da URL, como potenciais características dos links.

3 Solução Proposta

Durante este capítulo serão apresentadas as principais técnicas desenvolvidas que foram utilizadas na solução proposta. Inicialmente é apresentada uma comparação entre diferentes classificadores para a identificação das interfaces de consulta, ou formulários de busca. Em seguida é feita uma discussão sobre a existência de diferentes interfaces de consulta acessando uma mesma base de dados *online*, apresentando algumas abordagens que podem ajudar nesta detecção.

Na seção seguinte, Seção 3.3, são mostrados os critérios para seleção de *links* ao longo das páginas HTML, no momento que são visitadas pelos *crawlers* e na Seção 3.4 uma outra forma de coletar novas URLs, externas ao processo de *crawler*.

Por fim, nas duas seções finais a solução proposta é apresentadada, uma espécie de *crawler* focado na localização das bases de dados *online*, mostrando onde serão utilizadas as técnicas discutidas no início do capítulo. A arquitetura do *crawler* e a modelagem do banco de dados são apresentadadas nas Seções 3.5 e 3.6, respectivamente.

3.1 Localizando Interfaces de Consulta

As interfaces de consulta podem ser identificadas através de duas técnicas: pré-query e pós-query (COPE; CRASWELL; HAWKING, 2003). A segunda tem pior desempenho, pois envolve interação com o formulário de consulta: aprender interagir com o mesmo a fim de descobrir consultas válidas, analisando as páginas de repostas e certificando-se que se trata de um formulário de busca. O primeiro tem melhor desempenho por analisar apenas o conteúdo presente nas páginas HTML que possuem o formulário HTML, não necessitando recuperar as páginas de resposta às consultas. No entanto o pré-query é menos preciso que o pós-query por eleger erroneamente formulários não buscadores. Ainda assim, preferimos utilizar a técnica pré-query por possibilitar localizar rapidamente um maior número de formulários buscadores.

Formulários HTML variam muito quanto sua estrutura, não existindo um padrão que defina quais são interfaces de consulta. A partir de algumas evidências encontradas ao longo de sua estrutura, com auxílio de classificadores, é possível descobrir quais são formulários de busca, a exemplo de Barbosa e Freire (2005) e Cope, Craswell e Hawking (2003) que fizeram uso do algoritmo C4.5 (QUINLAN, 1993) para geração automática de árvores de decisão. No entanto, estes classificadores não identificam todos os formulários de busca, eles tentam ajustar seu algoritmo a obter maior precisão durante classificação.

O primeiro procedimento que foi feito durante este trabalho foi avaliar uma solução já proposta em trabalhos semelhantes. O trabalho escolhido foi o de Cope, Craswell e Hawking (2003), que utilizou o algoritmo de árvore de decisão C4.5 com as evidências elencadas em seu trabalho. Como não possuíamos a coleção de teste para a qual a árvore de decisão apresentada em Cope, Craswell e Hawking (2003) foi gerada, coletamos um novo conjunto de formulários e geramos a árvore de decisão novamente, utilizando as características originais propostas pelos autores. Estas características são usadas para pré-processar cada formulário, estruturando um vetor no qual cada posição representa uma característica (i.e. numérico, booleano, etc.).

Visando recuperar URLs de páginas que possuíssem formulários HTML, foram realizadas consultas em engenhos de busca conhecidos, Google e Yahoo, com as palavras *form*, *login* e *search*, escolhendo aleatoriamente 1.495 URLs que possuíam formulários HTML. Ao final, foram contabilizados 3.015 formulários com código HTML distinto, dos quais 2.446 foram classificados manualmente, separando-se um conjunto de 1.800 formulários para realização da classificação: 900 buscadores e 900 não-buscadores.

O conjunto de 1.800 formulários HTML conhecidos foi dividido em duas porções separadas: 66% (1.188) para treino e 34% (612) para teste, distribuídas igualmente (594 treino e 306 teste) entre os buscadores e não-buscadores. Com o apoio da ferramenta Weka (WEKA, 2007) e utilizando o classificador J48, que implementa a algoritmo C4.5 de árvore de decisão, uma nova árvore (sem poda) foi gerada (ver Anexo B), a partir do conjunto de treino, com as evidências levantadas por Cope, Craswell e Hawking (2003). A árvore gerada classificou corretamente (ver Equação 3.3) 92.17% do conjunto de treino e 90.03% do teste. As matrizes confusão para o conjunto de treino e teste são apresentadas na Tabela 1 e Tabela 2, respectivamente.

Tabela 1: Matriz confusão do treino da nova árvore C4.5 com características de Cope, Craswell e Hawking (2003)

Classes	Predita Buscador	Predita Não-Buscador
Buscador	576	18
Não-buscador	75	519

Tabela 2: Matriz confusão do teste da nova árvore C4.5 com características de Cope, Craswell e Hawking (2003)

Classes	Predita Buscador	Predita Não-Buscador
Buscador	288	18
Não-buscador	43	263

A matriz confusão é utlizada para apresentar os resultados obtidos durante uma classificação. Uma matriz confusão para duas classes (C+ e C-) é apresentada na Tabela 3, onde as linhas correspondem às classes reais do conjunto e as colunas às classes inferidas pelo classificador.

Tabela 3: Matriz confusão para duas classes (REZENDE, 2005)

Classes	Predita C+	Predita C-
Verdadeira C+	Verdadeiros positivos (TP)	Falsos negativos (FN)
Verdadeira C-	Falsos positivos (FP)	Verdadeiros positivos (TN)

Para fazer a análise dos resultados, foram utilizados os seguintes índices:

$$sens(h) = \frac{TP}{TP + FN} \tag{3.1}$$

$$spec(h) = \frac{TN}{TN + FN} \tag{3.2}$$

$$tacc(h) = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.3}$$

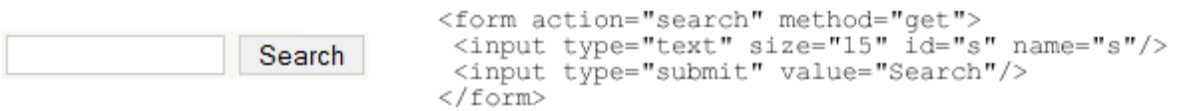
onde sens(h) seria a sensibilidade positiva ou *recall*, spec(h) a confiabilidade negativa e tacc(h) a precisão total.

Após a análise dos resultados, verificou-se que muitos formulários haviam sido classificados de forma incorreta e que novas características poderiam ser acrescentadas a fim de melhorar a precisão do classificador. Estas mesmas URLs foram visitadas e, a fim coletar novas características, os seguintes elementos de seus formulários HTML foram extraídos:

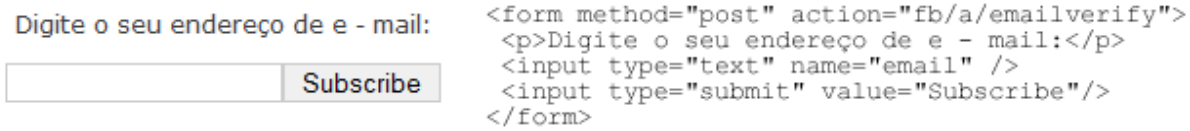
atributos do formulário (*name*, *id* e *action*), das *tags input* (*name*, *id*, *value*, *alt*) e a quantidade destas *tags* que constituem o formulário HTML.

Em seguida, os dados coletados foram separados, contabilizados e analisados individualmente, tentando-se descobrir características que potencialmente contribuiriam na classificação. A quantidade de cada elemento (e.g. campos de texto simples ou múltiplas linhas, botões de submissão, imagem) e a deteção de palavras-chave (e.g. *search*, *find*, *seek*, *query*) foram fundamentais na elaboração das novas características que ajudariam na caracterização das interfaces de consulta.

Alguns formulários HTML não-buscadores apresentavam estrutura bastante semelhante aos buscadores (ver Figura 5) e foram classificados incorretamente pela árvore de decisão. Observando novas evidências, as palavras-chave *email*, *subscribe*, *login*, *join*, *sign in*, *translate*; e alguns elementos estruturais (e.g. campo password), descaracterizavam formulários de busca quando presentes.



(a) Formulário buscador



(b) Formulário não-buscador

Figura 5: Similaridade entre formulários HTML

O conjunto destas novas características (ver Anexo A) foi usado na geração de uma nova árvore de decisão com o algoritmo C4.5 (ver Anexo C), seguindo o mesmo procedimento anterior, com a utilização do Weka e com os mesmos conjuntos de treino e teste. A Tabela 4 apresenta a matriz confusão do treino, que acertou 95.54% da classificação, e a Tabela 5 do teste desta nova árvore, com 94.61% de acerto.

Tabela 4: Matriz confusão do treino da árvore C4.5 com novas características

Classes	Predita Buscador	Predita Não-Buscador
Buscador	581	13
Não-buscador	40	554

Tabela 5: Matriz confusão do teste da árvore C4.5 com novas características

Classes	Predita Buscador	Predita Não-Buscador
Buscador	295	11
Não-buscador	22	284

A quantidade insuficente de características contribuem existência de dados incosistentes, fazendo com que os classificadores acabem errando. Ou seja, alguns formulários de busca são identificados como não-buscadores enquanto que outros formulários não-buscadores acabam sendo classificados como buscadores.

Neste momento, é preferível coletar o maior número de formulários buscadores à medida que são eliminados a maior quantidade de não-buscadores. Classificar incorretamente um formulário buscador como não-buscador significa perder bases de dados. Uma terceira solução, uma árvore de decisão construída manualmente, foi desenvolvida a fim de minimizar esta perda, utilizando as mesmas características usadas previamente. Esta solução foi construída executando diversas vezes a classificação para conjunto de treino, realizando os ajustes necessários até conseguir resultados aceitáveis, mas de forma a priorizar aqueles formulários que com certeza são buscadores e em seguida eliminar os não-buscadores, elegendo como buscadores os casos duvidosos. A solução final (ver Anexo D) apresentou acerto de 94.28% no conjunto de treino e 95.59% quando aplicado no conjunto de teste. Nas Tabelas 6 e 7 do treino e teste, nesta ordem, é possível observar que poucos formulários buscadores foram classificados como não-buscadores, falsos-negativos (ver Tabela 3). Isto foi possível graças à alta *confiabilidade negativa* (ver Equação 3.2) conseguida de 99.80% durante o treino e repetida com 98.27% durante o teste.

Tabela 6: Matriz confusão da árvore de decisão construída manualmente no conjunto de treino

Classes	Predita Buscador	Predita Não-Buscador
Buscador	593	1
Não-buscador	84	510

Tabela 7: Matriz confusão do teste da árvore de decisão construída manualmente

Classes	Predita Buscador	Predita Não-Buscador
Buscador	301	5
Não-buscador	22	284

Os resultados finais, com a *sensibilidade positiva* (Equação 3.1), *confiabilidade negativa* (Equação 3.2) e *precisão total* (Equação 3.3); acompanhados dos classificadores utilizados, são apresentados nas Tabelas 8 e 9 para o treino e teste, respectivamente.

Tabela 8: Resultados finais da classificação no conjunto de treino

Classificador	*Recall*	Conf. Negativa	Precisão Total
C4.5 com características de Cope (2003)	96.97%	96.65%	92.17%
C4.5 com novas características	97.81%	97.71%	95.54%
Árvore manual com novas características	99.83%	99.80%	94.28%

Tabela 9: Resultados finais da classificação no conjunto de teste

Classificador	*Recall*	Conf. Negativa	Precisão Total
C4.5 com características de Cope (2003)	94.12%	93.59%	90.03%
C4.5 com novas características	96.41%	96.27%	94.61%
Árvore manual com novas características	98.37%	98.27%	95.59%

Apesar do ganho pequeno na classificação, espera-se que quando esta solução for aplicada na Web uma maior quantidade de bases de dados sejam identificadas. Por fim, a árvore construída manualmente foi escolhida para ser usada para cumprir os objetivos deste trabalho, mas pretende-se voltar a estudar melhorias para serem aplicadas durante a classificação.

Alguns erros externos à classificação ocorreram durante a detecção das interfaces, os que mais se destacaram foram: erros de sintaxe do HTML e formulários gerados JavaScript. O primeiro é um erro comum, causado por falhas de codificação do desenvolvedor e que dificulta a construção da árvore DOM[1] pelo parser HTML, perdendo atributos ou até mesmo inviabilizando a extração de todo o formulário HTML. O segundo, menos comum, é quando o formulário de busca é gerado apenas no browser pela interpretação do código JavaScript, fugindo ao escopo deste trabalho.

3.2 Identificando Bases de Dados distintas

Considerando dois formulários HTML buscadores idênticos, mesmo *action* (destino dos dados do formulário) e parâmetros iguais, localizados na mesma página ou em páginas diferentes, sempre levarão à mesma base. Isto acontece porque a consulta será enviada ao mesmo processador de consulta de um buscador vertical e, com os mesmos parâmetros, sempre retornarão resultados idênticos. De acordo com a Figura 6, em um cenário bastante comum, uma única página HTML (*P1*), poderá ter formulários idênticos, e este mesmo formulário HTML (*F1*), poderá aparecer em outras páginas na *Web* (*P2*), sempre levando

[1]HTML DOM (Document Object Model) é a padronização de um modelo para acesso e manipulação a documentos HTML, apresentados em estrutura de árvore composta por elementos, atributos e textos. Maiores informações (W3C, 2007)

à mesma base (*B1*).

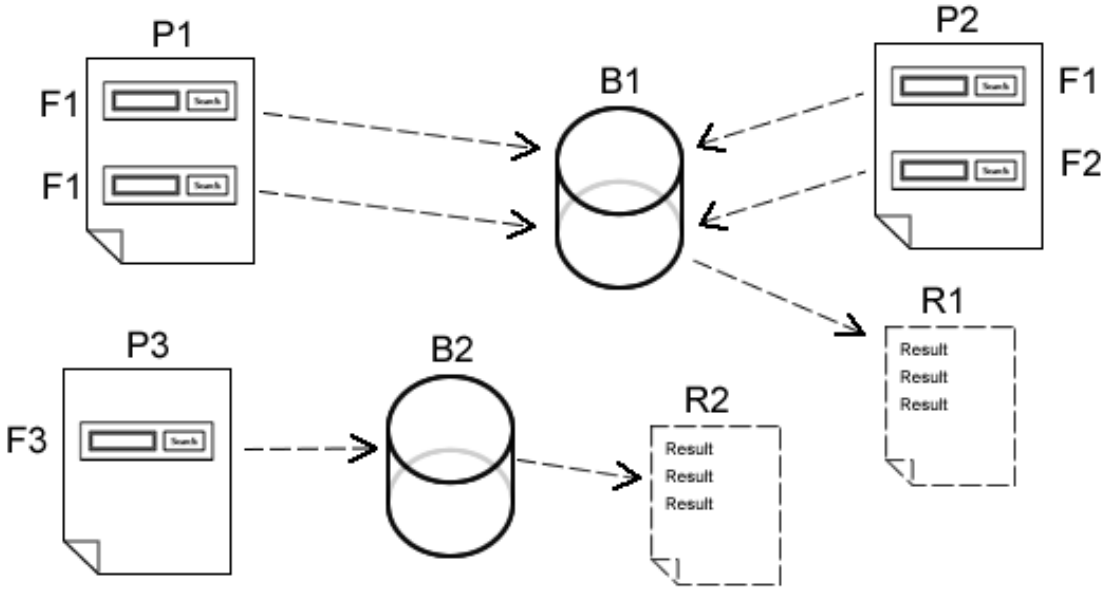


Figura 6: Acessando bases de dados distintas

Formulários de busca diferentes, no entanto, não levam necessariamente à bases distintas. É possível encontrar em páginas diferentes, ou na mesma página, diferentes formulários de busca que recuperam a informação da mesma base. Este exemplo é mostrado na Figura 6, onde a página HTML *P2* possui dois formulários diferentes, *F1* e *F2*, ambos dão acesso à mesma base, *B1*, e devolvem a mesma página de resposta *R1*.

Formulários diferentes representam a mesma base se ambos apresentam sempre os mesmos resultados para qualquer consulta. Tomando novamente como exemplo a Figura 6, o ideal é coletar apenas um formulário para cada base distinta, neste caso, os formulários *F1* e *F3* garantem acesso às bases de dados *B1* e *B2*, respectivamente. Nesta arquitetura, a extração e indexação dos documentos não ocorrem simultaneamente à identificação dos formulários de busca. Estes formulários são armazenados e os documentos extraídos posteriormente. Pelo fato de existirem na *Web* bases de dados com milhares de documentos, recuperar novamente os documentos da mesma base representa um desperdício computacional enorme. Assim, é necessário saber se a base já é conhecida à medida que os formulários de busca são identificados.

Uma solução bastante eficiente seria criar e armazenar uma assinatura das páginas de resposta para uma consulta válida - possui resultados, utilizada como identificador da base de dados. Este identificador poderia ser um *hash* do código HTML da página de resposta, possibilitando verificar rapidamente a existência da base. No entanto, um

mesmo formulário pode gerar, para a mesma consulta, páginas de resposta com códigos HTML sutilmente diferentes: alguns elementos mudam sempre que a página de resposta é construída, links de propagandas, por exemplo. Sites distintos que acessam a mesma base geralmente apresentam *layouts* diferentes mas resultados idênticos. Isolar os resultados das páginas de resposta do restante do *layout* é uma tarefa particular a cada site e necessita do download de várias páginas de resposta do mesmo formulário para realização desta análise, o que aumenta siginificativamente o tempo e custo de processamento.

Outra solução seria comparar o grau de similaridade entre as páginas de resposta para uma consulta padrão, estabelecendo um valor limiar para serem consideradas semelhantes. Algoritmos de comparação de similaridade entre textos exigem alto poder de processamento, o que torna esta solução inviável quando precisa-se sempre comparar a página de resposta do formulário corrente com cada uma das outras milhares de páginas de respostas já conhecidas. Outro ponto crítico desta solução é o grande espaço para armazenamento necessário para guardar estas milhares de páginas.

Apesar das páginas de resposta sofrerem mudanças a cada consulta, observou-se que alguns elementos permanecem estáticos (e.g. links dos resultados). A solução desenvolvida gera um identificador da página de resposta utilizando os elementos estáticos. A partir de um conjunto de páginas de resposta de um formulário, os elementos que são comuns entre elas são detectados e isolados. Logo em seguida, o código HTML destes elementos é utilizado para criar um código *hash*, a assinatura da página de resposta, citada anteriormente como identificador da base de dados. Portanto, para descobrir as bases de dados distintas, é preciso obter algumas páginas de resposta à consulta nos formulários encontrados, o que representa um aumento significativo no tempo total de processamento.

Esta solução, entretanto, nem sempre garante unicidade do identificador, podendo falhar em alguns casos. Quando o resultado da base muda constantemente (e.g. bases de dados de notícias) ou o resultado é alterado de acordo com o perfil do usuário, códigos *hashs* distintos podem ser gerados para a mesma base de dados.

3.3 Seleção de Links

O processo de *crawler* é iniciado a partir de um conjunto inicial de URLs, conhecidas como sementes, que devem ser visitadas para encontrar os formulários HTML. Mas para que o *crawler* continue sempre em execução, é preciso inserir novas URLs entre as sementes, senão quando todas as URLs forem visitadas o processo ficaria ocioso. Esta

inserção pode ser feita através do próprio *crawler*, extraindo as URLs encontradas à medida que as páginas *Web* são visitadas. Entretanto, em uma única página HTML pode-se encontrar dezenas ou centenas de outras URLs distintas, que geralmente pertencem ao mesmo domínio. Estas apontam para outras páginas que podem conter outras dezenas de URLs, novamente pertencentes ao mesmo domínio, repetindo este processo até que todas as URLs do domínio sejam conhecidas e/ou visitadas.

Para localizar todos os formulários de busca da *Web*, o ideal seria coletar e visitar todas as URLs que fossem encontradas ao longo das páginas HTML. No entanto, se todas as URLs fossem coletadas, muitas levariam um longo tempo aguardando para serem visitadas e, quando fossem selecionadas pelo *crawler*, poderiam não mais existir (BAEZA-YATES; RIBEIRO-NETO, 1999).

As entradas para as bases de dados estão localizadas próximas à raiz do site (CHANG et al., 2003), ou seja, não seria necessário continuar coletando as URLs encontradas nas páginas internas distantes da raiz do site, onde os formulários de busca costumam se repetir (i.e. facilitar a navegação) e outras sequer possuem formulários HTML.

Outro problema seria a capacidade de *hardware* e banda de Internet necessárias para atender à tamanha demanda. Em alguns testes, utilizando um subconjunto de 1.000 URLs dentre aquelas usadas anteriormente durante a classificação de formulários, foram coletados todos os links encontrados nestas páginas. Ao final, 76.495 URLs foram originadas de 859, sendo que as outras 141 não tinham *hyperlinks* ou apresentaram erro durante a conexão. Sabe-se que este crescimento é geométrico e, obdecendo esta razão (q = 89,05), quando forem visitadas as 76.495 URLs, cerca de 6.811.973 novas URLs seriam coletadas. Necessitar-se-ia de uma estratégia de seleção para limitar a quantidade de URLs coletadas, existindo um balanceamento entre as novas URLs produzidas e as que seriam consumidas pelo *crawler*.

A solução escolhida foi coletar apenas as URLs que possuíam em seu texto-âncora[2] as palavras-chave *"search"* e *"find"*, palavras comumente encontradas nos links para as páginas de busca de um site, aplicando-se filtros para elminação de algumas URLs.

O primeiro filtro foi aplicado para eliminar URLs muito grandes, julgadas potencialmente ruins. O número de 250 caracteres apresentou-se suficientemente grande representar as URLs desejadas, descartando as demais.

Links para notícias ou para itens em sites de compra, são exemplos de links indesejados

[2]Texto-âncora refere-se ao texto localizado entre as *tags* `<A>` e `</A>` que formam os *hyperlinks*. Maiores informações (W3C, 2007)

durante a seleção. Estes links costumam apresentar, nos parâmetros de suas URLs, blocos de números, usados para identificação de itens, paginação ou filtro dos resultados por datas. No entanto, alguns casos observados apresentavam números apenas para designar a seção do site (e.g. 1=audio, 2=sexo, 3=educação) e não deveriam ser descartados. O número 2, suficientemente pequeno, foi escolhido como limitador de caracteres numéricos, descartando todos links que possuíssem mais que dois caracteres numéricos nos parâmetros de sua URLs.

Um último filtro foi aplicado para descartar as URLs que possuíssem *tokens* muito grandes. Os *tokens* seriam os blocos de palavras obtidos após a quebra por "/"(barra) da URL. Apesar dos *tokens* formarem o caminho para localização da página, muitos destes *tokens* grandes são códigos gerados aleatoriamente pelo site origem ou podem ser, por exemplo, seções internas de sites de compra (e.g. `.../AcessoriosParaComputadorPessoal/mouses.html`). A quantidade de 30 caracteres (número não estimado, apenas um limitador) foi escolhida, descartando as URLs que que excedessem este número em algum de seus *tokens*.

As URLs apresentadas na Tabela 10 são exemplos de URLs que seriam invalidadas seguindo as regras previamente estabelecidas. Entretanto, algumas destas referem-se a páginas que possuem formulários de busca, geralmente o mesmo encontrado na raiz do site ou nas páginas mais próximas da raiz. Assim, as URLs deveriam ser quebradas por "/"(barra) e inseridas na lista do *crawler*, priorizando no momento da visita aquelas que fossem menores. Tomando como exemplo a URL da primeira linha da Tabela 10, a listagem das novas URLs geradas é apresentada na Tabela 11. Apesar da URL original não possuir formulário de busca, quatro das URLs derivadas possuíam o mesmo formulário, escolhendo a raiz do site para representar este formulário, permitindo antecipar a inserção da raiz do site antes de ser encontrada em uma outra página HTML.

Tabela 10: Exemplo de URLs que devem ser invalidadas

URLs
`http://www.ornl.gov/sci/techresources/human_genome/publicat/02santa/index.shtml`
`http://www.reuters.com/article/politicsNews/idUSWAT00913420080314`
`http://news.bbc.co.uk/1/hi/world/asia-pacific/7296837.stm`
`http://www.amazon.com/Samsung-LNT4071F-1080p-120Hz-HDTV/dp/B000U9ZCQS/ref=mmw_multi_asin/103-1736061-0276621?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-2&pf_rd_r=1BVEFYZNQ9FW2XWW4825&pf_rd_t=101&pf_rd_p=366690301&pf_rd_i=507846`
`http://srx.main.ebayrtm.com/clk?RtmClk&lid=359642&m=32336&pi=3907`

Tabela 11: URLs derivadas a partir de uma URL invalidada

URLs
http://www.ornl.gov/sci/techresources/human_genome/publicat/02santa/index.shtml
http://www.ornl.gov/sci/techresources/human_genome/publicat/02santa
http://www.ornl.gov/sci/techresources/human_genome/publicat
http://www.ornl.gov/sci/techresources/human_genome
http://www.ornl.gov/sci/techresources
http://www.ornl.gov/sci
http://www.ornl.gov

É possível encontrar ao longo das páginas HTML, URLs que levam diretamente à página de resposta de um buscador, podendo ser invalidadas através das palavras "*query*", "*search*", "*keyword*"e "*find*"quando presentes entre os parâmetros da URL. Quebrando-se novamente estas URLs, é possível localizar a página que possuía o formulário de busca de origem. Outra URL derivada foi extraída do início da URL original até o início dos parâmetros, marcado pelo caractere '?'. A Tabela 12 mostra um exemplo da aplicação da quebra de uma URL de uma página de resposta e as derivadas produzidas, que continham o formulário onde a busca foi feita.

Tabela 12: URLs derivadas a partir de uma URL de busca invalidada

URLs
Original: http://usasearch.gov/search?input-form=simple-firstgov&v\%3Asources=firstgov-search-select&v\%3Aproject=firstgov&query=love
Derivada: http://usasearch.gov
Derivada: http://usasearch.gov/search

Após a execução do *crawler*, a seleção de URLs foi reavaliada com 76.495 URLs, e 36.596 (48%) foram identificadas como buscadoras, com o auxílio do classificador de formulários. De todas as URLs, 749 foram selecionadas pela solução utilizada, com 425 entre buscadoras, correspondendo a 58% das selecionadas ou 1.16% do total de buscadoras, não correspondendo aos resultados esperados.

3.4 Expansão na Web

Mesmo optando por selecionar apenas partes das URLs encontradas em uma página HTML, ainda assim o *crawler* pode permanecer restritro a pequenas porções da *Web*, ficando preso nestes domínios por bastante tempo. Para não restringir o domínio da *Web* apenas aos links presentes nas páginas visitadas, seria preciso fornecer ao *crawler* novos

links, providos de fontes externas. Utilizar processo manual de inserção de URLs é uma técnica comum e de qualidade, no entanto trabalhosa e lenta. O ideal seria um processo automático para inserção de novas URLs a partir de uma fonte externa, diferente daquela que estivesse sendo visitada pelo *crawler*.

Alguns sites da *Web*, como o Google (`http://www.google.com`) e Alexa (`http://www.alexa.com`), fornecem serviços de informações sobre URLs (sites): título, descrição, rank, links de sites que o apontam, sites relacionados e outras informações. É possível criar um processo que execute de tempos em tempos a coleta automática de novas URLs destas fontes (no máximo 10 URLs para o Alexa e 50 para o Google), obtendo porções diferenciadas da *Web* (ver Figura 7). Para consultar estes serviços é necessário apenas a URL, recuperando novas URLs sem precisar realizar o *download* da página HTML apontada pelo endereço. Este processo será chamado de *expansão na Web* ao longo deste trabalho.

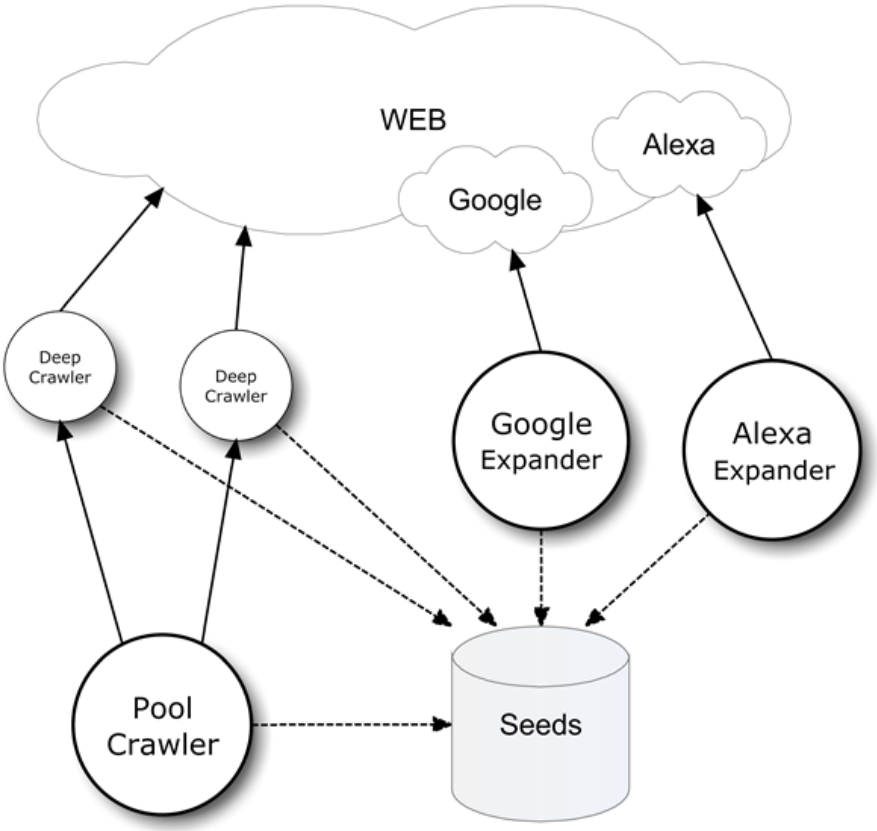


Figura 7: Expansão na *Web*

Alguns destes serviços costumam ser pagos: o do Alexa, por exemplo, cobra alguns centavos de dólar para cada mil consultas realizadas. Estes também podem ser limitados e lentos, não permitindo consultas simultâneas e exigindo um intervalo de espera entre

Tabela 13: Análise das URLs obtidas através da expansão na *Web*

	UNT	%
URLs buscadoras	400	46.2%
URLs não-buscadoras	303	35.0%
URLs com Erro	163	18.8%
Total de URLs	866	100%

as consultas. Portanto, não serão todas as URLs que deverão ser consultadas. À medida que o *crawler* visita suas URLs, aquelas que possuem formulários de busca são marcadas. Então, neste processo de *expansão na Web*, serão submetidas apenas as URLs identificadas como buscadoras pelo *crawler*, iniciando-se logo após este e continuando em paralelo à sua execução, estabelencendo uma relação mútua de alimentação.

Uma amostra de 100 URLs buscadoras foi selecionada arbitrariamente a partir de um conjunto de 1.340 URLs buscadoras conhecidas, revisado manualmente. Neste trabalho, uma URL é considerada buscadora se existe pelo menos um formulário de busca. Utilizando o classificador desenvolvido, discutido anteriormente na Seção 3.1, 95% destas URLs foram classificadas como buscadoras.

Em seguida esta amostra é submetida para *expansão na Web*. No total, foram recuperadas 1.939 URLs relacionadas com a amostra inicial das quais 1.352 eram distintas. Executando novamente o classificador para 866 destas URLs e, desconsiderando-se os 18.8% apresentaram erro (principalmente de conexão), 56.90% identificadas como buscadoras e 43.10% não-buscadoras. A Tabela 13 apresenta um resumo dos resultados gerais.

3.5 Arquitetura

Nesta seção será apresentada a arquitetura da solução desenvolvida, apontando onde foram aplicadas as técnicas discutidas e apresentadas anteriormente. A ferramenta desenvolvida está dividida em três processos principais: *DeepCrawler*, *AlexaExpander* e *GoogleExpander*. O primeiro é o *crawler* propriamente dito, responsável por visitar as páginas *Web* procurando por formulários de consulta. Os outros dois são responsáveis por recuperar novas URLs (expansão de URL) em serviços disponíveis na *Web*: Alexa (`http://www.alexa.com`) e Google (`http://www.google.com`), respectivamente.

Todos esses processos necessitam de um conjunto de URLs inicial (sementes) para que seja possível a realização de suas tarefas. As URLs conhecidas ficam armazenadas no

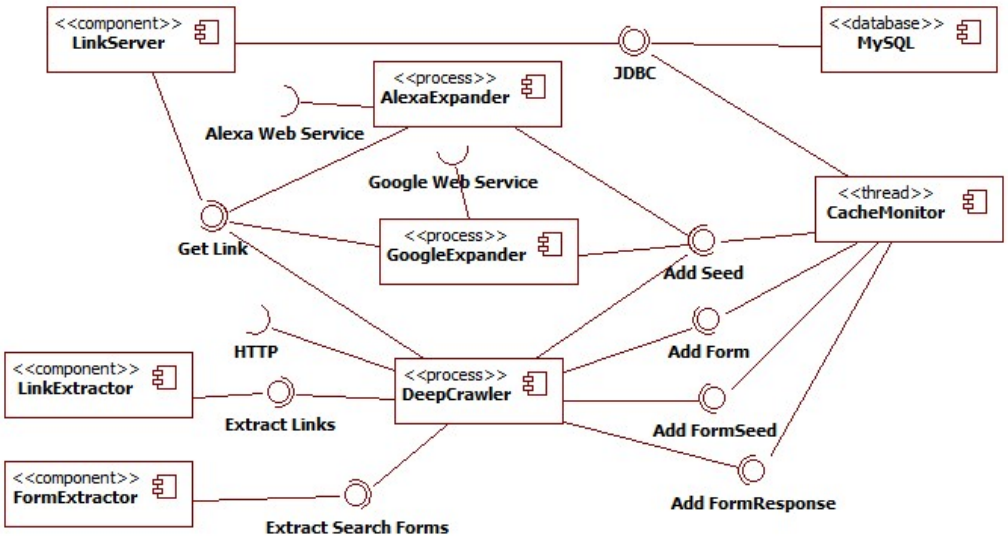


Figura 8: Diagrama de Componentes

banco de dados e o componente responsável por selecionar quais devem ser processadas é o *LinkServer*. O *LinkServer* fornece ao *DeepCrawler* e aos processos de expansão uma conjunto com melhores URLs dentre os critérios de seleção definidos. Um bom critério para definição das melhores URLs é o *PageRank* do Google, disponbilizados em alguns sites na Web, como o DigPageRank (`http://www.digpagerank.com`). Outros critérios utilizados são: a profundidade, distância entre da página atual raiz do site, quanto menor melhor; e tamanho da URL, priorizando as URLs menores.

O processo do *DeepCralwer* é executado utilizando várias *threads* em paralelo, não deixando o processador ocioso enquanto o *download* (relativamente demorado) das páginas HTML é realizado. O *DeepCrawler*, possui um controlador de *threads*, o *DeepCrawlerPool*, que recebe as URLs do servidor links (*LinkServer*) e cria uma nova tarefa para cada uma desta, executadas no *DeepCrawlerTask*.

As *threads* do *DeepCrawler* representam os *crawlers* do sistema. Eles realizam o *download* das páginas HTML, extraem e armazenam apenas as informações necessárias. O download das páginas é realizado com auxílio do *PageDownloader*, que realiza o controle de conexão, evitando que o processo fique preso por muito tempo nesta (*time out*) ou em infinitos redirecionamentos: algumas URLs que realizam redirecionamento para uma nova URL, repetindo este processo infinitamente e travando os *crawlers*. Logo após baixadas, as páginas passam por uma filtragem no código HTML: remoção de scripts, decodificação de caracteres HTML (e.g. á para á) e correção de erros nas tags que interferem na estrutura representativa em árvore DOM, montadada com o auxílio da API HTML

Parser (`http://htmlparser.sourceforge.net`).

Após realizado este tratamento, os formulários de busca serão extraídos pelo *FormExtractor*. Formulários HTML estão representados no código HTML através da *tag* `<FORM>` e são facilmente extraídos depois de contruída a árvore DOM com o auxilío do *HTML Parser*. O *FormExtractor* extrai todos os formulários HTML encontrados e os submete à classificação pelo *FormClassifier* (ver Figura 9), que identifica os que representam formulários de busca.

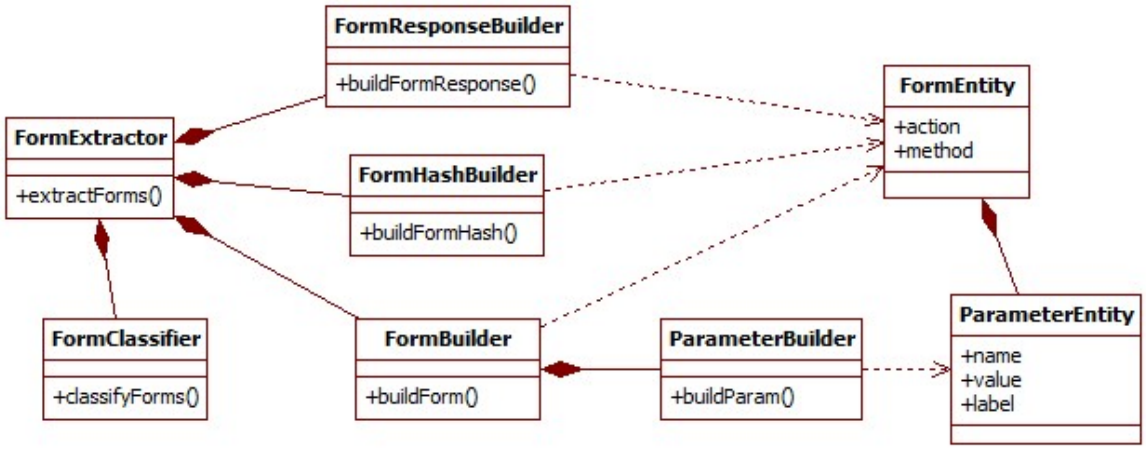


Figura 9: Diagrama de Classes - FormExtractor

O código HTML dos formulários classificados como buscadores serão transformados em objetos e submetidos ao *FormHashBuilder*, atribuindo-lhes um identificador único, utilizando uma função hash a partir dos elementos estruturais do formulário (action, input tags, etc.). Em seguida, um novo identificador, agora da página de resposta, é construído, chamado de *FormResponse* e representa uma assinatura do conteúdo escondido atrás de um formulário de consulta. O componente *FormResponseBuilder* é o responsável por extrair esta assinatura, que é utilizada para identificar uma mesma base de dados, acessada por diferentes formulários. O Diagrama de Ações, Figura 10, apresenta as principais etapas e os produtos intermediários no processo de extração dos formulários de busca a partir do código HTML.

Além dos formulários de busca, os *crawlers* também são responsáveis por coletar os links presentes nas páginas e armazená-los no banco de dados junto às existentes. Estes links são extraídos pelo *LinkExtractor* a partir do código HTML, selecionando o parâmetro HREF das tags âncora <A>. No entanto, não são todas as URLs encontradas em uma página que devem ser coletadas. O *LinkExtractor* utiliza o *LinkSelector* para descartar URLs consideradas inúteis e selecionar apenas as que apresentam maior possibilidade de

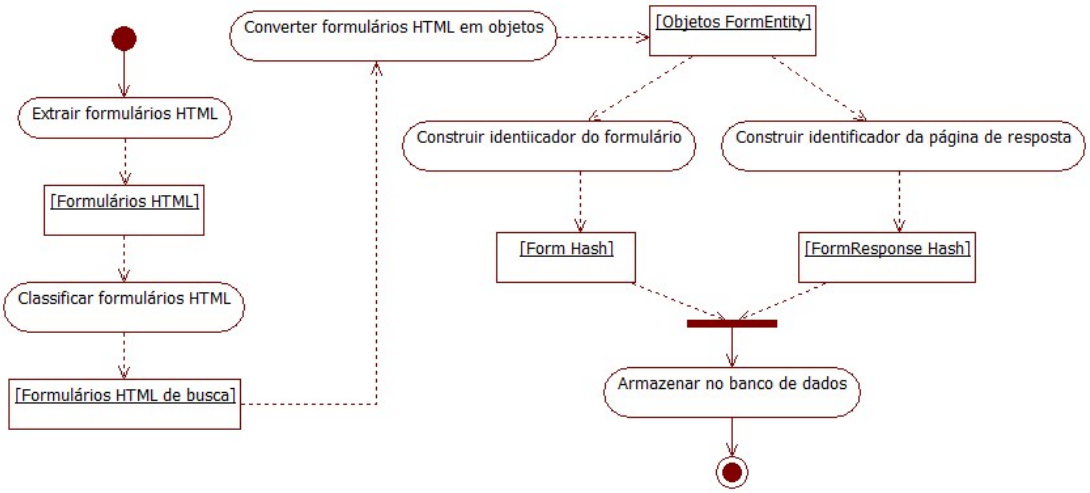


Figura 10: Diagrama de Ações - extração de formulários de busca

levarem a páginas que possuem formulários de consulta (ver Seção 3.3) - *hyperlinks* com texto âncora igual a *"Search"*, por exemplo. Ao final, estas informações são persistidas e as páginas baixadas pelos *crawlers* são descartadas.

Todo este processo demanda uma grande quantidade de operações, salvando e recuperando as informações armazenadas no banco de dados constantemente. Visando reduzir o tempo de acesso à informação e evitando a sobrecarga no banco de dados, um esquema de cache foi desenvolvido, mantendo grande parte das entidades em memória, gerenciadas pelo *CacheMonitor*. O *CacheMonitor* intercepta as ações realizadas diretamente com o banco, utilizando o padrão *Facade* (GAMMA et al., 1995), provendo acesso aos demais componentes do cache: *SeedCache*, *FormCache*, *FormResponseCache*, *FormSeedCache*.

O *SeedCache*, por exemplo, mantém em memória todas as URLs sob a forma de um código *hash* de 12 bytes, evitando que a existência de uma mesma URL seja verificada várias vezes no banco. A princípio, utilizou-se apenas um código *hash* gerado com o algoritmo *hash* proposto por Uzgalis (1996), um `long` (8 bytes), onde facilmente observou-se colisões. Para evitar maiores colisões, um `inteiro` (4 bytes) do tamanho da texto - neste caso URL - passou a distinguir quando diferentes URLs geravam o mesmo código hash de 8 bytes. Entretanto, mesmo com o par código hash e tamanho do texto, é possível haver colisões, optando-se pelo ganho computacional em memória - uma URL com 50 caracteres possui 100 bytes, seu hash representa 12% do tamanho real.

As assinaturas das páginas de resposta, os *FormResponses*, demandam várias conexões com a Internet para sua construção, por este motivo *FormCache* mantém em memória o hash de todos os formulários conhecidos e seu *FormResponse* correspondente, possibil-

itando rapidamente sua localização e recuperação. Os diferentes formulários que representam a mesma base podem ser encontradas em diferentes endereços *Web*, então *FormResponseCache* guarda para todas as bases de dados conhecidas, qual o melhor formulário que identifica esta base e o endereço onde foi encontrado. Armazenar tamanha quantidade de informação em memória compromete diretamente a escabilidade do sistema, devendo ser melhorada esta solução, caso venha ser utilizada em larga escala.

Outras operações de banco, como inserção/alteração de URLs, formulários e assinaturas das bases de dados, são realizadas em lote e sempre coordenadas pelo *CacheMonitor*. Além de manter representações compactas das entidades do banco em memória, listas com as novas entidades e com entidades modificadas também são mantidas. O *CacheMonitor* ordena o salvamento quando excede o limite de entidades em memória ou o tempo da última operação é muito longo.

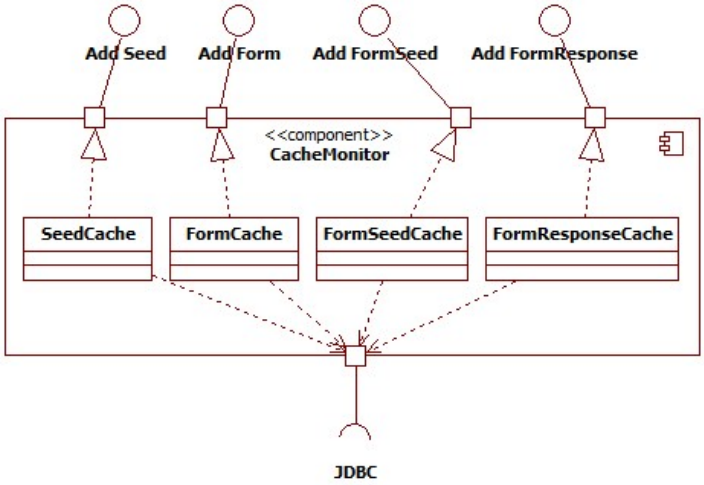


Figura 11: Componente CacheMonitor

O processo do *AlexaExpander* e *GoogleExpander* realizam a expansão na *Web* discutida na Seção 3.4. Estes dois processos executam separados do processo do *crawler* e utilizam dos serviços do Alexa e Google, respectivamente, para recuperar novas URLs e inserí-las no banco de dados com o auxílio do *CacheMonitor*. Diferente do *DeepCrawler*, os processos de expansão não necessitam realizar o *download* do código HTML da página, bastam apenas utilizar URLs recebidas do *LinkServer*, e que já foram visitadas pelo *DeepCrawler*, para consulta nos provedores de serviço. Entretanto, estes serviços apresentam restrições de conexão simultânea e intervalo entre consultas, o que não é problema para o *crawler*, que pode visitar dezenas de URLs simultaneamente.

3.6 Modelagem dos Dados

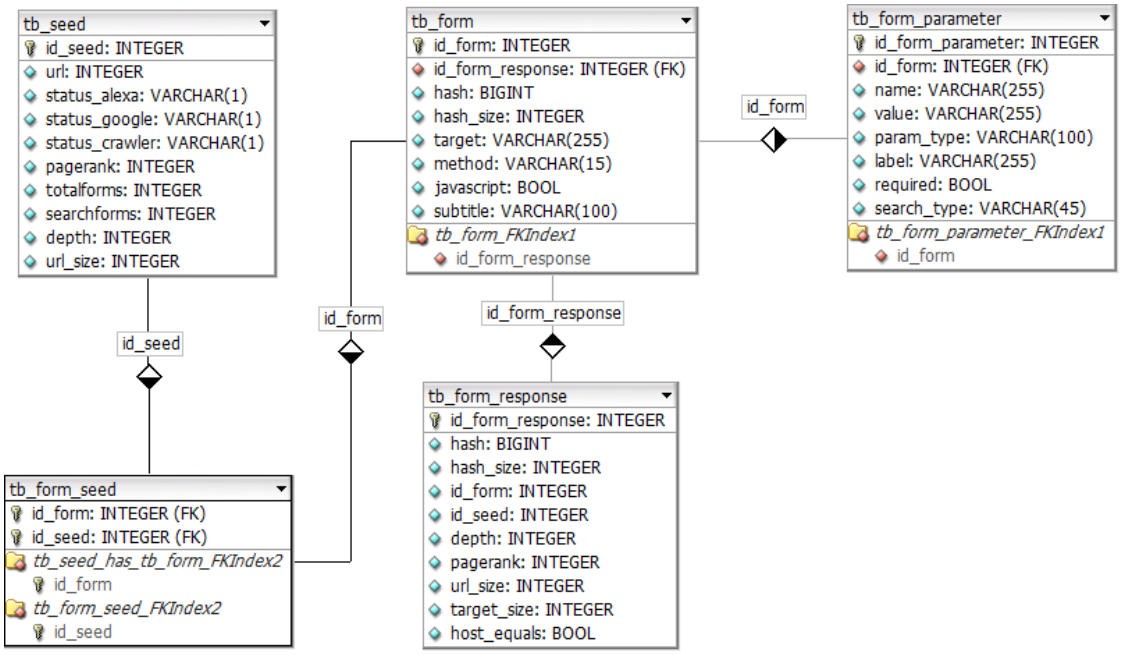


Figura 12: Modelo Relacional Lógico

O modelo de dados utilizado, ilustrado na Figura 12, é constituído das cinco principais tabelas: *tb_seed*, *tb_form*, *tb_form_parameter*, *tb_form_seed* e *tb_form_response*.

A principal tabela, que também é onde inicia todo o processo, é a *tb_seed*. Nesta tabela, na coluna *url*, ficam armazenadas o conjunto de URLs que servirão de sementes para os *crawlers*. As colunas *status_crawler*, *status_alexa* e *status_google* são utilizadas para armazenar o estado atual da URL (inicial, processado e erro) nos processos do DeepCrawler, AlexaExpander e GoogleExpander respectivamente. A quantidade de formulários HTML encontrados pelo *crawler* na página é armazenada em *totalforms* enquanto que a quantidade de formulários HTML classificados como buscadores é armazenada em *searchforms*. As demais colunas, *pagerank*, *depth*, *url_size* e *rank*, servem de critérios para eleger as melhores URLs, definindo a ordem de processamento.

A tabela *tb_form*, guarda as informações dos formulários buscadores encontrados: hash, target (ou action), method (post ou get), javascript, subtitle, e outros; além do código *hash* do formulário, que deve assegurar sua unicidade, e o relacionamento com identificador da base de dados, o qual falaremos mais à frente.

Os parâmetros (*input tags*) do formulário HTML são armazenados em *tb_form_parameter* com o nome, valor, label e o tipo do parâmetro encontrado (input, textarea, select, button, submit, etc.). Um formulário encontrado pode possuir diversos parâmetros, no entanto

não são todos obrigatórios para a realização da busca. Parâmetros de múltipla escolha, como os tipos checkbox, radio ou select, permitem que o usuário escolha um único valor (ou vários), descartando os demais no momento da busca. Os marcados por padrão, no momento em que o formulário é recuperado, serão considerados requeridos (coluna *required*), enquanto que os demais parâmetros não serão descartados mas serão marcados como não requeridos.

Os relacionamentos entre os formulários e as URLs onde foram encontrados estão guardados na tabela *tb_form_seed*. Este relacionamento é do tipo N:N, pois em uma URL podem ser encontrados vários formulários, e um mesmo formulário pode estar presente em diversas URLs.

Por fim, os identificadores das bases de dados da *Web* profunda (hash dos elementos presente nas páginas de resposta) são armazenados em *tb_form_response*. Diversos formulários podem levar à mesma base de dados. Em *tb_form_response* são armazenadas bases de dados distintas, indicando qual foi o melhor formulário HTML (*id_form*) que dá acesso a esta base e em qual endereço *Web* (*id_seed*) este formulário foi encontrado. As outras colunas (i.e. *depth*, *pagerank*, *url_size*, *target_size* e *host_equals*) guardam os critérios utilizados para eleger o melhor formulário para esta base, utilizando da desnormalização, para acelerar o carregamento destes dados na memória.

3.7 Considerações Finais

Este capítulo apresentou as técnicas necessárias na construção da solução proposta neste trabalho. Foram mostradas as diferenças existentes entre formulários HTML buscadores e não-buscadores, acompanhada da solução de classificação escolhida para identificação dos formulários buscadores, as entradas para as bases de dados da *Web* profunda. Também foi discutido neste capítulo a existência de diferentes formulários de busca acessando a mesma base de dados, técnicas de seleção de *links* das páginas HTML pelo *crawler* e uma proposta de alimentação externas de novas URLs para o *crawler*, utilizando serviços disponíveis na *Web*. A solução de crawler para identificação das bases de dados *online* é construída com base nestas técnicas, além do esquema de cache utilizado para reduzir o acesso ao banco de dados e, consequentemente, melhoria de desempenho.

4 Estudo de Caso

Os resultados experimentais apresentados até então foram executados em ambientes laboratoriais, buscando-se mensurar a eficiência de cada uma das técnicas apresentadas ao longo deste trabalho.

Neste capítulo, serão apresentados os resultados da utilização da ferramenta desenvolvida em uma máquina de busca real: utilização no sistema da máquina de busca para a *Web* profunda Goshme. A Seção 4.1 apresenta a máquina de busca Goshme utilizada como base para avaliação de desempenho das estratégias de *crawler* discutidas nesta monografia, apresentadando as principais características da máquina de busca e em seguida os resultados obtidos.

4.1 Máquina de Busca Goshme

O Goshme não é uma máquina de busca convencional. As máquinas de busca convencionais tentam trazer os documentos da *Web* mais relevantes para a consulta do usuário. O Goshme, por sua vez, procura trazer as melhores bases de dados, ou buscadores verticais, que abordam a consulta do usuário. Funciona como uma consulta especializada, devolvendo fontes com centenas de documentos sobre o assunto pesquisado.

O Goshme não funciona como um Meta-buscador. Meta-buscadores pesquisam simultaneamente nos principais buscadores da *Web* (no máximo 8), e trazem como resultado os links dos documentos encontrados nestes buscadores (GOSHME, 2006). O que o Goshme faz é encontrar o buscador que o usuário necessita dentre os cadastrados em seu sistema, preenchendo automaticamente o formulário e lhe enviando diretamente à página de resposta do buscador, onde serão encontrados os verdadeiros links para o resultado.

O sistema da máquina de busca Goshme, possuía uma solução para localização de bases de dados, o *Engine Finder*, que ajudou a cadastrar 1.258 bases de dados de forma semi-automática - URLs descobertas automaticamente com formulários cadastrados ma-

nualmente, número muito baixo quando deseja-se responder a qualquer consulta realizada pelo usuário. O *Engine Finder* apenas pré-categorizava as URLs como candidatas a buscadoras, com base apenas em palavras-chave (e.g. *search*) presente no conteúdo da página ou no texto-âncora do *link*, verificado logo quando a URL é encontrada. Esta solução não extraía, identificava ou salvava os formulários de busca automaticamente, este processo era realizado por intervenção manual, a partir das URLs pré-selecionadas pelo *Engine Finder*. Após meses de cadastro manual, vários problemas foram identificados dentre as URLs eleitas: muitas sequer possuíam formulários HTML; outras até possuíam formulários, mas não eram de busca; e uma enorme quantidade de URLs categorizadas possuíam formulários repetidos, por coletar muitas URLs do mesmo domínio.

O *Engine Finder* mostrou-se pouco eficiente e deveria substituído. A ferramenta desenvolvida neste trabalho, chamada de *DeepCrawler*, deveria ser capaz de localizar automaticamente novas bases de dados e integrá-las ao sistema Goshme, mantendo a compatibilidade com a arquitetura existente.

A nova arquitetura da máquina de busca Goshme, já com o módulo *DeepCrawler*, é apresentada na Figura 13. O processo é iniciado no módulo *DeepCrawler*, que identifica os formulários de consulta e armazena-os no banco de dados. O *DatabaseInterector* verifica se os formulários são realmente de busca, realizando sucessivas consultas e aprendendo como interagir com a base. O mesmo também detecta a lógica de boolenos (i.e. busca por todas as palavras, apenas algumas, frase exata) que a base de dados utiliza e extrai o seu template - esquema para separar os resultados do restante da página. O *LanguageModel*, ou simplesmente *LM*, recupera o conteúdo escondido na base de dados, usando o template extraído para identificar os *links* de resultado na página de resposta, e realiza a indexação com auxilio do *VIF Index*. Quando o usuário final utiliza a máquina de busca Goshme, através da aplicação *GoshmeWeb*, sua consulta é enviada ao processador de consultas *VIF Search*, que acessa o *Index* em disco e devolve as bases de dados ordenadas por relevância. Outras *features* (i.e. notícias, enciclopédia) também são apresentadas ao usuário, adquiridas no servidor de *features*, o *GoshmeWebFeatures*.

4.2 Metodologia e Ambiente de Experimentação

O ambiente de experimentação foi composto por um único computador com sistema operacional Red Hat Enterprise Linux 4. Operando com um processador Intel Core 2 Duo, com 4 gigabytes de memória principal e dois discos SATA de 150 e 250 gigabytes e

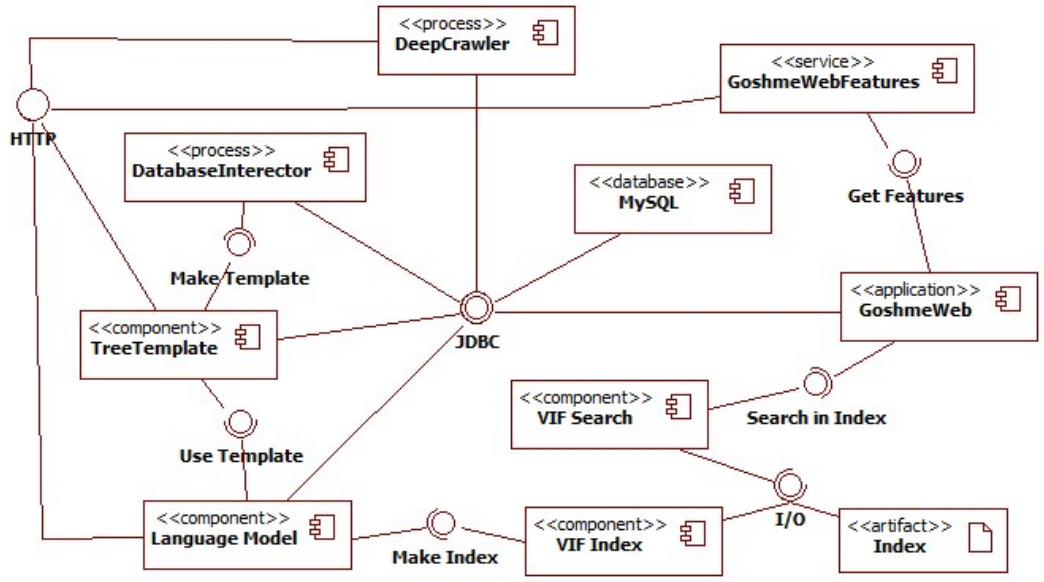


Figura 13: Nova arquitetura do Goshme com o componente *DeepCrawler*

conexão de 100 Mbps com a Internet.

As solução foi desenvolvida na liguagem Java, da Sun Microsystems, utilizando a versão 1.6 do JDK (*Java Development Kit*) e banco de dados MySQL versão 5.04.

O *DeepCrawler* deveria, a partir de um conjunto inicial de URLs (sementes), ser capaz de identificar automaticamente quais possuíam entradas para bases de dados e cadastrá-las do sistema Goshme. Aquelas URLs identificadas como buscadoras deveriam ser expandidas na *Web* (ver Seção 3.4) e os links contidos nas páginas HTML visitadas passariam por uma seleção e os escolhidos seriam guardados para futura visita. Este processo de expansão e seleção de links garantiriam a retroalimentação do sistema e que o mesmo permanecesse continuamente em execução.

O conjunto inicial de URLs foi construído automaticamente, efetuando sucessivas consultas específicas (e.g. *"audio search engine"*, *"science search engine"*) nos sites do Google (`http://www.google.com`), Yahoo! (`http://www.yahoo.com`) e CompletPlanet (`http://www.completeplanet.com`), coletando as URLs das páginas de resposta, totalizando pouco mais de 15.000 URLs distintas, aguardando para serem visitadas pelo *crawler*.

4.3 Resultados

Após aproximadamente 45 dias em execução, o processo foi interrompido e os resultados analisados (acompanhar Tabela 14). A quantidade que de URLs conhecidas pelo *crawler* passou para 972.306. Estas novas URLs provieram da extração dos *hyperlinks* contidos nas páginas visitadas e do resultado do processo de expansão na *Web*. No instante em que o processo foi interrompido, 965.174 já haviam sido processadas pelo *crawler* e apenas 32.675 haviam sido expandidas na *Web*, verificando a lentidão comentada na Seção 3.4.

Tabela 14: Resultados finais do DeepCrawler no sistema Goshme

URLs conhecidas	972.306
URLs visitadas	965.174
URLs com formulários de busca	237.576
Total de formulários HTML	584.753
Formulários buscadores	378.207
Formulários buscadores distintos	152.796
Supostas Bases de dados distintas	110.273
Formulários Interagidos	33.688
Bases de dados identificadas	23.466
Previsão de bases de dados	76.816

Foram contabilizados 584.753 formulários HTML, presentes em todas as URLs visitadas, com 378.207 formulários classificados como buscadores. Foram 237.576 sites, mais especificamente URLs, que possuíam formulários de consulta em suas páginas, entretanto muitos desses formulários eram comuns entre eles, exatamente o mesmo, reduzindo para 152.796 a quantidade formulários HTML de busca distintos.

Como apresentado na Seção 3.2, diferentes formulários de busca podem dar acesso à mesma base de dados. Interagindo com os formulários e criando um identificador para sua base de dados (ver Seção 3.2), os formulários classificados como buscadores dariam acesso a supostamente 110.273 bases distintas na *Web*. Dizemos supostamente porque até o presente momento não é certeza que os formulários recuperados são realmente de busca. Esta verificação é realizada logo após a localização dos formulários e base realizada pelo *crawler*, através de um outro componente do sistema Goshme. Esta verificação é mais lenta, pois exige uma interação maior com os supostos formulários de consulta, realizando sucessivas conexões a fim de descobrir se realmente é uma base de dados da *Web* profunda e quais palavras são aceitas por esta base. O processo foi executado por alguns dias, interrompendo-se após analisados 33.688 das 110.273 supostas bases de dados através de

seus formulários, certificando-se que 23.466 (69.66%) são realmente bases de dados da *Web* profunda, que nos permite estimar que ao final contaremos com 76.816 bases de dados.

A Tabela 14 apresenta os resultados finais da ferramenta *DeepCrawler* desenvolvida neste trabalho e integrada ao sistema da máquina de busca para *Web* profunda Goshme, disponibilizando mais de 20.000 novas bases de dados, de forma automática, prontas para serem exploradas, o que acontecerá nas etapas seguintes do sistema.

Dentre outros problemas encontrados, a qualidade de alguns dos formulários recuperados monstrou-se ruim. Alguns deles, apesar de responderem à consulta por palavras-chave, não eram tão interessantes: localização de carros, hotéis e outros através de código Zip; páginas de ajuda e suporte, com resposta a problemas de software ou serviço; presença de *Free Servers*, sites que respondem a qualquer consulta com *links* patrocinados, logo, irrelevantes para o usuário. Além deste problema, diversas possibilidades de trabalhos futuros serão discutidas na Seção 5.2.

5 Conclusão

5.1 Conclusões

Este trabalho estudou três diferentes classificadores, utilizados na identificação pré-query dos formulários de consulta que dão acesso às bases de dados da *Web* profunda. Os três classificadores foram efetivamente implementados e testados, adotando-se aquele que descartou a menor quantidade de formulários de consulta, apresentando uma maior precisão na classificação dos formulários não-buscadores.

Estes pontos de entrada, interfaces de buscas, são raros na *Web* e visitar todas as URLs da *Web* superficial à sua procura pode representar uma tarefa exaustiva e sem grande sucesso. Então, além do estudo dos classificadores, também foram apresentadas as técnicas utilizadas durante a seleção dos *hyperlinks* presentes nas páginas, que reduz a quantidade de *links* inúteis que devem ser visitados pelos *crawlers*, além de outras estratégias para alimentação do processo através de fontes externas.

Uma proposta de arquitetura para os novos *crawlers* da *Web* profunda, focados na localização de suas bases de dados, foi apresentada na Seção 3.5, apresentando seus principais módulos e demonstrando a técnica de cache utilizada para a redução de consultas ao banco de dados.

Os resultados experimentais apresentados no Capítulo 4, quando aplicados à máquina de busca Goshme, revelam que a abordagem utilizada aumentou o número de bases de dados da *Web* profunda, recuperando rapidamente mais de 20.000 bases de dados prontas para serem exploradas e previsão para mais de 70.000 bases de dados a serem integradas.

O presente trabalho foi realizado com o apoio do Goshme (http://www.goshme.com), através da parceria firmada com a Universidade Estadual de Feira de Santana, respeitando as cláusulas de publicação definidas neste convênio.

5.2 Trabalhos Futuros

Durante o desenvolvimento deste trabalho, percebeu-se que a capacidade computacional era limitada, impossibilitando-nos de visitar todas as URLs encontradas pelo *crawler*. Medidas deveriam ser tomadas para que o *crawler* pudesse ter o maior alcance possível, não ficando preso em porções da Web. A seleção de links apresentada na Seção 3.3 foi executada apenas com os links encontrados nas páginas cuja a URL fossem profundidade zero (e.g. `http://www.goshme.com`), pois centenas de milhares de URLs eram coletadas e não seriam visitadas em tempo hábil. Caso este sistema venha ser utilizado em larga escala em dezenas de máquinas distribuídas e Internet de alta velocidade, pretende-se remover esta restrição e continuar descendo até a profundidade três a partir da raiz do site, que é onde se concentram grande parte das entradas para as bases de dados, como afirmado por Chang et al. (2003).

Verificou-se manualmente que uma grande quantidade de sites buscadores conhecidos não haviam sido classificados como buscadores. Após a análise do código HTML de dezenas deles, verificou-se que muitos possuíam seus formulários HTML gerados total ou parcialmente por JavaScript, uma linguagem utilizada na construção de páginas dinâmicas e executada nos *browsers* dos clientes (*client side*). A interpretação desta linguagem envolve outras tecnologias não utilizadas neste trabalho, como a interpretação do JavaScript dentro do ambiente Java. Algums projetos vem sendo desenvolvidos nessa linha como o interpretador de JavaScript Rhino (MOZZILA, 2007) e o redenrizador de HTML Cobra (LOBO, 2002) que detecta mudanças na árvore DOM provocadas pelo JavaScript, os quais serão estudados em outra oportunidade, a fim de recuperar um número maior de bases de dados.

A presença de campos de texto múltiplos, obrigatórios ou não na consulta, dificultou a iteração com as bases de dados através do formulário de consulta: seria necessário descobrir qual campo realmente será utilizado pelo usuário. Outros elementos do formulário, como *comboboxes* de data presentes em sites de notícias, influenciam na montagem do *ranking* pelos buscadores, limitando o alcance dos resultados presentes em uma base. Como o crawler é responsável por armazenar a representação do formulário que será utilizado pelo sistema, será necessário criar heurísticas capazes de identificar o quão relevante é um campo texto e o quão irrelevante são os *comboboxes* para a recuperação dos documentos de uma base.

Pretende-se também futuramente descobrir sobre qual assunto as páginas HTML vi-

sitadas abordam. Com esta informação será possível distribuir mais uniformemente o *crawler* entre as áreas de conhecimentos ou a uma área específica de interesse, limitando-se apenas as URLs coletadas nas páginas pertencentes ao domínio desejado.

Referências

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern Information Retrieval*. [S.l.]: Addison Wesley, 1999. Paperback. ISBN 020139829X.

BARBOSA, L.; FREIRE, J. Searching for hidden-web databases. In: *WebDB*. [S.l.: s.n.], 2005. p. 1–6.

BARBOSA, L.; FREIRE, J. Combining classifiers to identify online databases. In: *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007. p. 431–440. ISBN 978-1-59593-654-7.

BERGMAN, M. K. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, University of Michigan, v. 7, n. 1, 2001. Disponível em: <http://www.press.umich.edu/jep/07-01/bergman.html>. Acesso em: Outubro de 2007.

BRANSKI, R. M. Localização de informações na internet características e formas de funcionamento dos mecanismos de busca. In: *Economia & Tecnologia*. [S.l.: s.n.], 2000. v. 12, n. 1, p. 11–19.

CHANG, K. et al. *Structured databases on the web: Observations and implications*. 2003.

COPE, J.; CRASWELL, N.; HAWKING, D. Automated discovery of search interfaces on the web. In: . [S.l.: s.n.], 2003.

DILIGENTI, M. et al. Focused crawling using context graphs. In: *26th International Conference on Very Large Databases, VLDB 2000*. Cairo, Egypt: [s.n.], 2000. p. 527–534.

DMOZ. *Dmoz Directory*. 2007. Disponível em: <http://dmoz.org>. Acesso em: Outubro de 2007.

FONTES, A. de C.; SILVA, F. S. Smartcrawl: a new strategy for the exploration of the hidden web. In: LAENDER, A. H. F.; LEE, D.; RONTHALER, M. (Ed.). *WIDM*. [S.l.]: ACM, 2004. p. 9–15. ISBN 1-58113-978-0.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Addison-Wesley Professional, 1995.

GOSHME. *Goshme White Paper - Quick Overview*. May 2006.

GULLI, A.; SIGNORINI, A. The indexable web is more than 11.5 billion pages. In: *WWW 2005*. [S.l.: s.n.], 2005.

HEALTHLINE. *Healthline - Health Search Engine and Medical Information*. 2007. Disponível em: <http://www.healthline.com>. Acesso em: Outubro de 2007.

IWS. *World Internet Usage Statistics News and PopulationStats*. 2007. Disponível em: <http://www.internetworldstats.com/stats.htm>. Acesso em: Fevereiro de 2008.

LOBO. *Cobra: Pure Java HTML Renderer and Parser (Open Source)*. 2002. Disponível em: <http://lobobrowser.org/cobra.jsp>. Acesso em: Março de 2008.

LYMAN, P. et al. *How much information? 2003*. 2003. Disponível em: <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/>. Acesso em: Dezembro de 2007.

MOZZILA. *Rhino - JavaScript for Java*. 2007. Disponível em: <http://www.mozilla.org/rhino/>. Acesso em: Março de 2008.

QUINLAN, J. R. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0.

RAGHAVAN, S.; GARCIA-MOLINA, H. Crawling the hidden web. In: *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. p. 129–138. ISBN 1-55860-804-4.

REZENDE, S. O. *Sistemas Inteligentes: fundamentos e aplicações*. [S.l.]: Manole, 2005. ISBN 8520416837.

SARAIVA, P. C. *Mecanismos Eficientes de Cache em Máquinas de Busca para a Web*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, 2001.

UZGALIS, R. *Hashing Concepts and the Java Programming Language*. 1996. Disponível em: <http://www.serve.net/buz/hash.adt/java.000.html>. Acesso em: Março de 2008.

W3C. *World Wide Web Consortium*. 2007. Disponível em: <http://www.w3.org>. Acesso em: Março de 2008.

WEKA. *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. 2007. Disponível em: <http://www.cs.waikato.ac.nz/ ml/weka/>. Acesso em: Março de 2008.

YAHOO. *Yahoo!* 2008. Disponível em: <http://www.yahoo.com>. Acesso em: Abril de 2008.

ANEXO A – Conjunto de novas características identificadas

Abreviatura	Característica	Tipo
NumText	Quantidade de campos do tipo *text*	Numérico
NumSubmit	Quantidade de campos do tipo *submit*	Numérico
NumImage	Quantidade de campos do tipo *image*	Numérico
NumPassword	Quantidade de campos do tipo *password*	Numérico
NumFile	Quantidade de campos do tipo *file*	Numérico
NumTextarea	Quantidade de campos de tipo *textarea*	Numérico
FormActionWords1	*action* do formulário contém palavras-chave*	Booleano
FormNameEquals	*name* do formulário é igual a *sForm* ou *qForm*	Booleano
FormIdEquals	*id* do formulário é igual a *sForm* ou *qForm*	Booleano
FormNameWords1	*name* do formulário contém palavras-chave*	Booleano
FormIdWords1	*id* do formulário contém palavras-chave*	Booleano
FormNameWords2	*name* do formulário contém palavras-chave**	Booleano
FormIdWords2	*id* do formulário contém palavras-chave**	Booleano
TextNameWords2	*name* do campo texto contém palavras-chave**	Booleano
TextIdWords2	*id* do campo texto contém palavras-chave**	Booleano
TextValueWords2	*value* do campo texto contém palavras-chave**	Booleano
SubmitNameWords1	*name* do botão de submisão contém palavras-chave*	Booleano
SubmitIdWords1	*id* do botão de submisão contém palavras-chave*	Booleano
SubmitValueWords1	*value* do botão de submisão contém palavras-chave*	Booleano
SubmitNameWords2	*name* do botão de submisão contém palavras-chave**	Booleano
SubmitIdWords2	*id* do botão de submisão contém palavras-chave**	Booleano
SubmitValueWords2	*value* do botão de submisão contém palavras-chave**	Booleano
ImageAltEquals	*alt* de *image* é igual a *Go* ou *Go!*	Booleano
ImageAltWords1	*alt* de *image* contém palavras-chave*	Booleano
ImageAltWords2	*alt* de *image* contém palavras-chave**	Booleano

**search*, *find*, *query*, *seek*

***login*, *username*, *subscribe*, *sign in*, *sign me*, *sign up*, *log on*, *log in*, *e-mail*, *email*, *e_ mail*, *translate*, *translation*, *join*

ANEXO B – Árvore de decisão C4.5 com características de Cope, Craswell e Hawking (2003)

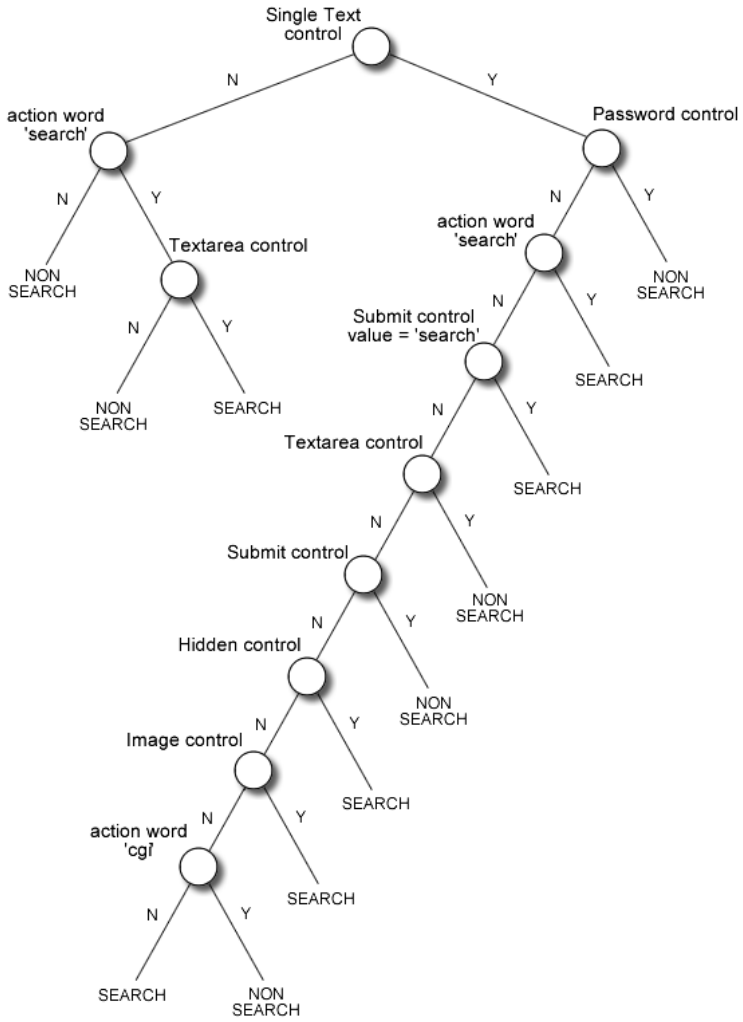


Figura 14: Árvore de decisão C4.5 com características de Cope, Craswell e Hawking (2003)

ANEXO C – Árvore de decisão C4.5 com novas características

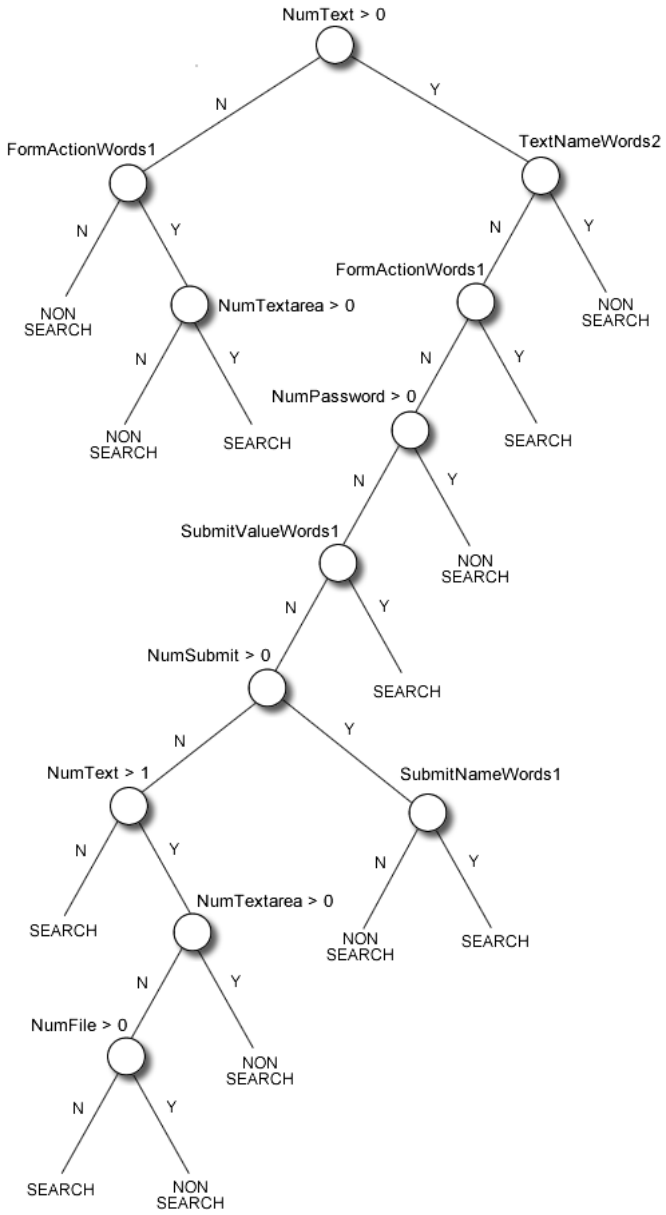


Figura 15: Árvore de decisão C4.5 com novas características (ver Anexo A)

ANEXO D – Árvore de decisão construída manualmente com novas características

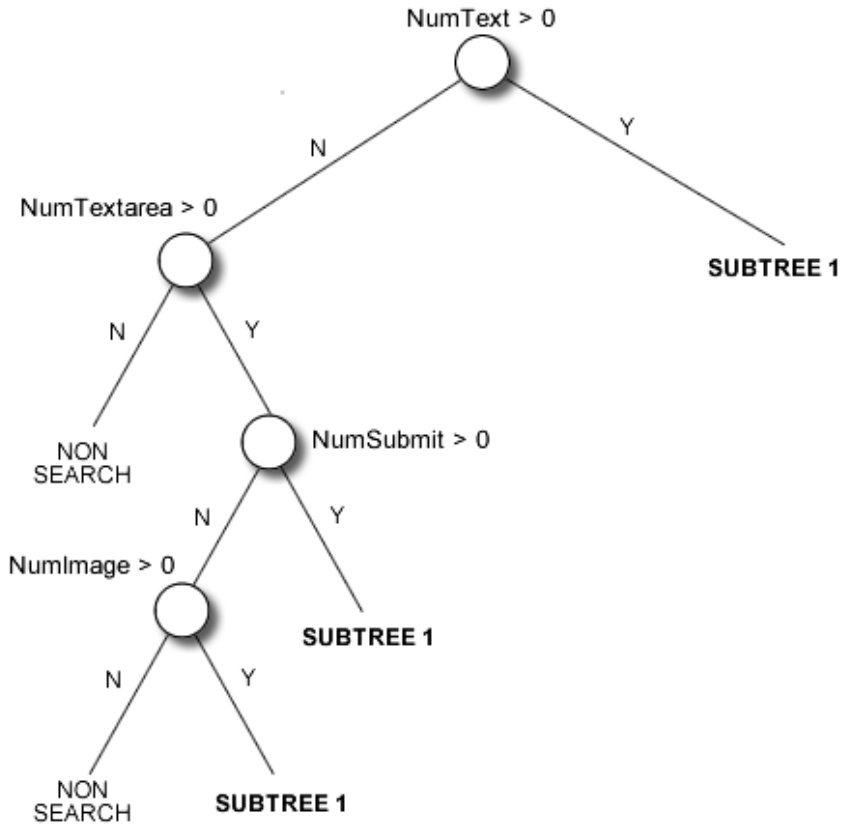


Figura 16: Base da árvore de decisão construída manualmente com novas características (ver Anexo A)

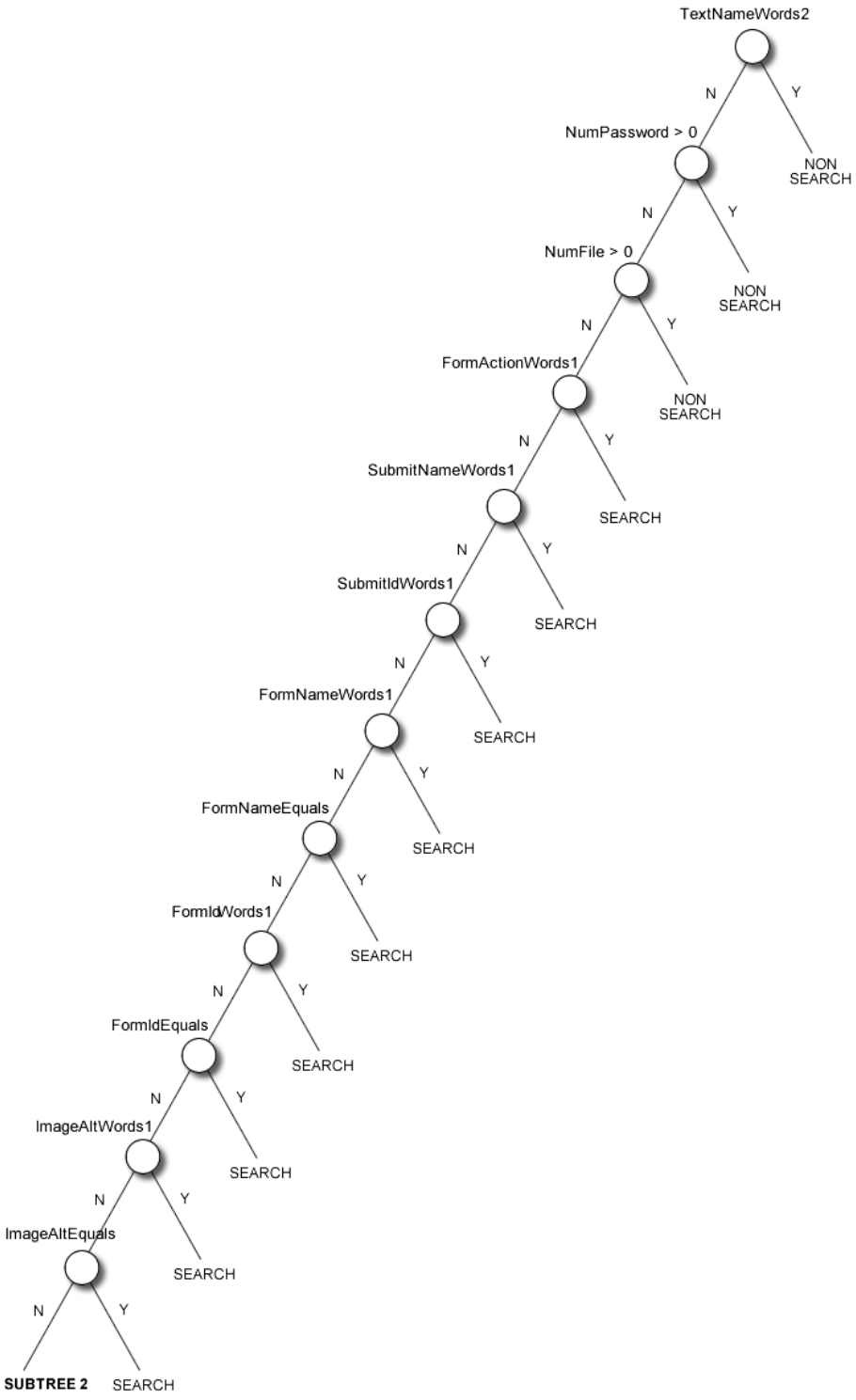


Figura 17: Sub-árvore 1 da árvore de decisão construída manualmente

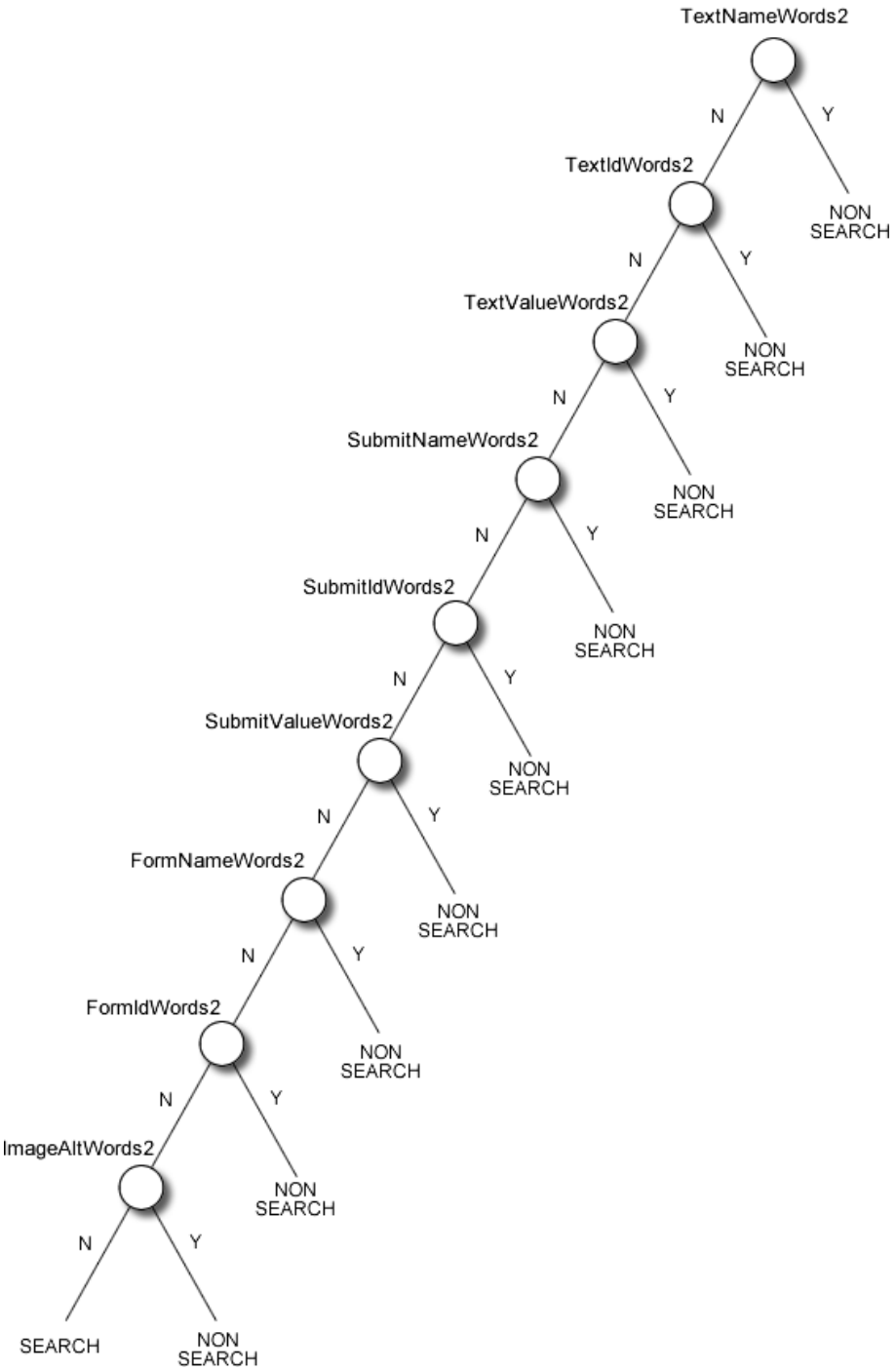


Figura 18: Sub-árvore 2 da árvore de decisão construída manualmente