UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO
PROJETO FINAL DE GRADUAÇÃO

Trabalho de Graduação de Curso: "Como os search engines funcionam?"

Autor: Fábio Carvalho Motta Ricotta

Itajubá – MG
Novembro, 2007

UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO

PROJETO FINAL DE GRADUAÇÃO

COMO OS SEARCH ENGINES FUNCIONAM?

Trabalho organizado por FÁBIO CARVALHO MOTTA RICOTTA
Matrícula 11871

Orientador: Roberto Affonso da Costa Junior

ITAJUBÁ
Novembro, 2007

AGRADECIMENTOS

Agradeço a todos aqueles que sempre me apoiaram, sempre com um voto de confiança nos momentos de inovação e superação. Agradeço principalmente ao destino, que colocou a Universidade Federal de Itajubá no meu caminho, mostrando que para grandes futuros não é necessário uma grande caminhada, basta olhar opções próximas a você.

Gostaria de agradecer a minha família, por me apoiar, incentivar e colaborar com a minha jornada de cinco anos na Universidade. A cada vitória e derrota que eu vivia eles sempre estiveram ao meu lado, prontos para me dar todo o suporte necessário e conselhos sempre sábios.

Uma pessoa importante que não pode ser deixada de lado é minha namorada, Adriana, que presenciou quase dois anos desta longa jornada, sempre ao meu lado e mostrando como a vida pode ser feliz e mais alegre, mesmo a vida colocando muitos obstáculos, ela sempre me apoiou, vibrou, chorou, riu e comemorou todos os meus feitos na Universidade, na minha vida pessoal e profissional.

Realizo agradecimentos também, ao meu orientador Roberto Affonso da Costa Junior e à professora Adriana Mattedi, que ofereceram todo o suporte ao desenvolvimento desta monografia.

Por fim, agradeço a todas as oportunidades que tive, juntamente com uma mente saudável para tomar decisões sábias em momentos apropriados.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CSS - Cascading Style Sheets

DOC - Documento

FTP - File Transfer Protocol

Gzip - GNU zip

HTML - HyperText Markup Language

HTTP - Hyper Text Transfer Protocol

IBM - International Business Machines Corporation

MIME - Multipart Internet Mail Extension

MSN - Microsoft Network

OPIC - On-line Page Importance Computation

PDF - Portable Document Format

PHP - PHP: Hypertext Preprocessor

RAR - Roshal ARchive

RSS - Rich Site Summary

SGML - Standard Generalized Markup Language

URL - Universal Resource Locator

UTF-8 - Unicode Transformation Format 8 bit encoding

XML - Extensible Markup Language

WWW - World Wide Web

RESUMO

Um *search engine* é um web site especializado em buscar e listar páginas da internet a partir de palavras-chave indicadas pelo utilizador.

Os *search engines* surgiram com a intenção de prestar um serviço importante: a busca de qualquer informação na web, apresentando os resultados de uma forma organizada, e também com a proposta de fazer isto de uma maneira rápida e eficiente. Ele permite que uma pessoa solicite conteúdo de acordo com um critério específico (tipicamente contendo uma dada palavra ou frase) e responde com uma lista de referências que combinam com tal critério.

Os *search engines* baseiam sua coleta de páginas em um robô que varre a Internet à procura de páginas novas para introduzir em sua base de dados automaticamente. Eles possuem índices atualizados constantemente para operar de forma rápida e eficiente.

Quando um usuário faz uma busca, tipicamente digitando palavras-chave, o sistema procura o índice e provê uma lista das páginas que melhor combinam ao critério, normalmente com um breve resumo contendo o título do documento e, às vezes, partes do seu texto.

Este trabalho abordará as três grandes áreas da arquitetura do *search engine*, Web Crawling, Indexação Web e Busca Web, e seguida apresentará um exemplo ilustrativo desta arquitetura.

ABSTRACT

A search engine is a web specialized site in search and retrieve pages of the internet starting from keywords indicated by the user.

The search engines appeared with the intention of rendering an extremely important service: the search of any information in the web, presenting the results in an organized way, and also with the proposal of doing this in a fast and efficient way. It allows a person to request content in agreement with a specific (typically containing a given word or sentence) criterion and it shows a list of references that combine with such criterion.

The search engines base their collection of pages automatically on a robot that sweeps the Internet searching new pages to introduce in their database. They have indexes updated constantly to operate in a fast and efficient way.

When a user makes a search, typically typing keyword, the system seeks the index and provides a list of the pages that best combines to the criterion, usually with an abbreviation summary containing the title of the document and, sometimes, parts of web page text.

This work will approach the three great areas of the architecture of the search engine, Web Crawling, Web Indexing and Web Search, and following it will present an illustrative example of this architecture.

1 INTRODUÇÃO

1.1 Contexto e objetivos

Search engine ou motor de busca é um programa feito para auxiliar a encontrar informações armazenadas em um sistema de computadores, por exemplo, como a *World Wide Web* (WWW), dentro de uma rede corporativa ou um computador pessoal. Ele permite que uma pessoa solicite conteúdo em web sites de acordo com um critério específico (tipicamente contendo uma dada palavra ou frase) e responde com uma lista de referências que combinam com tal critério. Os motores de busca usam regularmente índices atualizados para operar de forma rápida e eficiente. Os motores de busca em sua maioria referem-se ao serviço de busca Web, que procuram informações na rede pública da Internet. Outros tipos incluem *search engine* para empresas (Intranets), *search engines* pessoais e *search engines* móveis.

A Internet é um conjunto de inúmeras redes de computadores, conectadas entre si, que permite a comunicação, partilha de informações, programas e equipamentos entre seus usuários. Constitui a infra-estrutura sobre a qual trafegam grande volume de informações e outros serviços.

A Internet teve origem em uma rede, a Arpanet. Esta foi criada pelo Departamento de Defesa dos Estados Unidos, na época da Guerra Fria no início dos anos 70, interligando vários centros militares e de pesquisa com objetivos de defesa, na época da Guerra Fria. A tecnologia desenvolvida permitia a comunicação entre diferentes sistemas de computação, o que possibilitou a incorporação de outras redes experimentais que foram surgindo ao longo do tempo.

Os *search engines* surgiram logo após o aparecimento da Internet, com a intenção de prestar um serviço importante: a busca de qualquer informação na web, apresentando os resultados de uma forma organizada, e também com a proposta de realizar a tarefa de uma maneira rápida e eficaz. A partir deste preceito básico, diversas empresas surgiram, desenvolvendo produtos para este fim. Entre as maiores e mais conhecidas empresas encontram-se o Google, o Yahoo, o MSN, o Baidu, e mais recentemente a Amazon.com com o seu mecanismo de busca A9.

Os primeiros motores de busca (como o Yahoo) baseavam-se na indexação de páginas através da sua categorização. Posteriormente surgiram as meta-buscas. A mais recente geração de motores de busca (como o Google) utiliza tecnologias diversas, como a procura por palavras-chave diretamente nas páginas e o uso de referências externas espalhadas pela web, permitindo até a tradução direta de páginas, de maneira rústica, para a língua do utilizador.

Atualmente, a informação tornou-se o ativo mais valorizado para o público em geral, desde empresas, governos até pessoas físicas. A recuperação de informações em banco de dados tem sido um assunto muito discutido pelos profissionais da área de ciência da informação. O advento da Internet tornou esta questão ainda mais premente. A difusão de seu uso ampliou a necessidade de acessar, de forma rápida e precisa, informações armazenadas em banco de dados gigantescos.

Perante isto, várias empresas têm surgido e se firmado no mercado com produtos que visam tornar mais curtas as distâncias entre pessoas e a informação desejada. Estas empresas fazem isto por meio de *search engines*, os quais trazem, ou melhor, procuram trazer de forma ampla tudo o que uma pessoa deseja saber sobre determinado assunto.

A forma com que um *search engine* trabalha normalmente não é conhecida, pois trabalham com uma fórmula secreta, desconhecida por completo, até mesmo pelas equipes de desenvolvimento do próprio motor de busca. A divisão de equipes, em áreas especializadas, neste tipo de produto é fundamental, pois assim os segredos dos algoritmos ficam restritos às equipes que os desenvolvem.

Esta fórmula de busca é objeto de desejo de muitos usuários e desenvolvedores da área. Logo, este trabalho tem por objetivo estudar como os *search engines* funcionam e tentar traçar um perfil padrão desta fórmula “secreta”, discutindo de forma clara e objetiva, o seu funcionamento e gerenciamento.

Com base nesta pesquisa, muitos desenvolvedores poderão criar e melhorar seus motores de busca e até mesmo sugerir novas formas de disposição da informação para os usuários.

Este trabalho abordará as três grandes áreas da arquitetura do *search engine* e apresentará um exemplo ilustrativo desta arquitetura. As três áreas de um

motor de busca consistem em: *Web crawling*, onde as informações são capturadas; Indexação, responsável pela organização e classificação das informações; e por fim, a Busca, onde o usuário requisita informações e o *search engine* retorna um conjunto de dados referentes ao(s) termo(s) informados.

1.2 Estrutura do trabalho

O trabalho está dividido em seis capítulos. O primeiro é esta introdução onde se procura definir o objetivo do trabalho e a justificativa para realizá-lo. O Capítulo 2, Fundamentação Teórica, apresenta um resumo das principais bibliografias e artigos de sobre o assunto tratado neste trabalho, abordando as três frentes de desenvolvimento de um *search engine*

No capítulo 3, *Web crawling*, apresenta-se como os sistemas de busca trabalham armazenando informações sobre um grande número de páginas, as quais eles obtêm na internet. Será feita uma avaliação de algoritmos para captura de dados, identificação de sistemas já desenvolvidos, levantamento de requisitos físicos e teóricos para a construção do robô de busca.

No capítulo 4, Indexação, discute-se como o conteúdo de cada página é analisado para determinar como deverá ser indexado (por exemplo, as palavras são extraídas de títulos, cabeçalhos ou campos especiais chamados *meta tags*). Serão analisadas as formas de organização das informações no Banco de Dados, de forma que fique otimizado e fornecendo um tempo de resposta mínimo ao usuário.

O capítulo 5 mostra as técnicas para que os usuários do motor de busca consigam encontrar facilmente as respostas para suas perguntas. Será realizada uma análise de visual nos principais *search engines*, identificando as áreas de informações para os usuários.

Por fim, o Capítulo 6 traz as conclusões do trabalho assim como possíveis desdobramentos desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Em 1989 o armazenamento, acesso, e busca de coleções de documentos foi revolucionada por uma invenção chamada *Word Wide Web* por seu fundador Tim Berners-Lee (Gilles e Caillinau, 2000). A invenção de coleções de documentos com ligações foi bem original naquela época, devido ao fato de uma publicação da anterior a época, "*As May We Think*" (Bush, 1945), ter ilustrado uma máquina futurista que operava similarmente a um ser humano na identificação de relação entre documentos ligados.

A *Word Wide Web* se tornou o último sinal da dominação da Idade da Informação e a morte da Idade Industrial. Com esta revolução do armazenamento e acesso de informação, a quantidade de dados inserida na Web se tornou muito grande e com uma quantidade de atualizações alta. Com esta quantidade de informações tão grande e diversificada é praticamente impossível encontrar tudo o que se precisa sem o uso de uma ferramenta especializada em recuperação de informação.

A primeira ferramenta utilizada para busca na Web foi o Archie, desenvolvido por Alan Emtage em 1990, o qual indexava arquivos de computador. Já o primeiro search engine Web foi o Wandex, um índice feito pela World Wide Web Wanderer, com um *web crawler* desenvolvido por Matthew Gray no MIT em 1993. O primeiro sistema "full text" baseado em *crawler* foi o WebCrawler, que saiu em 1994. Ao contrário de seus predecessores, ele permite aos usuários buscar por qualquer palavra em qualquer página, o que tornou-se padrão para todos serviços de busca desde então (Wikipedia PT, Search Engine).

Existem ferramentas de busca muito boas na Internet, como o Altavista, o Baidu, o Yahoo e o MSN. No entanto, nenhum desses sites consegue ter a amplitude e qualidade do Google. Existem boas razões para isso.

O Google atualiza sua base de informações diariamente, pois existe um crawler, chamado de Googlebot, que busca por informações novas em todos os websites. Isso é realmente interessante pois cerca de 30 minutos após se publicar um artigo em um blog já é possível encontrá-la no Google. Outros mecanismos de busca

também possuem crawlers, mas eles não são tão eficientes em termos de atualização e de classificação de informações.

Outra razão para o sucesso do Google (Wikipedia, Google) é o sistema de classificação PageRank. Trata-se de um algoritmo desenvolvido por Larry Page, um dos fundodadores do Google, que atribui uma pontuação a cada página web de acordo com a quantidade e a qualidade dos *links* que apontem para ela. Este algoritmo é a base na definição da ordem em que os resultados serão mostrados aos usuários. Pode-se associar o PageRank à um nível de confiabillidade sobre determinado assunto, onde quem possui um link para uma determinada página, está ofederenco um voto de confiança para aquele conteúdo. Além disso, o Google analisa os assuntos mais pesquisados e verifica quais sites tratam sobre aquele tema de maneira significativa, verificando a quantidade de vezes que o termo pesquisado aparece na página, por exemplo.

Outros dois fatores importantes do sucesso do search engine são a simplicidade e clareza. A combinação desses itens foi aprimorada desde a concepção do Google. Devido a isso, é possível acessar um site leve, sem poluição visual e cujas opções são facilmente localizáveis.

Tabela 1 – Evolução dos *Search engines*

Ano	Engine	Evento
1993	Wandex	Lançamento
1993	Aliweb	Lançamento
1994	WebCrawler	Lançamento
	Infoseek	Lançamento
	Lycos	Lançamento
1995	AltaVista	Lançamento (parte do DEC)
	Excite	Lançamento
1996	Dogpile	Lançamento
	Inktomi	Fundado
	Ask Jeeves	Fundado
1997	Northern Light	Lançamento
1998	Google	Lançamento
1999	AlltheWeb	Lançamento
1999	Baidu	Fundado
2000	Singingfish	Lançamento
2000	Teoma	Fundado
2000	Vivisimo	Fundado
2003	Objects Search	Lançamento
2004	Yahoo! Search	Lançamento final (primeiros resultados originais)
	MSN Search	Lançamento beta
2005	MSN Search	Lançamento final
2006	Quaero	Fundado
	Ask.com	

Fonte: Wikipedia - http://pt.wikipedia.org/wiki/Search_Engine - Visitado em 2/11/2007

2.1 *Search engines* Geoespaciais

Uma recente melhoria na tecnologia de busca(Wikipedia, Search Engines) é a adição de geocodificação e *geoparsing* para o processamento dos documentos ingeridos. O *geoparsing* tenta combinar qualquer referência encontrada a lugares para um quadro geoespacial de referência, como um endereço de rua, localizações de dicionário de termos geográficos, ou a uma área (como um limite poligonal para uma municipalidade). Através deste processo, as latitudes e longitudes são atribuídas aos lugares encontrados e são indexadas para uma busca espacial posterior. Isto pode melhorar muito o processo de busca, pois permite ao usuário procurar documentos para uma dada extensão do mapa, ou ao contrário, indicar a localização de documentos combinando com uma dada palavra-chave para analisar incidência e agrupamento, ou qualquer combinação dos dois.

2.2 Futuro dos *Search engines*

Um dos princípios de uma guerra comercial é a de que não se pode desafiar o líder do mercado a não ser que se tenha muito mais poder de fogo ou uma tática inovadora. Assim, geralmente os mercados se assentam com um líder em posição de defesa, um concorrente forte desafiador e muitos outros guerrilheiros.

O cenário atual dos motores de busca é o seguinte:

Tabela 2 - Os 10 primeiros motores de busca com relação a buscas em Julho de 2007

Provedor	Buscas (000)	Percentagem no total das buscas (%)
Google	4,143,752	53.3
Yahoo	1,559,745	20.1
MSN/Windows Live	1,057,064	13.6
AOL	407,988	5.2
Ask.com	143,513	1.8
My Web Search	69,145	0.9
BellSouth	40,374	0.5
Comcast	37,311	0.5
Dogpile	25,675	0.3
My Way	24,534	0.3
Outros	264,073	3.4
Total de Buscas	7,773,174	100.0

Fonte: Nielsen//NetRatings, 2007

O detalhe é que o mercado de buscas é tão grande que existe um grande número de motores de busca pequenos, os quais atuam em sua maioria em paises isolados ou em comunidades internacionais, como é o caso da China e a Comunidade Européia.

No caso dos motores de busca há uma forte concorrência de nichos, determinadas por barreiras culturais e lingüísticas. Essas particularidades propiciam para algumas empresas uma vantagem enorme sobre os líderes de mercado.

O Baidu é o site de busca preferido entre os chineses e o 4º mais acessado no mundo. A sua vantagem é ser chinês, entender os chineses e fazer aplicativos específicos para os chineses.

O Quaero, ainda não lançado, é uma iniciativa estatal da Comunidade Européia. Ele foi pensado exatamente para tirar o poder das buscas das mãos dos norte-americanos. Sua vantagem é contar com o poder de alguns dos mais ricos Estados do mundo: Alemanha, França, Itália.

O Raftaar é o melhor site de buscas para o universo multilinguístico indiano. Sua vantagem é funcionar com idiomas similares ao Hindu.

O Sawafi será um buscador para o arábe, uma língua com caracteres complicados e difícil de conciliar nos motores de busca existentes.

2.3 ReREOS – *Search engine* de Casas Hipotecadas

A ReREOS é um projeto de um *search engine* focado em casas hipotecadas (“foreclosures”) do mercado americano. Este projeto é desenvolvido na empresa Deals Consultoria e Desenvolvimento e tendo como gerente Fábio Carvalho Motta Ricotta.

Este sistema foi amparado pelos conhecimentos adquiridos pelo profissional durante o desenvolvimento desta monografia, ganhando assim uma maior robustez, funcionalidades e qualidade de informação.

Por razões de sigilo de negócios, esta monografia não contém exemplos embasados no *search engine* ReREOS, mas para ilustrar as áreas abordadas, é utilizado o *search engine open-source* Sphider.

2.4 Sphider – *Search engine* em PHP

O Sphider é um popular *search engine* web *open-source* construído pelo programador Ando Saabas e encontra-se, atualmente, na versão 1.3.3, lançada em 15 de setembro de 2007.

O Sphider possui um *crawler* automático, que pode percorrer *links* encontrados em um website, e um indexador que cria um índice com todos os termos encontrados nas páginas. Ele é escrito em PHP e utiliza o MySQL como banco de dados.

2.5 Características

2.5.1 Crawling e Indexação

- Realiza indexação do texto completo;
- Pode indexar páginas estáticas e dinâmicas;
- Encontra *links* via href, frame, area e meta tags, e ainda pode seguir *links* fornecidos no javascript como, por exemplo, em *window.location* e *window.open*;
- Respeita o protocolo robots.txt, e as tags *nofollow* e *noindex*;
- Segue redirecionamentos de servidor;

- Permite o crawling limitado por profundidade (número máximo de cliques apartir da página incial) por (sub)domínio ou por diretório;
- Permite o crawling de URLs que atendem (ou não atendem) à certas palavras-chaves ou expressões fornecidas;
- Suporta a indexação de arquivos PDF ou DOC;
- Permite o resumo de processos de crawling já iniciados;
- Possibilita a exclusão de palavras a serem indexadas.

2.5.2 Busca

- Suporta os operadores binários AND, OR ou busca por frases;
- Suporta exclusão de palavras;
- Opção de adicionar e agrupar sites em categorias;
- Possibilidade de limitar a busca de uma determinada categoria e suas subcategorias;
- Possibilidade de buscar em um único domínio;
- Sugestão de busca “Você quis dizer” ou erros de digitação;
- Sugestão de busca em tempo real conforme palavras digitadas, como realizadas no Google Suggest;
- Busca por sinônimos.

2.5.3 Administração

- Inclui uma interface sofisticada de administração;
- Suporta a indexação via interface web ou linha de comando, facilitando o uso de tarefas programadas (cron job);
- Estatísticas de busca;
- Sistema simples de template, facilitando a integração com um web site.

3 *WEB CRAWLING*

Um web crawler, também conhecido como web spider ou web robot, é um programa ou um script automático que percorre a *World Wide Web* de uma forma metódica e automática. Outros nomes menos frequentes usados para os web crawlers são *ants, automatic indexers, bots,* e *worms*.

Este processo de percorrer a WWW é chamado de *Web crawling* ou *spidering*. Muitos sites, em particular os *search engines*, usam o método de spidering como um meio de obter conteúdo atualizado. Os web crawlers são usados principalmente para criar uma cópia de todas as páginas visitadas para um processamento posterior do *search engine*, que irá indexar as páginas para prover buscas rápidas. Os crawlers também podem ser usados para automatizar tarefas de manutenção em um website, como verificar *links* ou validar conteúdo HTML. Os crawlers pode também ser utilizados para obter informações específicas de páginas na Web, como a procura por emails, usualmente para spam.

Um web crawler é um tipo de bot, ou um agente de software. Em geral, ele inicia com uma lista de URLs a serem visitadas, que é chamado de sementes. De acordo com que o crawler visita estas URLs, ele identifica todos os hiperlinks na página e adiciona-os na lista de URLs a serem visitadas, chamada de fronteira de percorrimento. As URLs desta fronteira são recursivamente visitadas de acordo com um conjunto de políticas.

3.1 *World Wide Web*

A *World Wide Web* (Wikipedia, Word Wide Web) é um sistema de documentos em hipermídia que são interligados e executados na Internet. Os documentos podem estar na forma de vídeos, sons, hipertextos e figuras. Para visualizar a informação, pode-se usar um programa de computador chamado navegador para descarregar informações (chamadas "documentos" ou "páginas") de servidores web (ou "sítios") e mostrá-los na tela do usuário. O usuário pode então seguir as hiperligações na página para outros documentos ou mesmo enviar informações de volta para o servidor para interagir com ele. O ato de seguir hiperligações é comumente chamado de "navegar" ou "surfar" na Web.

3.2 Hiperligação

Uma hiperligação (Wikipedia, Hiperligação), ou simplesmente uma ligação (também conhecida em português pelos correspondentes termos ingleses, *hyperlink* e *link*), é uma referência num documento em hipertexto a outro documento ou a outro recurso. Como tal, pode-se vê-la como análoga a uma citação na literatura. Ao contrário desta, no entanto, a hiperligação pode ser combinada com uma rede de dados e um protocolo de acesso adequado e assim ser usada para ter acesso direto ao recurso referenciado. Este pode então ser gravado, visualizado ou mostrado como parte do documento que faz a referência.

3.3 Políticas de Crawling

Existem três características importantes na Web que criam um cenário no qual o *Web crawling* é muito difícil:

- é um grande volume,
- mudanças acontecem rapidamente, e
- geração de páginas dinâmicas,

que combinados produzem uma grande variedade de URLs possíveis de serem indexadas.

O grande volume implica que o crawler só pode captura uma fração de páginas da web em um determinado tempo, então ele necessita priorizar esta captura. O grande volume de mudanças implica que no tempo em que o crawler está capturando as últimas páginas de um site, é muito comum que novas páginas tenham sido adicionadas neste site, ou outras páginas foram atualizadas ou até mesmo excluídas.

O recente crescimento no número de páginas geradas por scripts server-side tem criado uma dificuldade em infinitas combinações de parâmetros HTTP GET, onde uma pequena quantidade irá retornar conteúdo único. Por exemplo, uma simples galeria de fotos pode oferecer três opções para os usuários, como especificado anteriormente, parâmetros HTTP GET. Se existirem quatro formas de ordenar imagens, três escolhas de tamanhos de miniaturas, dois formatos de arquivos, e uma opção para desabilitar conteúdo provido por usuários, então o mesmo conteúdo pode ser acessado de quarenta e oito URLs diferentes, tudo isto presente no site. Esta combinação

matemática cria um problema para os crawlers, pois eles devem realizar infinitas combinações em mudanças de script para conseguir um conteúdo único.

O comportamento de um web crawler é a combinação das seguintes políticas:

- A política de seleção que seleciona quais páginas capturar;
- A política de re-visitação que diz quando se deve verificar por mudanças em uma página;
- A política de cortesia que organiza o web crawler para não sobrecarregar os websites;
- A política de paralelização que indica como coordenar os web crawlers distribuídos.

3.3.1 Política de Seleção

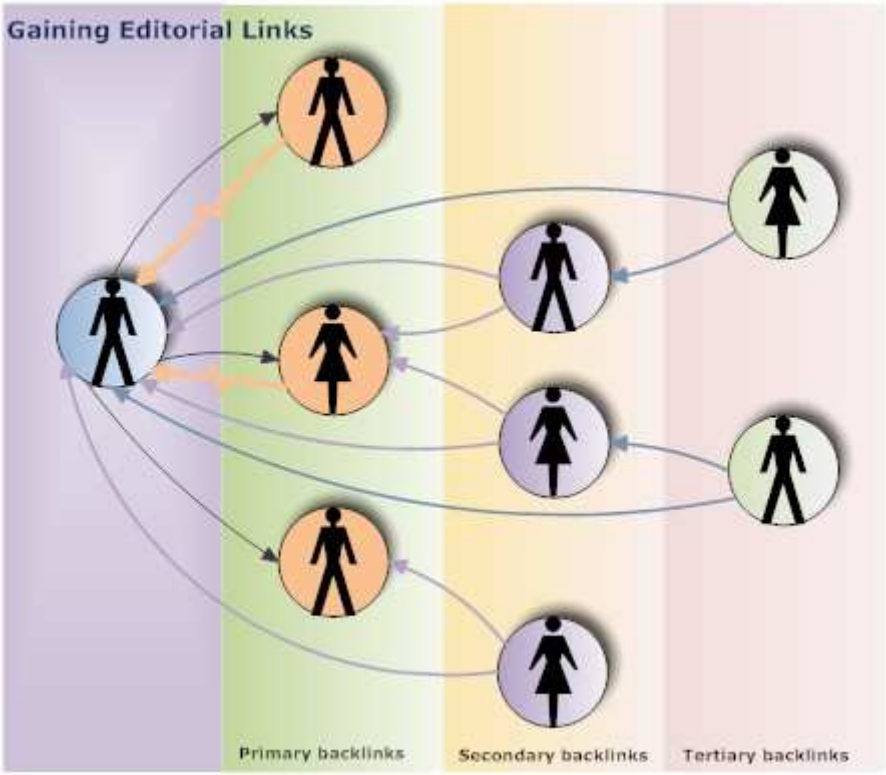
Devido ao tamanho da Web, até os maiores *search engines* cobrem apenas uma porção dos documentos públicos existentes na internet; um estudo de Lawrence e Giles(2000) mostrou que nenhum *search engine* indexa mais do que 16% da web. Como um crawler captura apenas uma fração das páginas web, é altamente desejado que esta pequena fração capturada seja de páginas relevantes, e não somente uma parte aleatória da web.

Isto requer uma métrica de importância (Gibotti da Silva, 2006) na priorização das páginas web. A importância de uma página é uma função com uma qualidade intrínseca, sua popularidade em termos de *links* ou visitas, e até mesmo de sua URL (este último é o caso de *search engines* verticais restritos a um único domínio top-level, ou *search engines* restritos a um web site fixo). Criar uma boa política de seleção possui uma dificuldade adicional: deve funcionar com parte da informação, pois o conjunto completo de páginas web não é conhecido durante a operação de crawling.

Cho et al. (1998) realizaram o primeiro estudo em políticas de agendamento de crawling. A sua amostragem de informações foi de 180.000 páginas capturadas do domínio stanford.edu, onde a simulação de crawling foi realizada com diferentes estratégias. As métricas de ordenação testadas foram *breadth-first*(Ilustração 2), *backlink-count*(Ilustração 1) e cálculos parciais de Pagerank(Ilustração 3). Uma de

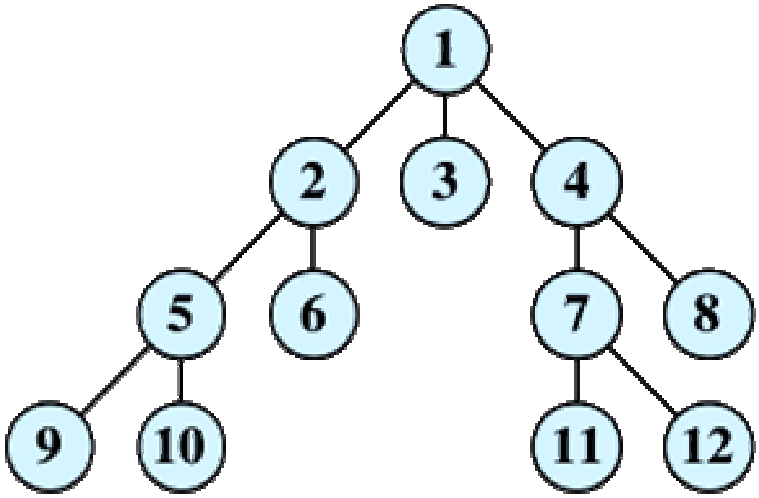
suas conclusões foi que se o crawler quiser capturar páginas com alto Pagerank, primeiramente, durante o processo de crawling, então a estratégia de Pagerank parcial é melhor, seguida da breadth-first e backlink-count. No entanto, estes resultados são para um único domínio.

Ilustração 1 - Backlinks



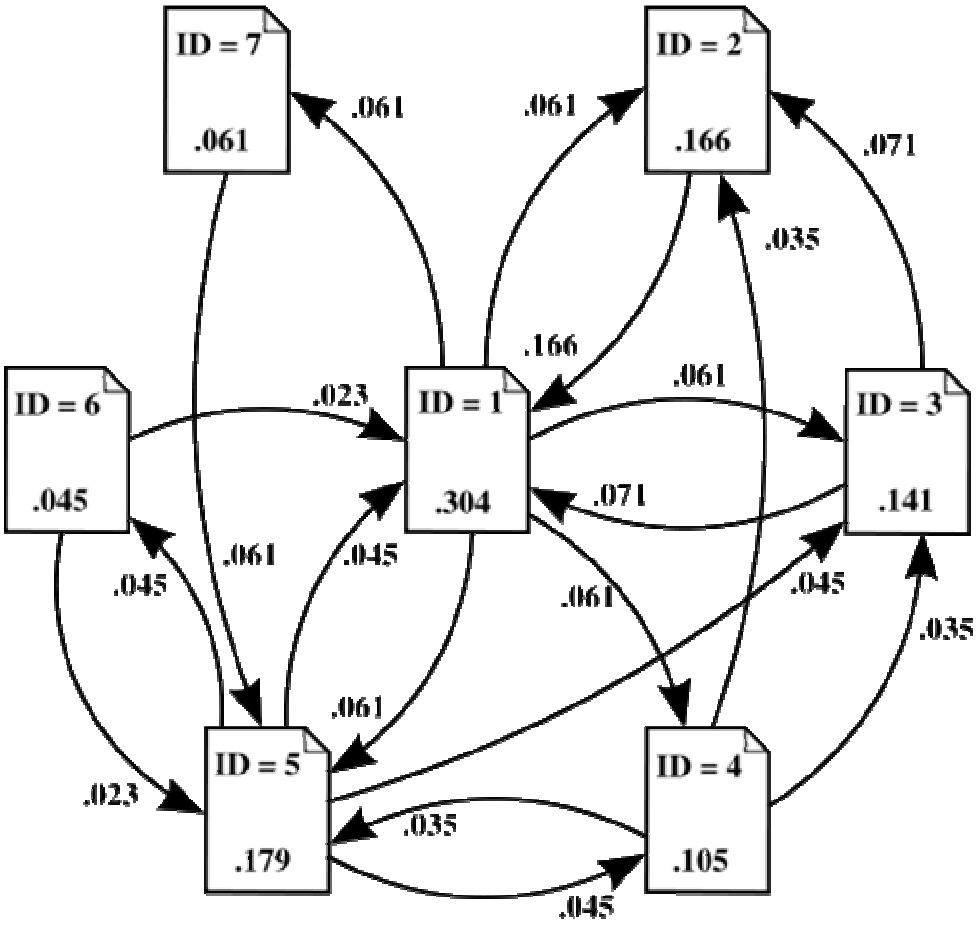
Fonte: *Search engine* Journal - http://www.searchenginejournal.com/altenate-link-building-strategies-the-linkerati-effect/4997/ - Visitado em 2/11/2007

Ilustração 2 – Busca Breadth-first – Ordem na qual os nós são expandidos



Fonte: Wikipedia - http://en.wikipedia.org/wiki/Breadth-first_search - Visitado em 2/11/2007

Ilustração 3 – Como o Pagerank funciona



Fonte: Wikipedia - http://en.wikipedia.org/wiki/Pagerank - Visitado em 2/11/2007

Najork e Wiener (2001) executaram um crawler em 328 milhões de páginas, usando a ordenação breadth-first. Eles descobriram que a captura breadth-first captura páginas com alto Pagerank inicialmente no processo de crawling (mas eles não realizaram a comparação de sua estratégia com outras). A explicação fornecida pelos autores para este resultado foi que "as páginas mais importantes possuem muitos *links* para elas de diversos hosts, e estes *links* são encontrados previamente, independentemente de qual host ou página que o crawler tenha originado".

Abiteboul et al. (2003) criaram uma estratégia de crawling baseada no algoritmo chamado OPIC (On-line Page Importance Computation). No OPIC, cada página recebe uma soma inicial de "dinheiro" que é distribuído igualmente por todas as páginas que ela aponta. É similar ao algoritmo Pagerank, mas é mais rápido e é realizado em apenas um passo. O crawler constituído pelo OPIC capturou primeiro as páginas que estavam na fronteira do crawling com grande quantidade de "dinheiro". Experimentos foram conduzidos com 100.000 páginas em grafos sintéticos com uma distribuição de poder nos *links* recebidos. Contudo, não houve comparações com outras estratégias ou experimentos reais na web.

Boldi et al. (2004) usaram uma simulação em subconjuntos da web de 40 milhões de páginas de domínio .it e 100 milhões de páginas do WebBase crawl, testando o algoritmo breadth-first contra o depth-first, ordenação randômica e uma estratégia onisciente. As comparações foram baseadas em quão bom foi a computação parcial do Pagerank na qual se aproximou do real valor do Pagerank. Surpreendentemente, algumas visitas que acumularam Pagerank muito rápido (notavelmente, a breadth-first e a visita onisciente) através de aproximações progressivas muito ruins.

Baeza-Yates et al. (2005) usou uma simulação em dois subconjuntos da web de 3 milhões de páginas dos domínios .gr e .cl, testando várias estratégias de crawling. Eles mostraram que ambas as estratégias OPIC e uma estratégia que utiliza o tamanho do número de filas de uma site são melhores que a estratégia breadth-first, e que é muito efetiva quando se usa um crawler anterior para orientar o atual.

3.3.2 Restringindo *links* a serem seguidos

Um crawler pode procurar apenas páginas HTML e deve descartar todos os outros tipos MIME. De modo a apenas requisitar recursos HTML, um crawler deve fazer uma requisição HTTP HEAD (usado para obter meta-informações por meio do cabeçalho da resposta, sem ter que recuperar todo o conteúdo), para determinar o tipo de MIME do recurso antes de requisitar o recurso completo com uma requisição GET. Para não realizar numerosas requisições HEAD, o crawler deve alternativamente examinar a URL e somente requisitar o recurso se a URL terminar com .html, .htm ou barra. Esta estratégia pode causar numerosos recursos HTML a serem descartados sem intenção. Uma estratégia similar compara a extensão do recurso web com uma lista de páginas HTML conhecidas: .html, .htm, .asp, .aspx, .php, e uma barra.

Alguns crawlers podem evitar requisições que possuem uma interrogação ("?") na URL (são dinamicamente produzidos) de modo a evitar armadilhas para spiders que podem causar um loop infinito de URLs para captura dentro de um web site.

3.3.3 Crawling de caminho ascendente

Alguns crawlers pretendem capturar todos os recursos que forem disponíveis de um web site particular. Cothey (2004) introduziu o crawler de caminho ascendente que deve ascender de qualquer caminho em cada URL que pretende capturar. Por exemplo, quando dada uma URL semente http://llama.org/hamster/monkey/page.html, ele tentará capturar /hamster/monkey/, /hamster/, e /. Cothey descobriu que o crawler de caminho ascendente foi muito efetivo em encontrar recursos isolados, ou recursos que não possuem *links* apontando para eles que sejam encontrados no crawling regular.

Muitos crawlers de caminho ascendente são conhecidos como softwares de colheita, pois eles usualmente "colhem" ou colecionam todo o conteúdo - talvez uma coleção de fotos em uma galeria - de uma página específica ou host.

3.3.4 Crawling Focado

A importância de uma página para um crawler pode ser expressa através de uma função de similaridade do conteúdo com relação ao conteúdo desejado. Os web crawlers que se propõem a baixar páginas similares são chamados de crawlers

focado ou crawlers de tópico. O conceito de crawler focado e crawler de tópico foram primeiramente introduzidos por Menczer (1997; Menczer and Belew, 1998) e por Chakrabarti *et al.* (1999).

O maior problema do crawling focado é que no contexto de web crawlers, gostaríamos de prever a similaridade do conteúdo de uma determinada página com relação ao conteúdo desejado antes de baixar a página. Um possível indicativo é o texto âncora do link; esta foi a aproximação feita por Pinkerton (1994) em um crawler desenvolvido no início da web. Diligenti *et al.* (2000) propuseram o uso do conteúdo completo das páginas já visitadas para inferir na similaridade do entre o conteúdo desejado e as páginas que não foram visitadas ainda. O desempenho do crawling focado depende em sua maioria da riqueza dos *links* em um determinado tópico a ser buscado, e o crawling focado sempre confia em um Web *Search engine* para prover os pontos de partida (sementes).

3.3.5 Fazendo Crawling na Web profunda

Uma grande quantidade de páginas encontra-se na web profunda ou na web invisível. Estas páginas normalmente só são acessíveis submetendo várias consultas ao banco de dados. O crawler regular não é capaz de encontrar estas páginas, pois, em sua maioria, não possuem *links* apontando para elas. O Google Sitemap Protocol e o Mod ai (Nelson *et al.*, 2005) propõem-se a descobrir esta web profunda através de uma listagem, fornecida pelo webmaster, de todas as páginas encontradas em seu website.

3.3.6 Política de re-visitação

A web possui uma natureza dinâmica, e realizando o crawling de uma fração da web pode levar uma grande quantidade de tempo, usualmente medido em semanas ou meses. Assim que um web crawler termina o crawling, muitos eventos podem ocorrer. Estes eventos podem incluir: criação, deleção e atualização.

Do ponto de vista dos buscadores, existe um custo associado na não detecção de um evento, e assim possuindo uma cópia desatualizada da fonte. As funções de custo mais usadas, introduzidas por Cho e Garcia-Molina(2000), são atualidade e idade.

Atualidade: É uma medida binária que indica se a cópia local é precisa ou não. A atualidade de uma página *p* no repositório no tempo *t* é definida como:

Ilustração 4 – Equação de Atualidade

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Fonte: Synchronizing a database to improve freshness - http://www.cs.brown.edu/courses/cs227/2002/cache/Cho.pdf - Visitado em 2/11/2007

Idade: Esta é a medida que indica quão antiquado a cópia local esta. A idade de uma página *p* no repositório, no tempo *t* é definida como:

Ilustração 5 – Equação de Idade

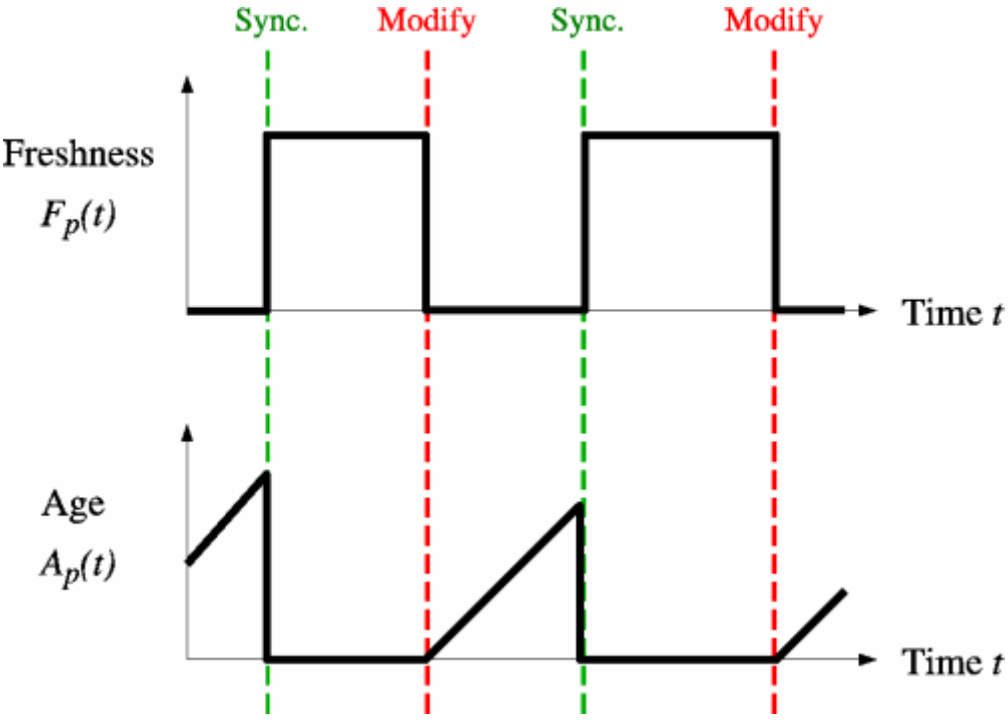
$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases}$$

Fonte: Synchronizing a database to improve freshness - http://www.cs.brown.edu/courses/cs227/2002/cache/Cho.pdf - Visitado em 2/11/2007

Coffman *et al.* (1998) trabalharam com uma definição do objetivo do web crawler que é equivalente à atualidade, mas utiliza de palavras diferentes: eles propuseram que o crawler deve minimizar a fração do tempo das páginas que ficam antiquadas. Eles notaram também que o problema do *web crawling* pode ser modelado como uma fila múltipla, de um servidor único recebendo votos do sistema, onde o web crawler é o servidor e os websites são as filas (Ilustração 6). A modificação de página é a chegada de novos clientes, e o interruptor é o intervalo entre o acesso de páginas de um único website.

O objetivo do crawler é manter a média de atualização das páginas em sua coleção o mais alto possível, ou manter a média de idade das páginas o mais baixo possível. Estes objetivos não são equivalentes: no primeiro caso, o crawler está preocupado apenas com a quantidade de páginas que estão desatualizadas, enquanto no segundo caso, o crawler está preocupado em quão antiquadas as cópias locais são.

Ilustração 6 – Evolução da Atualidade e Idade em *Web crawling*



Fonte: Synchronizing a database to improve freshness - http://www.cs.brown.edu/courses/cs227/2002/cache/Cho.pdf - Visitado em 2/11/2007

Duas políticas simples de visitação foram estudadas por Cho e Garcia-Molina (Cho and Garcia-Molina, 2003):

Política uniforme: envolve a re-visitação de todas as páginas da coleção com a mesma freqüência, indiferentemente da sua taxa de mudança.

Política proporcional: envolve a maior re-visitação das páginas que são modificadas freqüentemente. A freqüência de visitação é diretamente proporcional (estimado) à freqüência de mudança.

Em ambos os casos, o crawling repetitivo das páginas pode ser feito em ordem randômica ou fixa.

Cho and Garcia-Molina provaram um importante resultado, que em termos de média de atualização, a política uniforme ultrapassa a política proporcional nas duas simulações de crawling da Web e Web real. A explicação para este resultado vem do fato que, quando uma página muda freqüentemente, o crawler vai perder tempo tentando fazer o crawling novamente tão rápido e não vai conseguir manter uma cópia bem atualizada.

Para melhorar a atualização, devemos penalizar os elementos que mudam freqüentemente (Cho and Garcia-Molina, 2003a). A política de re-visitação ótima não é a política uniforme nem a política proporcional. O melhor método para manter a média de atualização alta é ignorar as páginas que mudam muito, e a melhor para manter a média de idade mais baixa é usar de freqüência de acesso monolíticamente (e sub-linearmente) que crescem com a taxa de mudanças de cada página. Em ambos os casos, o ótimo está mais próximo da política uniforme do que da política proporcional: como Coffman *et al.* (1998) descreveram, “em desejo de minimizar o esperado tempo obsoleto, o acesso para qualquer página deve ser mantido, o mais possível, uniformemente espaçado”.

3.3.7 Política de Cortesia

Os crawlers podem obter mais informação de forma mais rápida e profunda do que pesquisadores humanos, logo eles podem afetar o desempenho de um website. É desnecessário dizer que, se um único crawler está realizando múltiplas requisições por segundo e/ou obtendo grandes arquivos, um servidor deverá ficar uma grande quantidade de tempo mantendo as requisições de múltiplos crawlers.

Como observado por Koster (1995), o uso de web crawlers é útil para uma grande quantidade de tarefas, mas vem acompanhado com um preço para toda a comunidade. Os custos de uso de um web crawler incluem:

- Recursos de rede, os crawlers requerem uma considerável quantidade de largura de banda e operam com um alto degrau de paralelismo durante um longo tempo.
- Sobrecarga de servidor, especialmente se a freqüência de acesso em um determinado servidor é muito alta.
- Crawlers mal programados, que podem travar servidores e roteadores, ou podem obter páginas que não podem tratar.
- Crawlers pessoais que, se usado por vários usuários, podem causar a queda de uma rede ou um servidor web.

Uma solução parcial para estes problemas é o protocolo de exclusão de robôs, ou em inglês “robots exclusion protocol”, também conhecido como protocolo

robots.txt (Koster, 1996) que é a padronização para administradores com a finalidade de indicar quais as partes dos seus servidores web não deve ser acessado pelos robôs de busca. Este padrão não inclui uma sugestão de intervalo de visitação para o mesmo servidor, embora este intervalo seja a forma mais efetiva de evitar a sobrecarga do servidor. Recentemente *search engines* comerciais como Ask Jeeves, MSN e Yahoo são capazes de utilizar um parâmetro "crawl-delay" no arquivo robots.txt para indicar o número de segundo entre requisições.

A primeira proposta de intervalo entre conexões foi estipulada por Koster (1993) e era de 60 segundos. Contudo, se páginas são obtidas neste ritmo de um website com mais de 100.000 páginas com uma conexão perfeita de latência zero e infinita largura de banda, pode levar até 2 meses para obter todo o website; também, somente uma fração de recursos daquele servidor web será usado. Isto não parece ser aceitável.

Cho (2003) usa 10 segundos como intervalo de acesso, e o crawler WIRE (Baeza-Yates and Castillo, 2002) usa 15 segundos como default. O crawler MercatorWeb (Heydon and Najork, 1999) segue uma política de cortesia adaptativa: se demora-se *t* segundos para obter um documento de um determinado servidor, o crawler aguarda *10t* segundos antes de requisitar a próxima página. Dill *et al.* (2002) usam 1 segundo.

Uma análise dos logs de acesso mostra que o intervalo dos crawlers conhecidos é entre 20 segundos e 3-4 minutos. Vale notar que mesmo sendo muito cortês, e tomando todas as precauções para não haver sobrecarga do servidor web, algumas reclamações são recebidas dos administradores dos servidores web. Brin e Page, os fundadores do Google, notaram o seguinte: "rodando um crawler que conecta-se em mais do que meio milhão de servidores gera uma quantidade razoável de emails e telefonemas. Por causa da grande quantidade de pessoas que ficam on line, sempre existem aqueles que não sabem o que é um crawler, por que esta é a primeira vez que vêem um." (Brin and Page, 1998).

3.3.8 Política de Paralelização

Um crawler paralelo é um tipo de crawler que roda múltiplos processos em paralelo. O objetivo é aumentar a taxa de obtenção de arquivos com a

minimização da sobrecarga da paralelização e não obter o mesmo arquivo várias vezes. Para não obter a mesma página mais de uma vez, o sistema de crawling requer uma política de assinatura das URLs novas descobertas durante o processo de crawling, pois a mesma URL pode ser encontrada por dois diferentes processos de crawling.

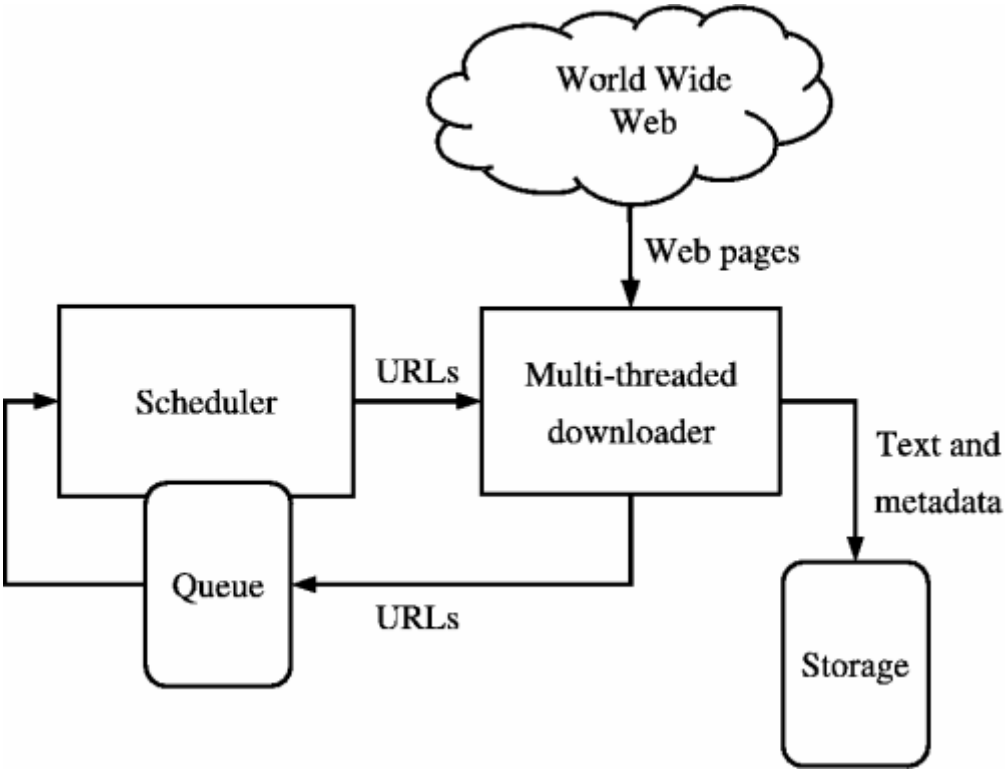
3.4 Arquitetura de Web Crawlers

Um crawler deve não ter somente uma boa estratégia de crawling, como discutido nos itens anteriores, mas deve possuir também uma arquitetura(Ilustração 7) muito otimizada.

Shkapenyuk e Suel (2002) observaram que: “Enquanto é fácil construir um crawler lento para obter uma pequena quantidade de páginas por segundo em um pequeno período de tempo, construir um sistema de alto desempenho que pode obter centenas de milhares de páginas em várias semanas apresenta um desafio no design do sistema, I/O e eficiência de rede, robustez e maleabilidade”.

Os web crawlers são a parte central dos *search engines*, e detalhes de seus algoritmos e arquitetura são mantidos como segredo de negócios. Quando o design do crawler é publicado, sempre é omitida uma parte da informação que previne a réplica do sistema por outros.

Ilustração 7 - Arquitetura de alto-nível de um Web Crawler padrão



Fonte: Design and implementation of a high performance distributed web crawler - http://cis.poly.edu/tr/tr-cis-2001-03.pdf - Visitado em 2/11/2007

3.4.1 Normalização de URL

Os crawlers normalmente executam algum tipo de normalização de URLs para que não haja a obtenção de uma mesma página mais de uma vez. O termo *normalização de URL*, também chamado de *canonização de URL*, refere-se ao processo de modificar e padronizar a URL de uma maneira consistente. Existem vários tipos de normalização que podem ser executadas, incluindo a conversão da URL para letras minúsculas, remoção de segmentos "." e "..", e adicionando barras inclinadas nos componentes do caminho não vazios (Pant *et al.*, 2004).

3.5 Identificação de Web Crawlers

Os web crawlers tipicamente identificam-se a um servidor web pelo uso de um campo chamado "User-agent" na requisição HTTP. Os administradores de servidores web normalmente examinam os logs dos servidores e utilizam o campo "User-agent" para determinar quais crawlers visitaram o servidor web e com qual freqüência. O campo "User-agent" pode conter ainda uma URL onde o administrador

do servidor web pode encontrar maiores informações sobre o web crawler. Spambots e outros crawlers maliciosos normalmente não colocam informações de identificação no campo “User-agent”, ou mascaram a sua identificação como um navegador ou um crawler conhecido.

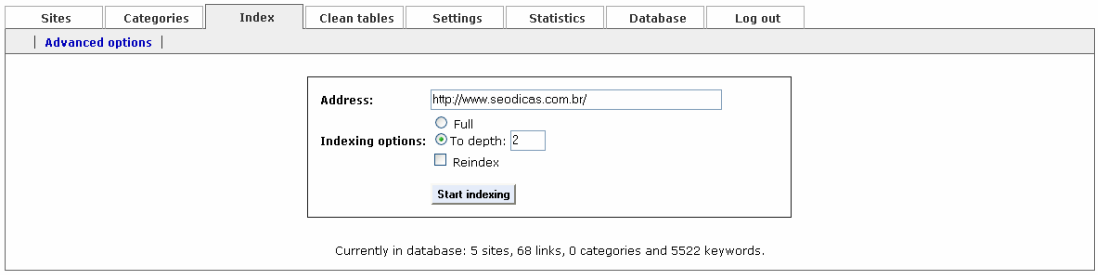
É muito importante que os web crawlers identifiquem-se, pois assim os administradores de servidores web podem entrar em contato caso seja necessário. Em alguns casos, os crawlers podem cair, acidentalmente, em armadilhas para crawlers ou eles podem estar sobrecarregando o servidor web com requisições, então o responsável precisa parar o crawler. A identificação também é útil para os administradores que desejam saber quando as suas páginas serão indexadas por um particular *search engine*.

3.6 Sphider e o seu processo de *Web crawling*

O processo de *Web crawling* do *search engine* Sphider possui uma interface de controle onde o administrador possui controle sobre as operações envolvidas.

Para requisitar que um web site seja inserido no sistema, deve-se inserir no formulário de indexação a URL desejada, servindo como semente para alimentar o sistema. Deve-se definir também, a profundidade com que o *Web crawling* será realizado. Logo após, deve-se pressionar o botão “Start Indexing”.

Ilustração 8 – Sphider: Formulário de Indexação de URL



Com isto, o sistema inciará o crawler, requisitando as páginas da URL informada no formulário.

Ilustração 9 - Sphider: Processo de *Web crawling*


[Back to admin]

Spidering **http://www.seodicas.com.br/**

Disallowed files and directories in robots.txt:
http://www.seodicas.com.br/wp-admin/
http://www.seodicas.com.br/author/
http://www.seodicas.com.br/comments/
http://www.seodicas.com.br/category/
http://www.seodicas.com.br/2007/
http://www.seodicas.com.br/2008/

1. Retrieving: **http://www.seodicas.com.br/** at 17:37:04.
Size of page: 82.33kb. Starting indexing at 17:37:10.
Indexed
Links found: **57**. New links: **57**
2. Retrieving: **http://www.seodicas.com.br/** at 17:37:38.
already in database
3. Link **http://www.seodicas.com.br/2007/09**: file checking forbidden in robots.txt file
4. Link **http://www.seodicas.com.br/2007/10**: file checking forbidden in robots.txt file
5. Link **http://www.seodicas.com.br/author/fabioricotta/**: file checking forbidden in robots.txt file
6. Link **http://www.seodicas.com.br/author/frank/**: file checking forbidden in robots.txt file
7. Link **http://www.seodicas.com.br/author/heron/**: file checking forbidden in robots.txt file
8. Link **http://www.seodicas.com.br/author/rafael/**: file checking forbidden in robots.txt file
9. Retrieving: **http://www.seodicas.com.br/black-hat/cloaking** at 17:37:38.
Size of page: 21.71kb. Starting indexing at 17:37:41.
Indexed
Links found: **44**. New links: **2**
10. Retrieving: **http://www.seodicas.com.br/black-hat/cloaking-e-geo-targeting** at 17:37:44.
Size of page: 22.52kb. Starting indexing at 17:37:48.
Indexed
Links found: **44**. New links: **2**
11. Retrieving: **http://www.seodicas.com.br/black-hat/seo-black-hat** at 17:37:50.
Size of page: 27.24kb. Starting indexing at 17:37:53.
Indexed
Links found: **44**. New links: **3**
12. Retrieving: **http://www.seodicas.com.br/black-hat/textos-com-fundo-black-hat-ou-nao** at 17:37:59.
Size of page: 27.59kb. Starting indexing at 17:38:02.
Indexed
Links found: **46**. New links: **2**
13. Retrieving: **http://www.seodicas.com.br/breadcrumb-navigation/breadcrumb-navigation-nao-perca-o-cliente-do-seu-site** at 17:38:08.
Size of page: 25.53kb. Starting indexing at 17:38:11.
Indexed
Links found: **45**. New links: **4**
14. Link **http://www.seodicas.com.br/category/ajax-seo**: file checking forbidden in robots.txt file
15. Link **http://www.seodicas.com.br/category/black-hat**: file checking forbidden in robots.txt file
16. Link **http://www.seodicas.com.br/category/breadcrumb-navigation**: file checking forbidden in robots.txt file
17. Link **http://www.seodicas.com.br/category/conteudo-duplicado**: file checking forbidden in robots.txt file
18. Link **http://www.seodicas.com.br/category/ferramentas**: file checking forbidden in robots.txt file
19. Link **http://www.seodicas.com.br/category/frames-seo**: file checking forbidden in robots.txt file
20. Link **http://www.seodicas.com.br/category/geral**: file checking forbidden in robots.txt file
21. Link **http://www.seodicas.com.br/category/google-seo**: file checking forbidden in robots.txt file

O primeiro passo do crawler do Sphider é verificar se a URL está acessível, tentando conectar ao servidor e, logo em seguida, verificando se o arquivo existe. Em seguida, o crawler tenta obter o arquivo robots.txt, para verificar quais páginas podem ser capturadas, e caso ele não seja encontrado, todas as páginas estarão liberadas para a captura.

Em seguida, ele captura a página especificada no formulário inicial, verifica se é uma página HTML, localizando todos os *links* contidos nela e comparando-os com *links* já existentes no banco de dados, estipulado, quantos são os novos *links*. Caso não seja uma página HTML, por exemplo, um arquivo DOC, ele envia o arquivo para o analisador responsável pelos arquivos DOC.

É importante verificar que, caso a página não possa ser capturada devido às regras no arquivo robots.txt, ela terá o seu conteúdo ou *links* inseridos do banco de dados.

Pude notar em alguns testes, que para sites com mais de 20 páginas, o Sphider passa a ter um grande funil na obtenção das páginas: o seu crawler. Ele indexa apenas uma página por de cada vez, não possuindo paralelismo na captura.

Este paralelismo poderia ser obtido através do uso de uma classe de Multi Threading criada por Mohammed Yousef (2007), que faz uso do paralelismo através de uma classe em PHP.

Outro ponto importante que foi observado, é que a programação do web crawler e do indexador é feita no mesmo arquivo PHP, sem modularização das funções e separação em arquivos. Esta alteração é necessária para o aumento de escalabilidade do sistema e produzir uma maior capacidade de alteração nos arquivos.

4 INDEXAÇÃO WEB

Indexação por *search engines* é definida por como a informação é coletada, analisada, e armazenada para facilitar a recuperação rápida e precisa da informação. O projeto do índice incorpora conceitos interdisciplinares de lingüística, psicologia cognitiva, matemática, informática, física e ciência da computação. Um nome alternativo para o processo é “Web Indexing”, que está contido no contexto projeto de *search engines* para encontrar páginas na internet.

Sistemas populares focam na indexação de textos completos de documentos de linguagem natural na Web, já que existem outro tipo de mídia buscável como vídeo e áudio, e imagens.

4.1 Indexação

O objetivo de armazenar um índice é de otimizar a velocidade e o desempenho na busca por um documento relevante devido a um termo buscado. Sem um índice, o *search engine* deveria pesquisar em cada corpo de documento, o qual pode levar uma boa quantidade de tempo e poder de computação. Por exemplo, um índice com 10.000 documentos pode ser pesquisado em milisegundos, enquanto a pesquisa seqüencial de cada palavra em 10.000 documentos grandes pode levar horas. O custo pelo ganho de tempo durante a recuperação de informação é o espaço de armazenamento computacional adicional necessário para armazenar o índice e um considerável aumento no tempo requerido para uma atualização acontecer.

4.1.1 Fatores do Projeto do Índice

Ao se projetar um índice para search engines, alguns fatores devem ser considerados, a fim de obter os melhores desempenhos e oferecer uma boa qualidade. A maioria dos fatores do projeto de uma arquitetura de *search engine* inclui:

4.1.1.1 Fatores de combinação

Os fatores de combinação são associados a como a informação entra no índice, ou como palavras ou característica de assuntos são adicionadas durante a tradução do corpo do texto e se múltiplos índices podem trabalhar assincronamente. O indexador deve verificar primeiro se está atualizando um conteúdo antigo ou adicionando um novo conteúdo.

4.1.1.2 Técnicas de armazenamento

As técnicas de armazenamento consistem em como armazenar a informação do índice – se a informação pode ser comprimida ou filtrada.

4.1.1.3 Tamanho de índice

O tamanho do índice é referido como quanto espaço computacional é necessário para suportar o índice.

4.1.1.4 Tempo de busca

Entende-se por tempo de busca, quão rápida a informação pode ser obtida no índice invertido. Quão rápido uma entrada na estrutura de dados pode ser encontrada, versus quão rápido ela pode ser atualizada ou removida.

4.1.1.5 Manutenção

Manutenção do índice no decorrer do tempo.

4.1.1.6 Tolerância a falhas

Quão importante é a necessidade de o serviço ser confiável, como lidar com dados corrompidos, se o dado ruim pode ser tratado isoladamente, negociando com hardware ruim, particionando esquemas tais como hash-based ou particionamento composto, replicação.

4.1.2 Estruturas de Índices de Dados

A arquitetura dos *search engines* varia em como a indexação é realizada e no armazenamento do índice para conhecer vários fatores de projeto. Tipos de índices incluem:

4.1.2.1 Árvores de sufixo

Pictoricamente estruturado como uma árvore possui tempo de busca linear e pode é construída usando-se o sufixo das palavras. As árvores de sufixos são usadas também para procurar padrões em seqüências de DNA e clusterização. A maior desvantagem é que o armazenamento de uma palavra na árvore pode gastar mais espaço do que o próprio armazenamento da palavra. Uma representação alternativa é um vetor

de sufixos, é considerado que se requer menos memória virtual e possui suporte a compressão de dados, como por exemplo, BTW.

4.1.2.2 Árvores

Uma estrutura ordenada de árvore de dados é usada para armazenar um vetor associativo onde as chaves são strings. Considerado mais rápido do que uma tabela hash, mas possui pouca eficiência no espaço ocupado.

4.1.2.3 Índices invertidos

Armazena uma lista de ocorrência de cada critério atômico de busca, tipicamente em forma de uma tabela hash ou de uma árvore binária.

4.1.2.4 Índices de citação

Armazena a existência de citações ou hiperlinks entre documentos para suportar análise de citação.

4.1.2.5 Índices Ngram

Para armazenar seqüências do tamanho dos dados para suportar outros tipos de recuperação ou mini texto.

4.1.2.6 Matriz de termos do documento

Usado em análise semântica latente, armazena a ocorrência de palavras de um documento em uma matriz escassa.

4.1.3 Desafios no Paralelismo

O maior desafio no projeto de *search engines* é o gerenciamento de processos computacionais paralelos. Existem muitas oportunidades para condições de corrida e falhas coerentes. Por exemplo, um novo documento é adicionado ao corpo e o índice necessita ser atualizado, mas ao mesmo tempo o índice necessita continuar respondendo aos termos das buscas. Esta é uma colisão entre duas tarefas competidoras. Considere que os autores são produtores de informação, e o web crawler é o consumidor de informação, pegando o texto e armazenando em cache (ou corpo). O índice dianteiro é o consumidor da informação produzida pelo corpo, e o índice invertido é o consumidor da informação produzida pelo índice dianteiro. Isto é comumente referido

como um modelo produtor-consumidor. O indexador é o produtor da informação pesquisável e os usuários são os consumidores que necessitam buscar. O desafio é aumentado quando se trabalha com armazenamento distribuído e processamento distribuído. Em um esforço para trabalhar com grande quantidade de informação indexada, a arquitetura do *search engine* deve envolver computação distribuída, onde o *search engine* constituído de vários computadores trabalhando em união. Isto aumenta a possibilidade de incoerências e faz com que seja mais difícil manter uma arquitetura totalmente sincronizada, distribuída e paralela.

4.1.4 Índices invertidos

Muitos *search engines* incorporaram um índice invertido quando avaliam um termo buscado para localizar rapidamente documentos contendo as palavras do termo buscado e classificar estes documentos por relevância. O índice invertido armazena uma lista de documentos que contém cada palavra. O *search engine* pode retornar rapidamente os documentos correspondentes usando acesso direto para encontrar os documentos associados com a palavra buscada. A tabela abaixo é uma ilustração simplificada de um índice invertido:

Tabela 3 – Tabela de índice invertido

Palavra	Documentos
a	Documento 1, Documento 3, Documento 4, Documento 5
vaca	Documento 2, Documento 3, Documento 4
diz	Documento 5
moo	Documento 7

Usar este índice pode determinar apenas se uma determinada palavra existe em um particular documento, ele não armazena informações sobre a freqüência e posição da palavra e é considerado então um índice booleano. Tal índice pode apenas servir para determinar quais documentos coincidem com a busca, mas podem não contribuir para classificar os documentos encontrados. Em alguns projetos o índice inclui uma informação adicional tal como a freqüência de cada palavra em cada documento ou as posições da palavra no documento. Com a posição, o algoritmo de

busca pode identificar a proximidade da palavra para possuir busca por frases. A freqüência pode ser usada para ajudar na classificação da relevância dos documentos dada uma busca. Tais tópicos são os focos centrais de pesquisa da recuperação da informação.

4.1.5 Fusão de índice

O índice invertido é preenchido via fusão ou reconstrução. A reconstrução é similar à fusão, mas primeiro exclui o conteúdo do índice invertido. A arquitetura deve ser projetada para suportar índice incremental, onde a fusão envolve identificação do documento ou documentos para adicionar ou atualizar o índice e analisando gramaticalmente cada documento em palavras. Para precisão técnica, uma fusão envolve a união de documentos recentemente indexados, tipicamente residindo na memória virtual, com o cachê do índice residindo em um ou mais disco rígidos.

Após a análise gramatical, o indexador adiciona o documento na lista de documentos para a palavra apropriada. O processo de encontrar cada palavra no índice invertido para indicar que existe uma ocorrência dentro de um documento pode gerar um alto tempo de consumo quando projetado para um grande *search engine*, então este processo é comumente dividido em um desenvolvimento de um índice dianteiro e o processo de ordenação do conteúdo do índice avançado para entrada no índice invertido. O índice invertido é chamado invertido por ser uma inversão do índice avançado.

4.1.6 Índice avançado

O índice avançado armazena uma lista de palavras para cada documento. A tabela abaixo é uma ilustração simplificada de um índice avançado:

Tabela 4 – Tabela de índice avançado

Documento	Palavras
Documento 1	a, vaca, diz, moo
Documento 2	o, boi, e, a, vaca
Documento 3	vaca, come, no, pasto

A razão por trás de desenvolver um índice avançado é que os documentos são analisados gramaticalmente, é melhor armazenar imediatamente as palavras por documento. O índice avançado é ordenado para ser transformado em índice invertido. O índice avançado é essencialmente uma lista de pares, consistido de um documento e uma palavra, colecionado pelo documento. Converter o índice avançado a um índice invertido consiste em ordenar os pares pelas palavras. Neste contexto, o índice invertido é um índice avançado ordenado pelas palavras.

4.1.7 Compressão

Gerar ou manter um índice de um *search engine* de larga escala representa um desafio de processamento e armazenamento. Muitos *search engines* utilizam uma forma de compressão para reduzir o tamanho dos índices no disco. Considere o seguinte cenário para um texto completo, internet e um *search engine*.

- Foi estimado (Sergey Brin and Lawrence Page, 1998) que em 2000 existisse 2.000.000.000 diferentes páginas web;
- Uma estimativa fictícia de 250 palavras, em média, por página web, baseado na suposição similaridade com páginas de um romance.
- O custo de armazenamento de um caractere é de 8 bits (1 byte). Algumas codificações utilizam 2 bytes por caractere.
- A estimativa de média de caracteres em uma palavra é 5.
- Um computador pessoal possui uma média de 20 gigabytes de armazenamento útil.

Dadas estas estimativas, a geração de um índice não comprimido para 2 bilhões de páginas necessitaria armazenar 5 bilhões de palavras. Sendo 1 byte por caractere, ou 5 bytes por palavra, isto requer 2.500 gigabytes de espaço vazio, mais do que a média de espaço que um computador pessoal possui. Este espaço cresce no caso de uma arquitetura de armazenamento com tolerância à falhas. Usando compressão, o tamanho do índice pode ser reduzido a uma porção deste tamanho, dependendo de quais técnicas de compressão forem adotadas. O ponto crucial é o tempo e processamento necessário para executar a compressão e descompressão.

Notavelmente, o projeto de um *search engine* de larga escala inclui o custo de armazenamento, e o custo de energia elétrica para o armazenamento. Compressão, neste caso, é uma boa medição de custo.

4.2 Análise Gramatical de Documento

A análise gramatical de documento envolve a quebra de componentes (palavras) de um documento ou outra forma de mídia para inserção em um índice avançado ou invertido. Por exemplo, se o conteúdo completo de um documento consiste na sentença “Hello World”, haveria duas palavras encontradas tipicamente, o token “Hello” e o token “World”. No contexto de indexação do *search engine* e no processamento da linguagem natural, a análise gramatical é mais comumente referenciada como tokenização ou ainda como análise léxica.

O processamento da linguagem natural, a partir de 2006, é um tema de continua pesquisa e inovações tecnológicas. Existe uma série de desafios na tokenização, na extração da informação necessária na indexação de documentos para possuir uma boa qualidade de pesquisa. A tokenização para indexação envolve múltiplas tecnologias, a implementação delas é comumente mantida como segredo de negócio.

4.2.1 Desafios no Processamento do Idioma Natural

Limitação de ambigüidade de palavras – nativos em inglês podem, em um primeiro momento, considerar que a tokenização seja uma tarefa direta, mas não é tão direto quanto projetar um indexador multilíngüe. Na forma digital, o texto de outros idiomas tais como o chinês, o japonês ou o árabe representa grandes desafios com as palavras, pois não são claramente delineados por um espaço em branco. O objetivo na tokenização é encontrar palavras as quais os usuários irão buscar. A lógica específica do idioma é usada para identificar corretamente os limites da palavra, que é a freqüentemente a razão de se projetar um analisador para cada idioma suportado (ou para grupos de linguagens com marcadores de limite ou sintaxe similares).

Ambigüidade de idioma – para ajudar na classificação correta de documentos, muitos *search engines* coletam informações adicionais sobre palavras, tais como sua categoria lingüística ou léxica. Estas técnicas são dependentes do idioma, pois cada sintaxe varia a cada idioma. Os documentos nem sempre identificam claramente o

idioma do documento ou representam ela com precisão. Na tokenização do documento, alguns *search engines* tentam identificar automaticamente o idioma do documento.

Diversidade de formato de arquivos – de modo a identificar corretamente quais bytes de um arquivo representam um caractere, o formato do arquivo deve ser controlado corretamente. *Search engines* que suportam vários tipos de arquivos devem ser capazes de abrir e acessar corretamente o documento e ser capazes de tokenizar os caracteres do documento.

Armazenamento defeituoso – a qualidade dos dados do idioma natural não é sempre considerada perfeita. Uma quantidade não especificada de documentos, particularmente na internet, não obedece fielmente o protocolo do arquivo. Caracteres binários podem ser codificados acidentalmente em várias partes do documento. Sem o reconhecimento destes caracteres e controle apropriado, a qualidade do índice ou o desempenho do indexador pode degradar.

4.2.2 Tokenização

Diferentemente de humanos adultos alfabetizados, computadores não estão inerentemente atentos ao idioma natural do documento e não reconhecem automaticamente palavras e sentenças. Para um computador, um documento é somente uma grande quantidade de bytes. Computadores não sabem que um espaço entre duas seqüências de caracteres significa que existem duas palavras separadas no documento. Uma saída foi um programa de computador desenvolvido por seres humanos para treinar o computador, ou instruir o computador, em como identificar o quê constitui uma palavra individual ou distinta, referenciado como um token. Este programa é comumente referenciado como tokenizador ou parser ou lexer. Muitos *search engines*, assim como outros softwares de processamento de idiomas naturais, incorporam programas especializados em análise gramatical, tais como YACC ou Lex.

Durante a tokenização, o analisador gramatical identifica uma seqüência de caracteres, que tipicamente representa palavras. Tokens comumente reconhecidos incluem pontuação, seqüências de caracteres numéricos, caracteres do alfabeto, caracteres alfanuméricos, caracteres binários (espaço, nulo, imprimir, ou outro comando de impressão antiquado), espaço em branco (espaço, tab, enter ou return, alimentador de linha) e entidades como endereço de emails, números de telefone, e

URLs. Na identificação de cada token, várias características podem ser armazenadas tais como a formatação do token (maiúsculas, minúsculas, mixado, etc), idioma e codificação, categoria léxica (nome ou verbo, por exemplo), posição, número da sentença, posição da sentença, tamanho, e número da linha.

4.2.3 Reconhecimento do idioma

Se os *search engines* suportam múltiplos idiomas, o passo inicial comum durante a tokenização é a identificação do idioma de cada documento, dado que muitos dos posteriores passos são dependentes do idioma. O reconhecimento do idioma pelo qual um programa de computador tenta identificar automaticamente, ou categorizar, o idioma do documento. Outros nomes para o reconhecimento do idioma são: classificação de idioma, análise de idioma, identificação de idioma, e etiquetagem de idioma. O reconhecimento de idioma automático é tema de pesquisa contínua no processamento de idioma natural. Encontrar qual idioma a palavra pertence pode envolver o uso de uma tabela de reconhecimento de idioma.

4.2.4 Análise de Formato

Dependendo se o *search engine* suporta múltiplos formatos de documento, os documentos devem ser preparados para a tokenização. O desafio é que muitos formatos de documentos, em adição ao conteúdo textual, possuem informação de formatação. Por exemplo, documentos HTML contêm tags HTML, as quais especificam a informação de formatação, como por exemplo, onde começar uma nova linha, ou disponibilizar uma palavra em negrito, ou mudar o tamanho ou família da fonte. Se o *search engine* ignorar a diferença entre conteúdo e marcação, os segmentos serão incluídos no índice, conduzindo a péssimos resultados de busca. A análise de formato envolve a identificação e controle do conteúdo formatado embutido nos documentos que controlam como o documento é renderizado em uma tela de computador ou interpretado por um software. A análise de formato também é conhecida como análise de estrutura, análise gramatical do formato, remoção de tags, normalização de texto, limpeza de texto, ou preparação de texto. O desafio de análise de formato é complicado posteriormente pela complexidade de vários formatos de arquivo. Alguns formatos de arquivo são proprietários e pouca informação é liberada, enquanto

outras são bem documentadas. Formatos de arquivos comuns, bem documentados, que muitos *search engines* suportam incluem:

- Microsoft Word
- Microsoft Excel
- Microsoft Power Point
- IBM Lotus Notes
- HTML
- ASCII arquivos texto (documento sem formatação)
- Adobe's Portable Document Format (PDF)
- PostScript (PS)
- LaTex
- XML e derivados como o RSS
- SGML (é mais um protocolo geral)
- Formatos meta dados multimídia como o ID3

Técnicas para trabalhar com vários formatos incluem:

- Uso de uma ferramenta de análise gramatical comercial publicamente conhecida que é disponibilizada pela organização que desenvolveu, mantém, e é dona do formato.
- Escrever um analisador gramatical próprio.

Alguns *search engines* suportam inspeção em arquivos que são armazenados em formato compactado ou encriptado. Se estiver trabalhando com formato compactado, então o indexador primeiro descompacta o documento, que pode resultar em um ou mais arquivos, onde cada qual será indexado separadamente. Alguns formatos comuns de arquivos compactados incluem:

- ZIP
- RAR
- CAB

- Gzip
- BZIP
- TAR, GZ, e TAR.GZ

A análise de formato pode envolver métodos de melhoria de qualidade para evitar incluir “informação ruim” no índice. O conteúdo pode manipular a informação de formatação para incluir um conteúdo adicional. Exemplos de formatação de documento abusiva para spamindexing;

- Incluir dezenas de palavras em uma seção que fica escondida na tela do computador, mas é visível ao indexador, através do uso de formatação (ex. uma tag “div” no HTML, que pode ser escondida com o uso de CSS ou Javascript).
- Colocar a cor da fonte do texto igual à cor de fundo da página, fazendo com que as palavras fiquem “invisíveis” à tela de computador de uma pessoa que está visitando a página, mas não está invisível ao indexador.

4.2.5 Reconhecimento de seção

Alguns *search engines* incorporam o reconhecimento de seção, a identificação das principais partes de um documento, prioritárias na tokenização. Nem todos os documentos não possuem o corpo como um livro bem escrito, dividido em organizados capítulos e páginas. Muitos documentos na web possuem conteúdo errôneo e seções que não contém material primário, no qual se descreve sobre o que trata o documento, tais como newsletters e relatório de corporações. Por exemplo, um blog pode disponibilizar um menu lateral com palavras nos *links* apontando para outras páginas web. Alguns formatos de arquivo, como HTML e PDF, permitem que o conteúdo possa ser mostrado em colunas. Embora o conteúdo seja mostrado, ou renderizado, em diferentes áreas de visão, a real marcação de conteúdo pode armazenar esta informação sequencialmente. As palavras que aparecem seqüencialmente no real conteúdo são indexadas seqüencialmente, embora estas sentenças e parágrafos sejam renderizados em diferentes partes da tela do computador. Se os *search engines* indexam este conteúdo como se fosse um conteúdo normal um dilema resulta onde a qualidade do índice é degradada e é degradada a qualidade da busca devido ao conteúdo misturado e a imprópria proximidade de palavra. Dois problemas primários são notados:

- Conteúdos em diferentes seções são tratados como relacionados no índice, mas na realidade não são.
- O conteúdo da barra lateral é incluído no índice, mas não contribui para o significado do documento, e o índice é preenchido com uma representação pobre do documento, assumindo que a meta é perseguir o significado de cada documento, uma submeta de prover resultados de busca com qualidade.

A análise de seção pode requerer que o *search engine* implemente uma lógica de renderização de cada documento, essencialmente uma representação abstrata do documento atual, e então indexar esta representação. Por exemplo, algum conteúdo na internet é renderizado via Javascript. Visitantes destas páginas web vêem este conteúdo em seus navegadores web. Se o *search engine* não renderiza esta página e avalia o Javascript dentro desta página, ele não irá ver o conteúdo da mesma forma, indexando o documento de forma errônea. Dado isso, alguns *search engines* não se importam com problemas de renderização, muitos web designers evitam exibir conteúdo via Javascript ou usam a tag Noscript para garantir que a página web seja indexada corretamente. Ao mesmo tempo, este fato é explorado para que o indexador do *search engine* "veja" um conteúdo diferente do visitante da página web.

4.2.6 Indexação de Meta Tags

Alguns tipos específicos de documentos incluem informações meta, tais como o autor, palavras-chave, descrição, e idioma. Para páginas HTML, a meta tag contém palavras-chave que são incluídas no índice. Durante o período inicial de crescimento da Internet e da tecnologia dos *search engines* (em específico, do hardware no qual eles trabalham) era indexada as palavras-chave contidas na meta tags para o índice avançado (e aplicando técnicas de origem e stop words). O documento completo não era analisado gramaticalmente. Naquele tempo, a indexação do texto completo não estava muito bem estabelecida, assim como o hardware para suportar tal tecnologia. O projeto da linguagem de marcação HTML inicialmente incluía suporte para meta tags para este mesmo propósito de ser correta e facilmente indexado, sem necessitar da tokenização.

Com o crescimento da Internet (o número de usuários capaz de visitar a web e o número de web sites aumentaram e a tecnologia de criar web sites e hospedar web sites melhorou) muitas corporações arrojadas tornaram-se “online” no meio da década de 90 e estabeleceram os web sites corporativos. As palavras-chave para descrever as páginas web (muitas das quais foram páginas web corporativas semelhantes a panfletos de produtos) mudaram de palavras-chave descritivas para palavras-chave orientadas ao marketing com o objetivo de aumentar as vendas pelo melhor posicionamento da página web nos resultados de busca para termos específicos de busca. O fato era que estas palavras-chave que eram especificadas subjetivamente estavam causando uma manipulação do índice (spamdexing), o que conduziu muitos *search engines* a adotar tecnologias de indexação do texto completo nos anos 90. Projetistas de *search engines* e companhias podiam colocar várias palavras-chave de marketing antes de disponibilizar a informação realmente interessante e útil. Devido a este conflito de interesses entre a meta de negócios de projetar web sites orientados a usuários, a equação de valor do tempo de vida de um cliente foi mudada para incorporar conteúdo mais útil no web site com a esperança de reter o visitante. Neste pensamento, a indexação de texto completo era mais objetiva e aumentou a qualidade dos resultados dos *search engines*, pois era um passo a mais fora do controle subjetivo dos resultados do *search engine*, resultando em avanços na tecnologia de indexação de texto completo.

4.3 A indexação do Sphider

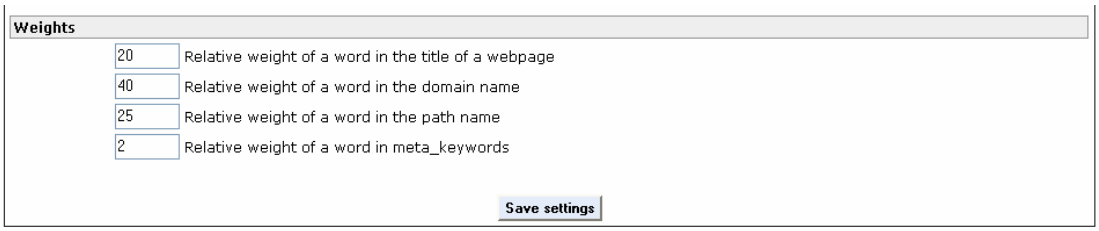
Após a obtenção do documento, explanado na seção 3.6, o sistema deve analisar o conteúdo do documento a fim de classificar o documento, suas palavras e seus *links*.

O Sphider possui a possibilidade de integração com dois analisadores (além de analisar arquivos HTML), um de arquivos DOC e outro de arquivos PDF, conseguindo assim, indexar este tipo de documentos.

Depois da detecção de qual o tipo de arquivo, o analisador recebe todo o conteúdo do documento, e realiza a tokenização, separando as palavras e contando-as. Assim pode-se iniciar a identificação do assunto da página e a inserção das palavras e suas relações no banco de dados.

O *search engine* é organizado de modo que a classificação das páginas é feita com base no título da página, nome do domínio, nome do arquivo e meta *keywords*. Cada item recebe um peso, que pode ser editado a qualquer momento na interface de administração através da aba “Settings”.

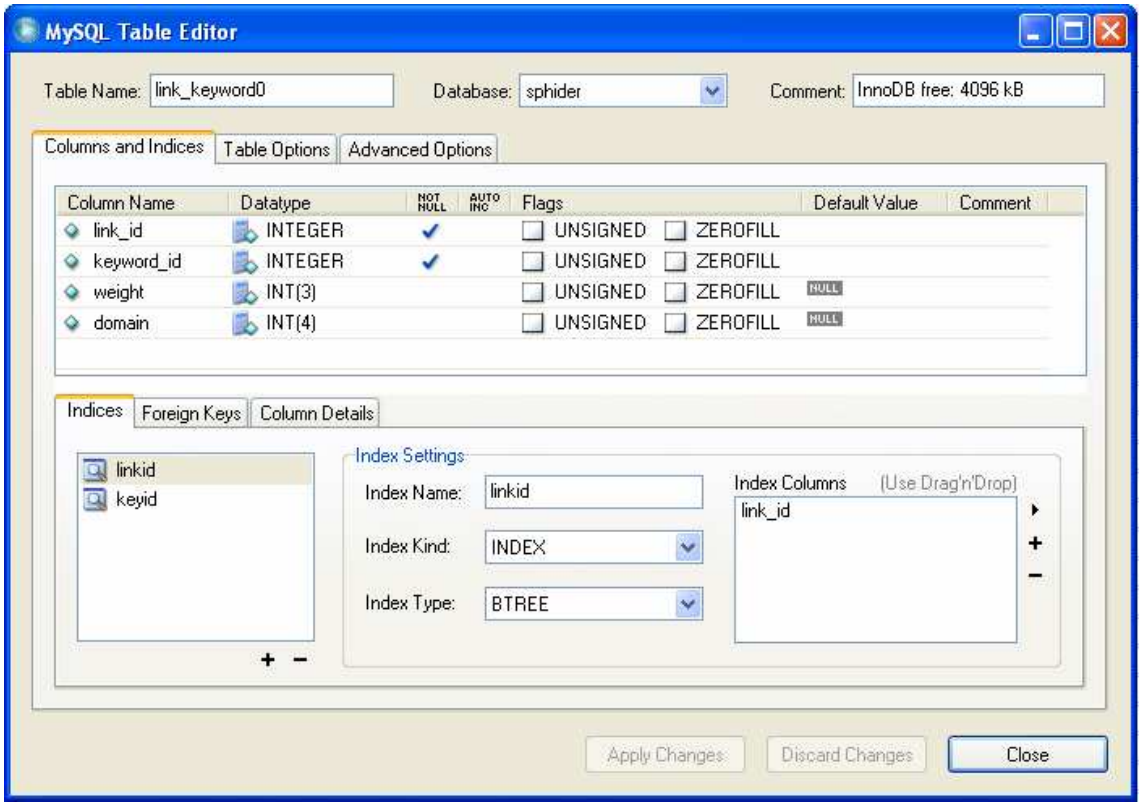
Ilustração 10 - Sphider: Controle dos pesos para rankeamento



Na imagem acima, a página receberia 20 pontos no título, 40 pontos no nome de domínio, 25 no nome de arquivo e apenas 2 pontos na meta *keywords*, para cada palavra-chave que o documento contiver. Assim, caso um usuário realize uma busca pela palavra “fabio” e este documento contiver a palavra “fabio”, ela possuiria pelo menos 87 pontos para esta palavra-chave.

A organização das tabelas de palavras-chave e *links* é algo similar ao índice avançado, onde temos uma relação de quais palavras-chave um determinado link contém. Nesta tabela também é relacionado o peso que a palavra-chave possui.

Ilustração 11 - Sphider: Organização do índice avançado



Após a inserção de todas as palavras-chave no banco de dados e a criação de seu peso, obtemos uma base de dados pronta para receber consultas.

O Sphider possui uma incompatibilidade com caracteres especiais como, por exemplo, “ç”, “ã”, “ó” e outros similares. Este problema surge quando o sistema requisita documentos que estão em idiomas que possuem estes caracteres. Em testes, o ideal é que o Sphider indexe páginas codificadas em UTF-8, assim poucos problemas de acentuação ocorrem. Uma solução para isto é a mudança no sistema, para trabalhar dependendo do tipo de codificação da página, ou ainda, uma análise para reparos em caso de caracteres considerados “estranhos”.

Outro ponto fundamental, que poderia ser melhorado no Sphider, é o processo de classificação. Atualmente ele baseia-se em fatores que podem ser manipulados com facilidade. Uma análise na densidade de palavras no texto, ou seja, contar quantas vezes a palavra-chave aparece no texto, dividido pela quantidade total de palavras, seria um fator muito bom. Acrescentar pesos em palavras-chave que aparecem em negrito, itálico ou em títulos (headings), agregaria uma qualidade ainda maior para o rankeamento.

Uma sugestão para aprimorar mais ainda o *search engine*, seria a implantação de um algoritmo baseado na idéia do algoritmo do Pagerank, que consiste na análise da linkagem entre páginas, onde cada página, ao linkar para outra, receberia um tipo de “voto”, afirmando a qualidade daquela página. Assim, as páginas com mais votos, possuiriam um ranking maior do que as outras.

5 BUSCA WEB

Quando um usuário realiza uma busca web, tipicamente digitando palavras-chave, o sistema procura no índice e provê uma lista das páginas que melhor combinam ao critério, normalmente com um breve resumo contendo o título do documento e, às vezes, partes do seu texto. A questão de busca é distinta, normalmente não estruturada e ambígua, variando em um padrão de termos governados por estritas regras de sintaxe.

5.1 Tipos

Existem três grandes categorias que cobrem a maioria das buscas realizadas nos *search engines* (Manning e Schutze, 2007):

- Questões de informação – Questões que cobrem grandes tópicos (ex. caminhões ou Brasil) para os quais existem milhares de resultados relevantes;
- Questões de navegação – Questões que procuram por um único web site ou página web ou ainda por uma entidade (ex. youtube ou Mercado Livre)
- Questões de transação – Questões que refletem a intenção do usuário em executar uma ação em particular, como por exemplo, comprar um carro ou adquirir um protetor de tela.

O *search engines* freqüentemente suportam um quarto tipo de questão que é pouco utilizado:

- Questões de conectividade – Questões que reportam a conectividade do grafo web indexado (ex. Quais *links* apontam para esta URL?, e Quantas páginas deste domínio estão indexadas?)

5.2 Características

A maioria dos *search engines* comerciais não liberam informações de seus logs de busca. Então, informações sobre o que os usuários estão procurando na Web são difíceis de estipular. Contudo, um estudo (Kawamoto e Mills, 2006) analisou

dados de 2001 do search engine Excite, verificando as questões de busca do *search engine*, mostrando algumas interessantes características da busca na Web:

- A média de tamanho de uma questão de busca é de 2.4 termos;
- Cerca de metade dos usuários realizou apenas uma busca, enquanto menos do que um terço dos usuários realizou 3 ou mais buscas únicas;
- Quase metade dos usuários analisou apenas uma ou duas páginas dos resultados (contendo 10 resultados por página);
- Menos de 5% dos usuários utilizaram a busca avançada (ex. Operadores Booleanos como AND, OR, e NOT)
- Os três termos mais usados freqüentemente foram “and”, “of” e “sex”.

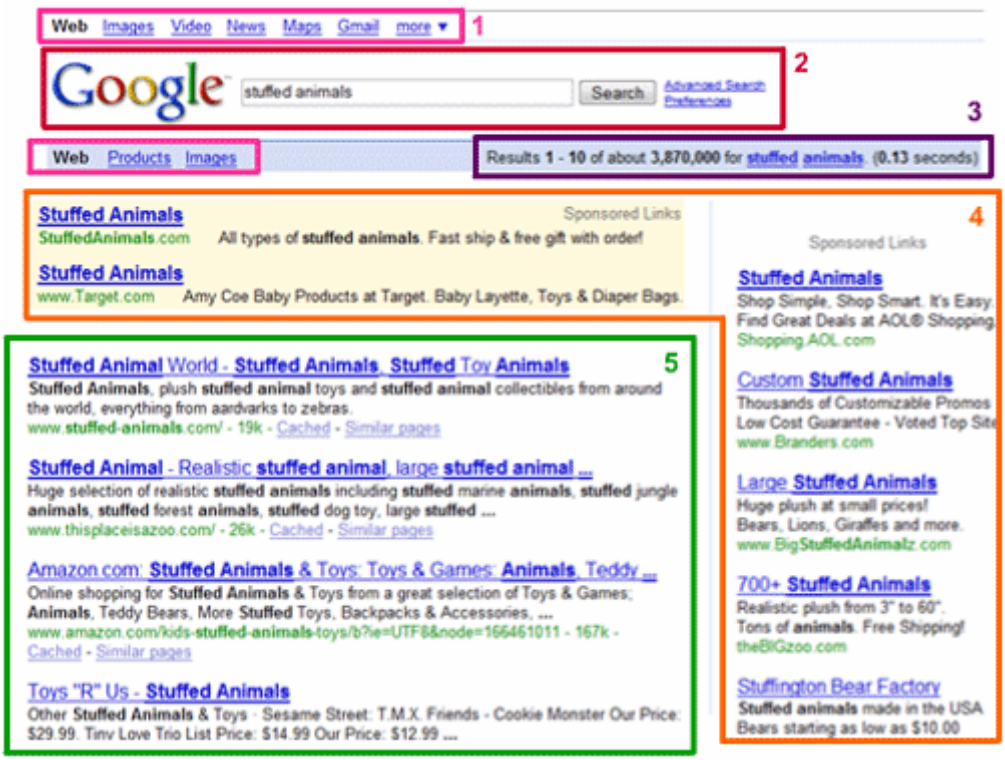
Outro estudo, realizado em 2005, nos logs de busca do Yahoo revelaram que 33% das buscas de um mesmo usuário eram repetidas e que 87% das vezes o usuário clicaria no mesmo resultado. Este resultado sugere que muitos usuários repetem questões para revisitar ou reencontrar informações.

5.3 Interação com o *Search engine*

De modo a adquirir um completo entendimento em como os *search engines* e os usuários interagem, é fundamental observar como é formada a página que retorna os resultados relevantes. Cada *search engine* retorna os resultados em um formato ligeiramente diferente e irá incluir resultados verticais (conteúdo específico focado na uma questão buscada, baseado em certos gatilhos, onde iremos ilustrar abaixo).

Google (Ilustração 12) – atualmente o *search engine* mais popular do mundo, não teve a sua interface simples mudada durante os anos de atuação.

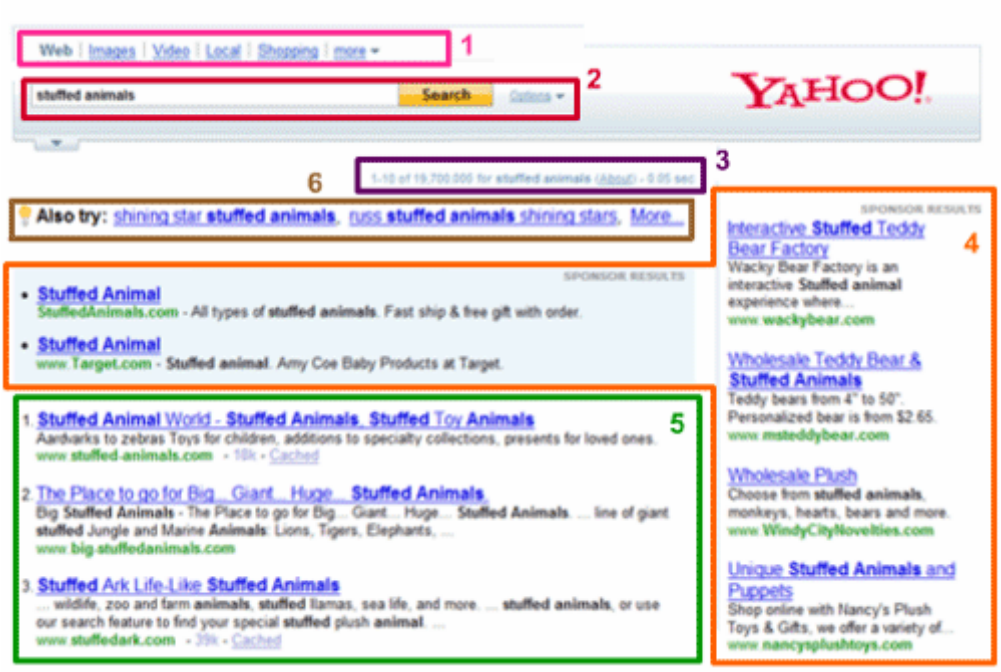
Ilustração 12 – Página de Resultados do Google



Fonte: SEOmoz - http://www.seomoz.org/blog/rewriting-the-beginners-guide-understanding-the-visuals-of-the-serps - Visitado em 2/11/2007

O Yahoo! (Ilustração 13) possui um layout similar, mas organiza as informações de um modo um pouco diferente e inclui uma seção adicional:

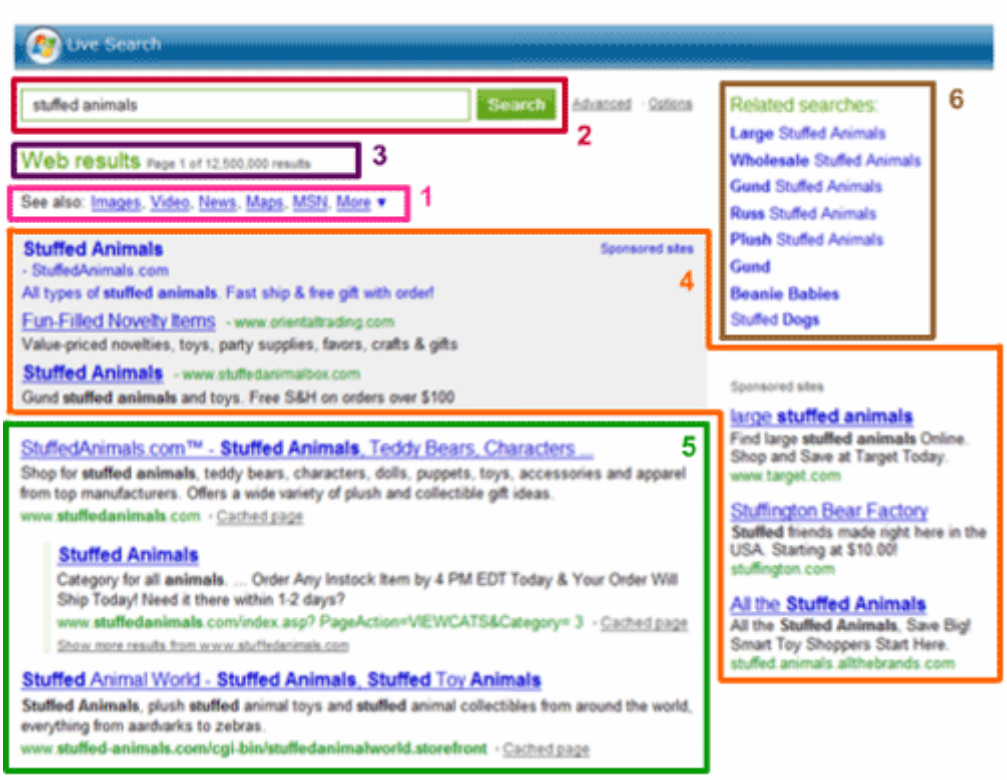
Ilustração 13 – Página de Resultados do Yahoo!



Fonte: SEOmoz - http://www.seomoz.org/blog/rewriting-the-beginners-guide-understanding-the-visuals-of-the-serps - Visitado em 2/11/2007

O MSN/Live (Ilustração 14, o *search engine* da Microsoft) é muito similar ao Yahoo!:

Ilustração 14 – Página de Resultados do MSN/Live



Fonte: SEOmoz - http://www.seomoz.org/blog/rewriting-the-beginners-guide-understanding-the-visuals-of-the-serps - Visitado em 2/11/2007

Cada uma das seções representa uma porção de informação provida pelos *search engines*. Abaixo, a lista de definições de o que cada uma das porções de informação deve fornecer:

1. Navegação Vertical: cada *search engine* oferece a opção de procurar em diferentes verticais tais como imagens, notícias, vídeos ou mapas. Ao clicar nestes *links* a sua busca será executada no dado índice limitado – no nosso exemplo acima, nós poderíamos procurar por notícias sobre “stuffed animals” ou vídeos sobre “stuffed animals”.

2. Caixa de Busca: todos os *search engines* mostram qual a questão que foi buscada e permitem a edição ou a criação de uma nova consulta a partir da página de busca. Eles oferecem também, *links* para a página de busca avançada.

3. Resultados da Informação: esta seção provê uma quantidade pequena de informação sobre os resultados que são mostrados, incluindo uma

estimativa do número de páginas relevantes para aquela consulta particular.

4. Propaganda Patrocinada: os “Resutados Patrocinados” utilizam os termos da busca para relacionar anúncios textuais adquiridos por empresas em diversas plataformas de propaganda – Google Adwords, Yahoo! Search Marketing e MSN AdCenter. Estes resultados patrocinados são ordenados por uma variedade de fatores, incluindo relevância e quantia de oferta.

5. Resultados Orgânicos: estes resultados são adquiridos dos índices primários do *search engine* e rankeados em ordem de relevância e popularidade de acordo com a complexidade dos algoritmos.

6. Sugestões de Refinamento da Busca: é uma característica relativamente recente oferecida pelo Yahoo! e pela Microsoft (e ocasionalmente pelo Google, dependendo da busca). O objetivo destes *links* é possibilitar aos usuários uma busca mais específica e mais relevante para atender as suas necessidades.

É importante observar que os *search engines* estão constantemente inovando e melhorando as páginas de resultado para que os usuários sejam mais bem atendidos e possam utilizar, com mais eficiência, todas as ferramentas disponibilizadas.

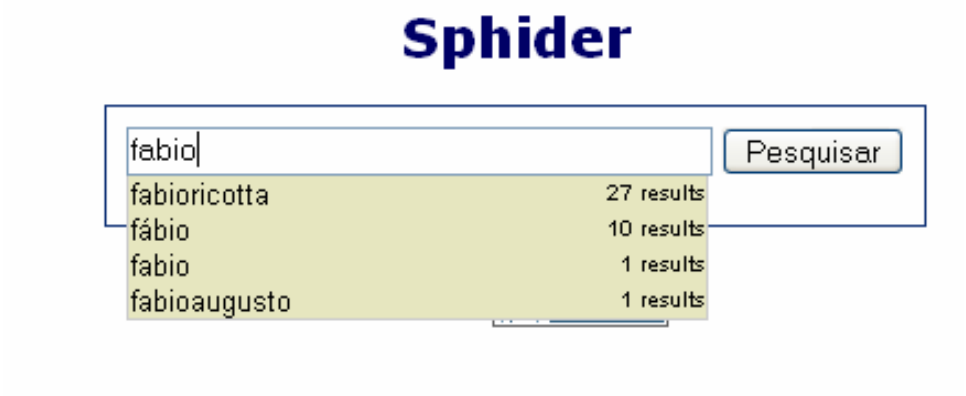
5.4 A interface do Sphider

O Sphider possui uma interface de busca muito simples e agradável, lembrando muito a interface do Google, exibida na seção 5.3.

Ilustração 15 - Sphider: Formulário de Busca

Uma funcionalidade interessante é que ao digitar uma palavra na caixa de busca, o sistema fornece sugestões de palavras similares. Este recurso faz uso da lista de palavras-chaves cadastradas no banco.

Ilustração 16 - Sphider: Formulário de Busca exibindo sugestões



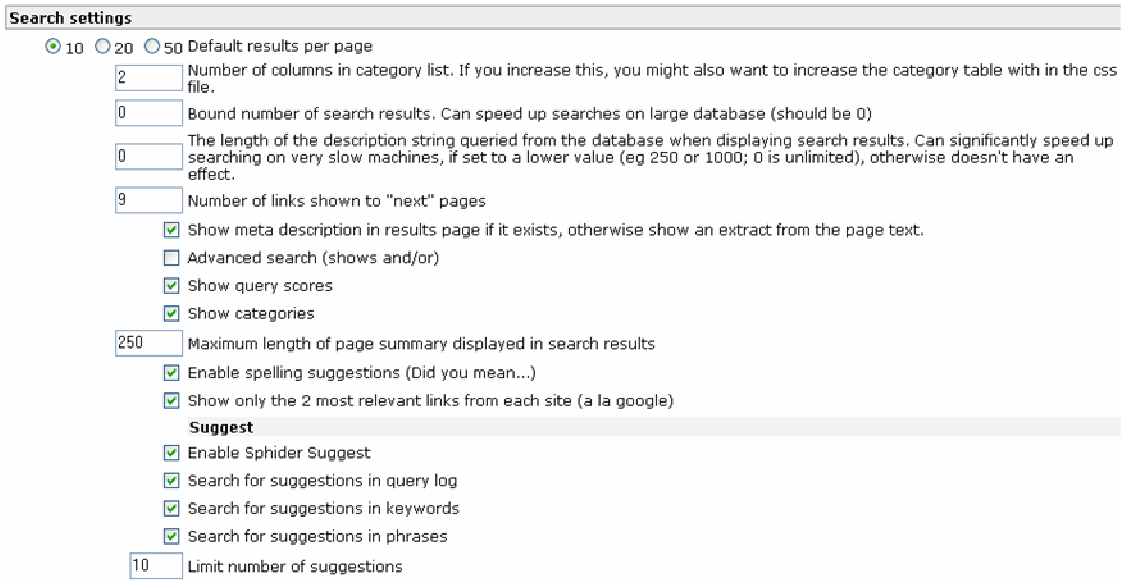
Assim como os maiores *search engines*, a página de resultados do Sphider é organizada com *links* à direita, apontando para os documentos relevantes.

Ilustração 17 - Sphider: Página de Resultados

Uma propriedade interessante que o Sphider possui é a amostragem do percentual de relevância de um documento com relação ao termo buscado. É possível, configurar o Sphider para não mostrar este resultado.

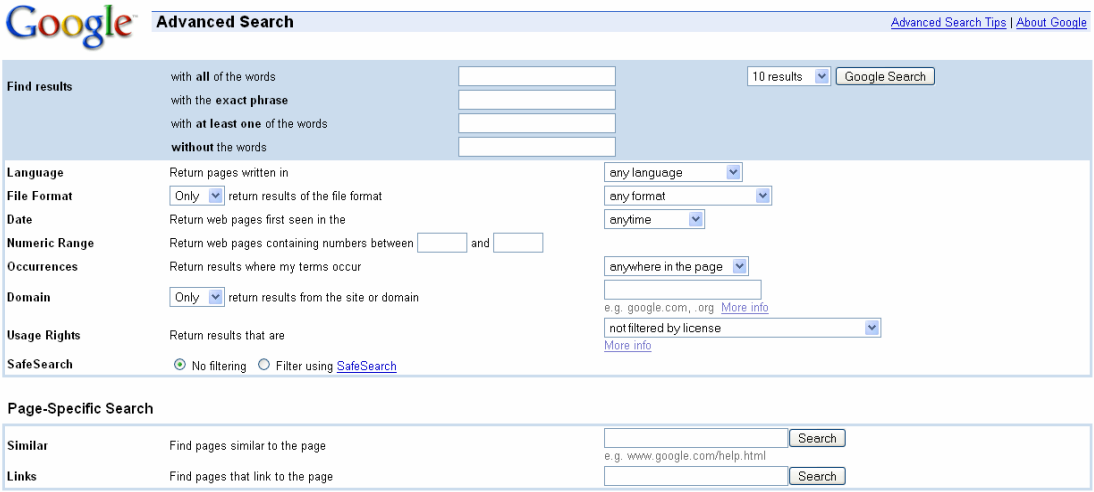
Outras configurações podem ser realizadas no sistema para que o usuário possua mais funcionalidades em sua busca. Estas configurações incluem: maior número de resultados por página, controle de tamanho da descrição do link, limitação de *links* relevantes de um mesmo domínio, controle de navegação inferior, etc.

Ilustração 18 - Sphider: Opções para a Busca



A interação com o usuário é um dos melhores atributos que o Sphider possui, mas algumas alterações podem ser realizadas para a melhoria desta interação. A maioria das configurações exibidas na imagem anterior poderiam ser alocadas em uma área de busca avançada do sistema, assim como o Google, e outros *search engines*, realizam:

Ilustração 19 – Busca Avançada do Google



Fonte: Google - http://www.google.com/advanced_search?hl=en - Visitado em 2/11/2007

6 CONCLUSÃO

A *World Wide Web* é um sistema de documentos em hipermídia que são interligados e executados na Internet. Para visualizar a informação, pode-se usar um programa de computador chamado navegador web para obter informações de servidores web e mostrá-los na tela do usuário. O usuário pode então seguir as hiperligações na página para outros documentos ou mesmo enviar informações de volta para o servidor para interagir com ele. O ato de seguir hiperligações é comumente chamado de "navegar" ou "surfar" na Web.

A quantidade de informações na Internet é tão grande e diversificada que é praticamente impossível encontrar tudo o que se precisa sem o uso de um mecanismo de busca de qualidade e que devolva uma resposta rapidamente.

O web crawler é responsável por percorrer a *World Wide Web* de uma forma metódica e automática para obter novas informações e atualizar documentos antigos. O grande volume de informações implica que o crawler só pode captura uma fração de páginas da web em um determinado tempo, então faz-se necessário priorizar esta captura. O grande volume de mudanças implica que o crawler necessite de uma política clara de re-visitação dos documentos para que possua um conjunto de informações grande a atualizada ao mesmo tempo.

A indexação é definida por como a informação é coletada, analisada, e armazenada para facilitar a recuperação rápida e precisa da informação. O objetivo de armazenar um índice é de otimizar a velocidade e o desempenho na busca por um documento relevante devido a um termo buscado. O uso de índices avançado e invertido são as mais comuns formas de implementação do índice. Caso o search engine não possua um índice bem modelado e otimizado, problemas podem surgir a partir do momento onde a quantidade de informações aumente, tornando o tempo de resposta à busca muito alto.

Quando um usuário realiza uma busca web, o sistema procura no índice e provê uma lista das páginas que melhor combinam ao critério, obedecendo um algoritmo de classificação, normalmente com um breve resumo contendo o título do documento e, às vezes, partes do seu texto. Através de observações nos principais

search engines, é notável a diferença nos algoritmos de classificação para as mais variadas buscas.

A área de search engines representa um grande desafio para os profissionais de computação, pois envolvem conceitos de Banco de Dados, Engenharia de Software, Computação Paralela, Análise de Algoritmos, Matemática Discreta, Redes Neurais e outras. Logo, o estudo nesta área é sempre renovado, com novas idéias e técnicas para oferecer à comunidade uma ferramenta de qualidade e agilidade.

Este trabalho abordou ainda, o Sphider, um *search engine open-source*, que ilustrou cada uma das áreas da arquitetura de um *search engine* e constatou-se que o sistema possui muitos pontos falhos, mas com várias possibilidades de melhorias.

Como sugestão para trabalhos futuros, segue abaixo uma lista de alguns assuntos que podem ser explorados:

- Pagerank, o algoritmo de rankeamento do Google;
- Implementação de melhorias no *search engine* Sphider;
- Compressão de dados para *search engines*;
- Funcionamento de indexadores de arquivos DOC ou PDF;
- Algoritmos de classificação de páginas web.

BIBLIOGRAFIA

Abiteboul, S., Preda, M., and Cobena, G. (2003). "Adaptive on-line page importance computation". In Proceedings of the twelfth international conference on World Wide Web: 280-290.

Baeza-Yates, R. and Castillo, C. (2002). Balancing volume, quality and freshness in web crawling. In **Soft Computing Systems – Design, Management and Applications**, pages 565–572, Santiago, Chile. IOS Press Amsterdam.

Baeza-Yates, R., Castillo, C., Marin, M. and Rodriguez, A. (2005). Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering. In Proceedings of the Industrial and Practical Experience track of the 14th conference on World Wide Web, pages 864–872, Chiba, Japan. ACM Press.

Boldi, P., Codenotti, B., Santini, M., and Vigna, S. (2004a). **UbiCrawler: a scalable fully distributed Web crawler**. Software, Practice and Experience, 34(8):711–726.

Boldi, P., Santini, M., and Vigna, S. (2004b). Do your worst to make the best: Paradoxical effects in pagerank incremental computations. In Proceedings of the third Workshop on Web Graphs (WAW), volume 3243 of Lecture Notes in Computer Science, pages 168-180, Rome, Italy. Springer.

Brin, S. and Page, L. (1998). **The anatomy of a large-scale hypertextual Web search engine**. Computer Networks and ISDN Systems, 30(1-7):107–117.

Burner, M. (1997). **Crawling towards eternity – building an archive of the World Wide Web**. Web Techniques, 2(5).

Castillo, C. (2004). **Effective Web Crawling**. PhD thesis, University of Chile.

Chakrabarti, S. (2003). **Mining the Web**. Morgan Kaufmann Publishers. ISBN 1-55860-754-4

Chakrabarti, S., van den Berg, M., and Dom, B. (1999). **Focused crawling: a new approach to topic-specific web resource discovery**[dead link – history]. Computer Networks, 31(11–16):1623–1640.

Cho, J., Garcia-Molina, H., and Page, L. (1998). "Efficient crawling through URL ordering". In Proceedings of the seventh conference on World Wide Web.

Cho, J. and Garcia-Molina, H. (2000). Synchronizing a database to improve freshness. In Proceedings of ACM International Conference on Management of Data (SIGMOD), pages 117-128, Dallas, Texas, USA.

Cho, J. and Garcia-Molina, H. (2002). Parallel crawlers. In Proceedings of the eleventh international conference on World Wide Web, pages 124–135, Honolulu, Hawaii, USA. ACM Press.

Cho, J. and Garcia-Molina, H. (2003). **Effective page refresh policies for web crawlers**. ACM Transactions on Database Systems, 28(4).

Cho, J. and Garcia-Molina, H. (2003). **Estimating frequency of change**. ACM Transactions on Internet Technology, 3(3).

Cothey, V. (2004). "**Web-crawling reliability**". Journal of the American Society for Information Science and Technology 55 (14).

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. (2000). Focused crawling using context graphs. In Proceedings of 26th International Conference on Very Large Databases (VLDB), pages 527-534, Cairo, Egypt.

Dill, S., Kumar, R., Mccurley, K. S., Rajagopalan, S., Sivakumar, D., and Tomkins, A. (2002). Self-similarity in the web[dead link]. ACM Trans. Inter. Tech., 2(3):205–223.

Eichmann, D. (1994). The RBSE spider: balancing effective search against Web load. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

Edward G. Coffman, Z. Liu, R. W. (1998). **Optimal robot scheduling for Web search engines**. Journal of Scheduling, 1(1):15–29.

Edwards, J., McCurley, K. S., and Tomlin, J. A. (2001). "An adaptive model for optimizing performance of an incremental web crawler". In Proceedings of the Tenth Conference on World Wide Web: 106-113.

Heydon, A. and Najork, M. (1999). **Mercator: A scalable, extensible Web crawler**. World Wide Web, 2(4):219–229.

Ipeirotis, P., Ntoulas, A., Cho, J., Gravano, L. (2005) Modeling and managing content changes in text databases. In Proceedings of the 21st IEEE International Conference on Data Engineering, pages 606-617, April 2005, Tokyo.

Kobayashi, M. and Takeda, K. (2000). **"Information retrieval on the web"**. ACM Computing Surveys 32 (2): 144-173.

Koster, M. (1993). **Guidelines for robots writers**.

Koster, M. (1995). **Robots in the web: threat or treat ?** ConneXions, 9(4).

Koster, M. (1996). **A standard for robot exclusion**.

Lawrence, S. and Giles, C. L. (2000). **Accessibility of information on the web**. Intelligence, 11(1), 32–39.

McBryan, O. A. (1994). GENVL and WWWW: Tools for taming the web. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

Menczer, F. (1997). ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery. In D. Fisher, ed., Machine Learning: Proceedings of the 14th International Conference (ICML97). Morgan Kaufmann

Menczer, F. and Belew, R.K. (1998). Adaptive Information Agents in Distributed Textual Environments. In K. Sycara and M. Wooldridge (eds.) Proc. 2nd Intl. Conf. on Autonomous Agents (Agents '98). ACM Press

Miller, R. and Bharat, K. (1998). Sphinx: A framework for creating personal, site-specific web crawlers. In Proceedings of the seventh conference on World Wide Web, Brisbane, Australia. Elsevier Science.

Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages[dead link – history]. In Proceedings of the Tenth Conference on World Wide Web, pages 114–118, Hong Kong, May 2001. Elsevier Science.

Nelson, M. L. , Van de Sompel, H. , Liu, X., Harrison, T. L. and McFarland, N. (2005). "mod_oai: An Apache module for metadata harvesting". In Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2005): 509.

Pant, G., Srinivasan, P., Menczer, F. (2004). **"Crawling the Web"**. Web Dynamics: Adapting to Change in Content, Size, Topology and Use, edited by M. Levene and A. Poulovassilis: 153-178.

Pinkerton, B. (1994). Finding what people want: Experiences with the WebCrawler[dead link – history]. In Proceedings of the First World Wide Web Conference, Geneva, Switzerland.

Risvik, K. M. and Michelsen, R. (2002). **Search Engines and Web Dynamics**. Computer Networks, vol. 39, pp. 289–302, June 2002.

Shkapenyuk, V. and Suel, T. (2002). Design and implementation of a high performance distributed web crawler. In Proceedings of the 18th International Conference on Data Engineering (ICDE), pages 357-368, San Jose, California. IEEE CS Press.

da Silva, A. S., Veloso, E. A., Golgher, P. B., Ribeiro-Neto, B. A., Laender, A. H. F., and Ziviani, N. (1999). Cobweb – a crawler for the Brazilian web. In Proceedings of String Processing and Information Retrieval (SPIRE), pages 184–191, Cancun, Mexico. IEEE CS Press.

Zeinalipour-Yazti, D. and Dikaiakos, M. D. (2002). Design and implementation of a distributed crawler and filtering processor. In Proceedings of the Fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, pages 58–74, Caesarea, Israel. Springer.

Clarke, C., Cormack, G.: **Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System**. TechRep MT-95-01, University of Waterloo, February 1995.

Stephen V. Rice, Stephen M. Bailey. **Searching for Sounds**. Comparisonics Corporation. May 2004. Verified Dec 2006

Charles E. Jacobs, Adam Finkelstein, David H. Salesin. **Fast Multiresolution Image Querying**. Department of Computer Science and Engineering, University of Washington. 1995. Verified Dec 2006

Lee, James. **Software Learns to Tag Photos**. MIT Technology Review. November 09, 2006. Pg 1-2. Verified Dec 2006. Commercial external link

Brown, E.W.: **Execution Performance Issues in Full-Text Information Retrieval**. Computer Science Department, University of Massachusetts at Amherst, Technical Report 95-81, October 1995.

Cutting, D., Pedersen, J.: Optimizations for dynamic inverted index maintenance. Proceedings of SIGIR, 405-411, 1990.

Linear Hash Partitioning. **MySQL 5.1 Reference Manual**. Verified Dec 2006

Gusfield, Dan [1997] (1999). **Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology**. USA: Cambridge University Press. ISBN 0-521-58519-8. .

C. C. Foster, Information retrieval: information storage and retrieval using AVL trees, Proceedings of the 1965 20th national conference, p.192-205, August 24-26, 1965, Cleveland, Ohio, United States

Landauer, W. I.: **The balanced tree and its utilization in information retrieval**. IEEE Trans. on Electronic Computers, Vol. EC-12, No. 6, December 1963.

Google Ngram Datasets for sale at LDC Catalog

Jeffrey Dean and Sanjay Ghemawat. **MapReduce: Simplified Data Processing on Large Clusters**. Google, Inc. OSDI. 2004.

Grossman, Frieder, Goharian. **IR Basics of Inverted Index**. 2002. Verified Dec 2006.

Tang, Hunqiang. Dwarkadas, Sandhya. **"Hybrid Global Local Indexing for Efficient Peer to Peer Information Retrieval"**. University of Rochester. Pg 1. http://www.cs.rochester.edu/u/sarrmor/publications/eSearch-NSDI.ps

Tomasic, A., et al: **Incremental Updates of Inverted Lists for Text Document Retrieval**. Short Version of Stanford University Computer Science Technical Note STAN-CS-TN-93-1, December, 1993.

Sergey Brin and Lawrence Page. **The Anatomy of a Large-Scale Hypertextual Web Search Engine**. Stanford University. 1998. Verified Dec 2006.

H.S. Heaps. **Storage analysis of a compression coding for a document database**. 1NFOR, I0(i):47-61, February 1972.

Murray, Brian H. **Sizing the Internet**. Cyveillance, Inc. Pg 2. July 2000. Verified Dec 2006.

Blair Bancroft. **Word Count:A Highly Personal-and Probably Controversial-Essay on Counting Words**. Personal Website.

Berners-Lee, T., **"Hypertext Markup Language - 2.0"**, RFC 1866, Network Working Group, November 1995.

Krishna Nareddy. **Indexing with Microsoft Index Server**. MSDN Library. Microsoft Corporation. January 30, 1998.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze (2007), **Introduction to Information Retrieval**, Ch. 19

Dawn Kawamoto and Elinor Mills (2006), **AOL apologizes for release of user search data**

Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, Tefko Saracevic (2001). **"Searching the web: The public and their queries"**. Journal of the American Society for Information Science and Technology 52 (3): 226-234.

Jaime Teevan, Eytan Adar, Rosie Jones, Michael Potts (2005). "History repeats itself: Repeat Queries in Yahoo's query logs". Proceedings of the 29th Annual ACM Conference on Research and Development in Information Retrieval (SIGIR '06): 703-704. DOI:10.1145/1148170.1148326.

Fernando Renier Gibotti da Silva. **GEODISCOVER - MECANISMO DE BUSCA ESPECIALIZADO EM DADOS GEOGRÁFICOS**. Instituto Nacional de Pesquisas Espaciais. 1996.

Yousef, Mohammed 2007. Class: PHP Thread. Disponível em: <http://www.phpclasses.org/browse/package/4082.html> Acesso em: 02/11/2007.

A Companhia, 2007. “Uma visão no futuro dos motores de busca (ou search engines se você não fala português)”. Disponível em: < http://acompanhia.blogarium.net/2006/08/11/uma-visao-no-futuro-dos-motores-de-busca-ou-search-engines-se-voce-nao-fala-portugues/> Acesso em: 02/11/2007.

Fish, Hand 2007. SEOmoz. “Rewriting the Beginner's Guide - Understanding the Visuals of the SERPs”. Disponível em: < http://www.seomoz.org/blog/rewriting-the-beginners-guide-understanding-the-visuals-of-the-serps> Acesso em: 02/11/2007.

James Gillies and Robert Cailliau. **How the Web Was Born: The Story of the Word Wide Web**. Oxford University Press, 2000.

Vannevar Bush. **As we may think**. Atlantic Monthly, 176(1):101-8, 1945.

Wikipedia. “Breadth-first search”. Disponível em: <http://en.wikipedia.org/wiki/Breadth-first_search> Acesso em: 02/11/2007.

Wikipedia. “Listo of Search Engines”. Disponível em: < http://en.wikipedia.org/wiki/List_of_search_engines> Acesso em: 02/11/2007.

Wikipedia. “Pagerank”. Disponível em: < http://en.wikipedia.org/wiki/Pagerank> Acesso em: 02/11/2007.

Wikipedia. “Search Engine”. Disponível em: < http://pt.wikipedia.org/wiki/Search_Engine> Acesso em: 02/11/2007.

Wikipedia. “Motores de Busca”. Disponível em: < http://pt.wikipedia.org/wiki/Motores_de_busca> Acesso em: 02/11/2007.

Wikipedia. “Word Wide Web”. Disponível em: < http://pt.wikipedia.org/wiki/World_Wide_Web> Acesso em: 02/11/2007.

Wikipedia. “Hiperligação”. Disponível em: < http://pt.wikipedia.org/wiki/Hiperligação> Acesso em: 02/11/2007.

ANEXOS

7 ANEXO A - SPHIDER – DOCUMENTAÇÃO

7.1 Documentação do Sistema

7.1.1 Instalação

1. Descompacte os arquivos, e copie-os para o servidor, por exemplo, para */home/youruser/public_html/sphider*
2. No servidor, crie um banco de dados MySQL para armazenar os dados do Sphider.
    a. Na linha de comando digite (para logar no MySQL):

    mysql -u <usuário> -p

    Entre com a senha quando for requisitado.

    b. No MySQL digite:

    CREATE DATABASE sphider;

    Você pode usar outro nome para o banco de dados.

    c. Digite *exit* para sair do MySQL
3. No diretório *settings*, edite o arquivo *database.php* e mude as variáveis *$database*, *$mysql_user*, *$mysql_password* e *$mysql_host* para os valores corretos.
4. Abra o arquivo *install.php* (no diretório *admin*) no seu navegador, o qual irá criar as tabelas necessárias para o Sphider operar.

    Outra maneira de criar as tabelas é utilizando o script *tables.sql* que está no diretório *sql* da distribuição do Sphider. Na linha de comando digite:

    mysql -u <usuário> -p sphider_db < [caminho_do_sphider]/sql/tables.sql
5. No diretório *admin*, edite o arquivo *auth.php*, para mudar o usuário e senha do administrador (os valores padrão são “admin” e “admin”)
6. Abra o arquivo *admin/admin.php* no seu navegador e comece a realizar o crawling.

7. A página de busca é *search .php*.

7.1.2 Opções de Indexação

Full: o processo de indexação continua até que não haja mais *links* ou não seja permido seguir mais *links*.

To deph: indexa até certa profundidade, onde profundidade significa quantos “clicks” uma dada página está da página inicial. Profundidade 0 significa que somente a página inicial é indexada, profundidade 1 indexa a página inicial e todas as páginas que ela linka, etc.

Reindex: marcando esta opção, o processo de indexação é forçado a indexar uma página mesmo que ela já tenha sido indexada.

Spider can leave domain: Por padrão, o Sphider nunca sai de um domínio fornecido, então os *links* do *dominio.com* que apontam para o *domínio2.com* não são seguidos. Marcando esta opção o Sphider poderá deixar o domínio, contudo é aconselhável definir uma lista de deve incluir / não deve incluir para prevenir o crawler de seguir muito longe.

Must include / must not include: veja a seção de “Evitando páginas de serem indexadas” para maiores explicações.

7.1.3 Customizando

Se você deseja mudar o comportamento padrão do Sphider, você pode fazer isto através de interface de administração, ou editando diretamente o arquivo *conf.php* no diretório *settings*.

Para mudar o visual da página de busca para adequar ao visual do seu web site, modifique ou adicione um template no diretório *templates*. Você pode modificar o arquivo *search.css* e os templates de topo e rodapé (*header.html* e *footer.html*) para obter um layout modificado. Modificações maiores podem ser realizadas editando o resto dos arquivos de template.

A lista de arquivos que não são verificados para indexação está disposta no arquivo *ext.txt* no diretório *admin*. A lista de palavras comuns que não são indexadas é fornecida no arquivo *common.txt* no diretório *include*.

7.1.4 Utilizando o indexador através da linha de comando

É possível utilizar o crawling através da linha de comando, usando a sintaxe:

php spider.php <opções>

onde <opções> são:

Tabela 5 – Listagem de opções do indexador do Sphider

-all	Reindexa todos os sites do banco de dados
-u <url>	Seta uma URL a ser indexada
-f	Seta a profundidade da indexação para completa (profundidade ilimitada)
-d <num>	Seta a profundidade da indexação para <num>
-l	Permite o crawler a deixar o domínio original
-r	Requisita que o crawler reindexe um determinado web site
-m <string>	Seta a(s) string(s) que uma URL deve incluir (use \n como delimitador entre múltiplas strings)
-n <string>	Seta a(s) string(s) que uma URL não deve incluir (use \n como delimitador entre múltiplas strings)

Por exemplo, para utilizar o crawling e indexação em http://www.dominio.com/teste.html com profundidade 2, utilize:

php spider.php -u http://www.dominio.com/teste.html -d 2

Se você deseja reindexar a mesma URL utilize:

php spider.php -u http://www.dominio.com/teste.html -r

7.1.5 Indexando arquivos PDF

Arquivos PDF e DOC podem ser indexados via binários externos. Baixe e instale o pdftotext e o catdoc e insira a localização (path) no arquivo *conf.php*

(note que no Windows, você não deve utilizar espaços no endereço do executável). Adicionalmente, na seção de administração, marque as caixas de Indexar PDF e Indexar DOC (alternativamente, mude os parâmetros *$index_pdf* e *$index_doc* para 1 no arquivo *conf.php*).

7.1.6 Evitando páginas de serem indexadas

7.1.6.1 Robots.txt

A maneira mais comum de evitar páginas de serem indexadas é utilizando o padrão robots.txt, através da criação de um arquivo robots.txt no diretório raiz do seu servidor, ou alterando a meta tag *ROBOTS* para *NOINDEX*.

7.1.6.2 Deve incluir / não deve incluir lista de strings

Uma opção muito interessante que o Sphider suporta é a inclusão ou não de uma lista de strings para um site (clique nas opções avançadas na área de índice para localizar esta opção). Qualquer URL contendo uma string da lista de “não deve incluir” será ignorada. Todas as strings da lista devem ser separadas pelo terminador de linha (enter). Por exemplo, para prevenir que um fórum de seu site seja indexado, você deve inserir www.seusite.com/forum na lista de “não deve incluir”. Isto significa que todas as URLs que contiverem esta string serão ignoradas e não serão indexadas. Toda string iniciada com um “*” na frente será considerada como uma expressão regular, então “*/[a]+/” significa uma string contendo um ou mais a’s .

7.1.6.3 Ignorando *Links*

O Sphider respeita o atributo rel=“nofollow” em tags <a href...>, então por exemplo, o link foo.html em <a href=“foo.html” rel=“nofollow”> será ignorado.

7.1.6.4 Ignorando partes de uma página

O Sphider inclui uma opção para excluir partes de uma página de serem indexadas. Isto pode ser usado, por exemplo, para prevenir o excesso de palavras-chave nos resultados de busca quando estas aparecem na maioria das páginas (como um cabeçalho, rodapé ou em um menu). Qualquer parte da página que estiver entre as tags

<!--sphider_noindex--> e <!--/sphider_noindex--> não é indexado, contudo, *links* contidos nesta área serão seguidos.