DCC923: Information Retrieval – Web Search Engines
Assignment 1: Crawling

Dan Valle[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

dan.valle@dcc.ufmg.br

1. Introdução

As *Máquinas de Busca* são usadas todos os dias por milhões de pessoas e são fundamentais para a busca eficiente e rápida por informação, consequentemente, levando à divulgação de conhecimento e avanço tecnológico, científico e cultural. É imediata a necessidade de acessar alguma dessas ferramentas assim que as pessoas se deparam com alguma questão que não se tenha uma resposta pronta, levando a acumular mais conhecimento a ponto de poder refletir para encontrar uma resposta apropriada. Com cada vez mais pessoas fazendo isso, barreiras são quebradas a todo momento e novas ideias e conhecimentos surgem. Os primeiros buscadores e mais difundidos globalmente surgiram na década de noventa e alguns se encontram presentes até hoje como gigantes do mercado, como no caso do *Google*[1], *Bing*[2] e *Yahoo*[3].

As máquinas criadas podem ter objetivos diferentes e atuam com estratégias aperfeiçoadas para completá-los com maior eficiência. Alguns exemplos mais conhecidos são os buscadores *globais* e *verticais*. O primeiro, representado pelos nomes citados, tem como objetivo pesquisar todos os documentos da rede para que possam apresentar uma variedade de contextos para qualquer tipo de busca enquanto o segundo faz buscas em algum domínio de objetivos que propõem.

A primeira etapa de um motor de busca é conhecida como *Web Crawler*. Nessa fase, um grande número de paginas são armazenadas percorrendo as ligações que uma faz à outra. Assim, o *Crawler* percorre uma página e, ao mesmo tempo, encontra informações que possam levar a outras em seguida. Algumas características devem ser levadas em conta para respeitar a comunidade, como só fazer aquilo que lhe for permitido pelos próprios desenvolvedores de cada página caso alguma regra seja imposta pelos mesmos. Este trabalho tem como objetivo desenvolver um *Crawler* de alta performance e que respeite todas as regras aplicáveis. Além disso, fazer uma análise do método escolhido e do seu comportamento à medida que sofre variações internas ou externas. O foco do *Crawler* é encontrar a maior quantidade de domínios diferentes e priorizar aqueles que são brasileiros.

2. Materiais e Métodos

Esta seção visa explicar a forma como o problema foi abordado, mostrando as principais decisões de projeto e o porquê dessas escolhas. Também apresenta as ferramentas que

[1]https://www.google.com.br
[2]https://www.bing.com/
[3]https://www.yahoo.com/

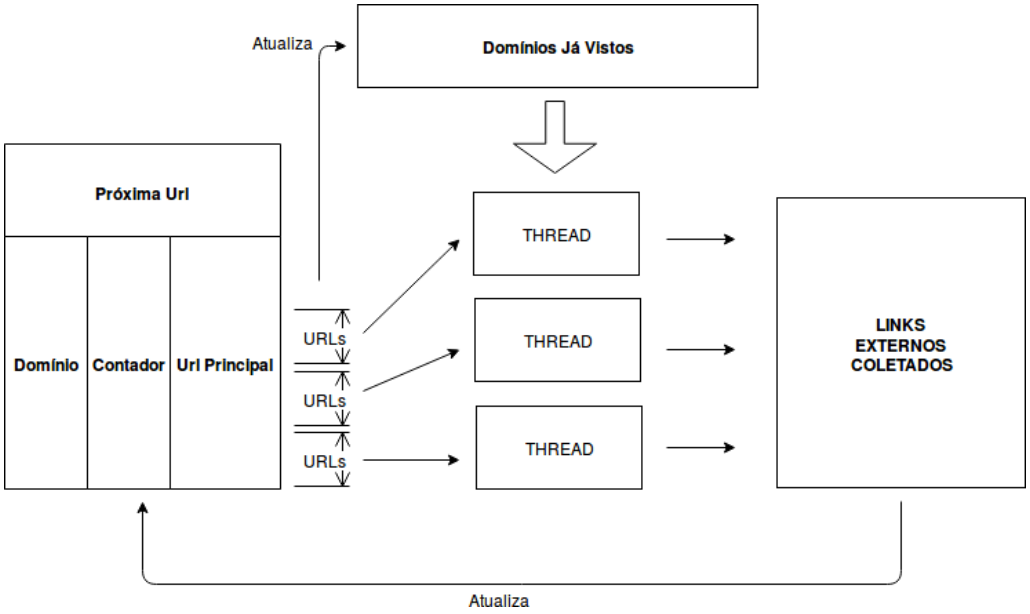

Figura 1. Modelo Escolhido.

auxiliaram tecnicamente na realização do que fora proposto e como elas foram utilizadas para facilitar o desenvolvimento deste projeto.

2.1. Ferramentas

A linguagem de programação usada foi *C++* pela sua eficiência e popularidade para projetos de busca semelhantes. Além das bibliotecas padrões, foram utilizadas duas ferramentas do *Chilkat Software*[4] que auxiliam com funções básicas para comunicação e lidar com páginas Web, chamadas *CkSpider* e *CkStringArray*. A primeira é responsável pela movimentação entre páginas e a segunda armazena de uma forma eficiente domínios escolhidos.

2.2. Arquitetura

Vários pontos devem ser observados para formular uma estratégia que seja eficiente e que não degrade rapidamente à medida que as páginas são coletadas, como o tempo de espera por páginas (extremamente superior ao de processamento interno) e por uma nova requisição a um mesmo domínio que respeite o mínimo de 30 segundos da anterior. Para superar esses problemas, um modelo foi criado para reduzir ao máximo o tempo de espera nestas ocasiões, podendo ser observado na Figura 1.

O primeiro elemento que pode ser observado é uma tabela que guarda as próximas URLs (*Uniform Resource Locator*) que o *Crawler* passará. Cada tupla dessa tabela contém um domínio base e dois valores ligados a ele, um contador e uma URL. Toda vez que ele vai ser atualizado, várias URLs chegam e o seguinte processo acontece para cada uma:

- Observa-se a existência do domínio base dessa URL na tabela.
- Caso não exista, adiciona-o com seu contador em um e a URL é a mesma observada.

[4]https://www.chilkatsoft.com/

- Se existir, soma um ao seu contador e observa se esta nova URL é melhor que a atual a partir de um *score* (2.2.2) para poder substituí-la.

Na próxima etapa, são escolhidos conjuntos de tamanhos determinados de URLs para serem enviados para as *Threads*. Tem-se um conjunto para cada *Thread* criada. A escolha é feita a partir do contador pois, dessa forma, o modelo prioriza páginas mencionadas mais vezes até o momento. Assim, páginas que são mais vistas e referenciadas por outras são consideras mais importante.

Assim que todas foram escolhidas, a lista de domínios já vistos é atualizado com os domínios base das URLs. Ele é fundamental para que não sejam coletados *links* de domínios já visitados pelo *Crawler*. Isso pode ser feito pois páginas desses domínios serão coletadas nas *Threads* e não haverá necessidade de voltar a visitar nenhuma página dos mesmo domínios no futuro. Manter somente domínios (não URLs) é fundamental para a eficiência do modelo, pois a quantidade de URLs visitadas é muito maior que a de domínios.

Dentro das *Threads*, as páginas dos domínios enviados são visitadas de uma forma eficiente e que respeite as regras da *Web*, como será explicado em 2.2.1. Ao mesmo tempo, toda URL de domínios ainda não visitados são coletados para que, ao final das *Threads*, eles sejam usados para atualizar a tabela das próximas URLs. Foi pensado dessa forma para criar o máximo de independência entre elas, envolvendo o mínimo de variáveis que se compartilhem, e criar um sistema cíclico que se auto alimenta e abre possibilidade para novas decisões e análises entre cada volta.

2.2.1. Threads

Está parte faz a coleta e passeia entre as páginas. Para manter sua independência e simplicidade, ela recebe um conjunto de URLs (uma por domínio) e controla seu próprio arquivo onde serão escritas as páginas em disco. Para respeitar os 30 segundos entre duas requisições a um mesmo domínio e não ter que simplesmente aguardar, as URLs são visitadas alternando-se até que todas são visitas. Ao retornar à primeira, observa se o tempo mínimo já foi respeitado e continua seguindo o mesmo processo.

Cada visita a uma página retorna *links* que apontam para o mesmo domínio e para outros. Como foi mencionado, os externos são utilizado para atualizar a tabela para o próximo ciclo. Os internos são ordenados a partir do mesmo *score* 2.2.2 visto na tabela e usados para continuar a navegar em um mesmo domínio. Isso aumenta o aproveitamento, elevando a quantidade de *links* externos coletados em menos páginas internas.

Como o objetivo deste projeto é encontrar o máximo de domínios possíveis, a movimentação interna deve ser limitada ao mesmo tempo que procura a maior quantidade de referências externas. Se uma página aponta para diversos domínios que não o mesmo, ele é considerado valioso e mais páginas do mesmo devem ser visitadas. Caso contrário, não há necessidade de aprofundar-se mais. No entanto, domínios com páginas importantes não devem se prolongar demais. Então, um critério de parada foi definido a partir desse objetivos. Com i sendo a quantidade de páginas visitadas em um mesmo domínio e N a quantidade de *links* externos encontrados na página i, prolonga-se o pas-

seio interno caso:

$$2^i \leq N$$

Como 2^i cresce rapidamente, páginas extremamente valiosas não se prolongam demais. Por outro lado, páginas não tão úteis têm a chance de serem vistas algumas vezes.

2.2.2. Score

O *score* é fundamental para controlar quais páginas são mais importantes para os objetivos propostos. Alguns elementos da URL são considerados para o cálculo desse valor:

- Tamanho
- Quantidade de blocos (observados por '/')
- Padrões que aparecem

Urls menores e com menos blocos são consideradas melhores e os padrões servem para penalizar ou bonificar. Para priorizar URLs brasileiras não seria ideal simplesmente usar aquelas que terminam com o termo *'.br'*, pois existem outras que não possuem esse padrão e são mais importantes. Uma solução usada é bonificar aquelas que possuem esse padrão e ainda assim competir com as outras observando os demais elementos. Padrões observados como prejudiciais pelos testes são usados para penalizar e receber scores piores para evitar seu uso.

2.3. Checkpoints e Armazenamento

Checkpoints são a forma principal de controlar o andamento do *Crawler*, fazer análises e assegurar a continuidade do projeto caso algo fora do normal aconteça. Ao final de cada ciclo, assim que as URLs coletadas atualizam a tabela, o estado atual para o próximo ciclo fica salvo em disco e pode ser carregado pelo mesmo programa ao ser iniciado. Este momento de transição disponibiliza de várias informações que podem ser analisadas, como quantidade de páginas e domínios vistos, tempo e tamanho da tabela, para cada ciclo. Devido à independência passada para as *Threads* e o modelo desenvolvido, uma quantidade mínima de informações salvas representam todo o contexto até o momento atual e garantem a continuidade como se o programa não tivesse sido interrompido. As únicas informações armazenadas são as da tabela e da lista de domínios já vistos, acompanhados pela quantidade de páginas já vistas para manter sempre atualizada.

Quando o *Crawler* inicia, a primeira coisa que ele faz é buscar por um *Checkpoint* para continuar do último estado interrompido. Caso não encontre, inicia todos os valores a partir das sementes escolhidas *a priori*.

O armazenamento do HTML da página é feito em arquivos de texto. Como foi dito, cada *Thread* se responsabiliza por um arquivo próprio e vão acumulando páginas até que chegam a um máximo escolhido. Quando esse tamanho é alcançado, ela adiciona ao final do arquivo tudo que foi coletado e recomeça o processo. Isso evita que várias tentem escrever no mesmo lugar ao mesmo tempo e varia as informações em lugares diferentes. O padrão de escrita elimina qualquer ocorrência de '|' e segue o formato:

$$|||<url>|<html>|||<url>|\cdots$$

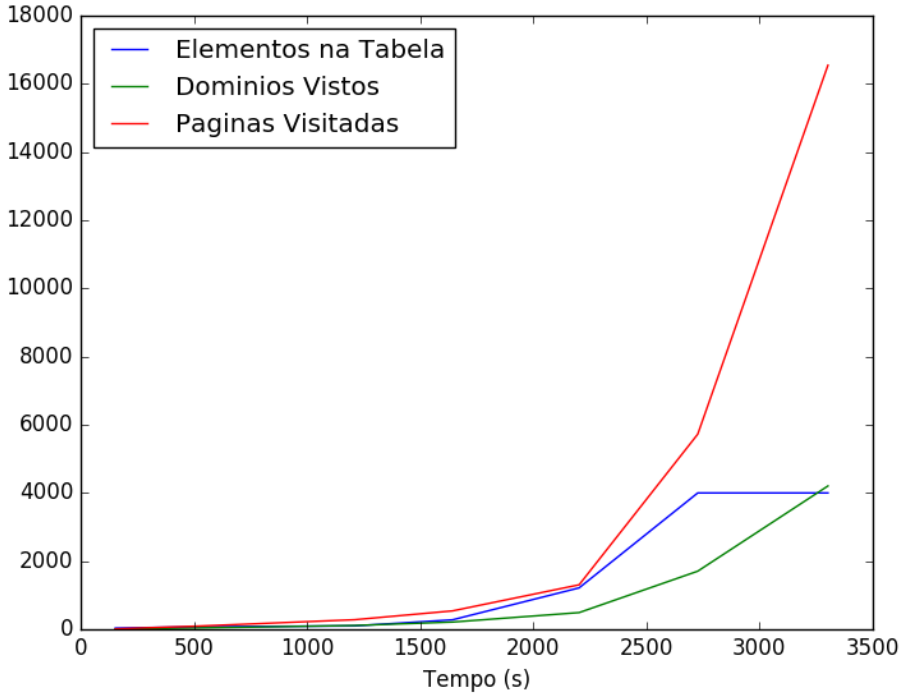

Figura 2. Tempo para encher a tabela.

3. Análise dos Resultados

Alguns testes foram importantes para mostrar a qualidade do modelo escolhido e entender o comportamento a partir da variação de parâmetros. Primeiramente, é fácil notar que, da forma como o modelo foi pensado, ao ser iniciado pela primeira vez com poucos domínios como semente, existe um tempo para que haja domínios suficientes para que todas as *Threads* criadas possam ser usadas. Esse é o tempo necessário para que a tabela esteja cheia o suficiente para controlar a distribuição de URLs. Ao limitar o tamanho dessa tabela a 4000 elementos, podemos observar a Figura 2 e o tempo necessário para que o máximo seja alcançado, acompanhado pela quantidade de páginas coletadas e o total de domínios dessas páginas à medida que a tabela enche. Pode-se notar, como esperado, um começo mais lento seguido por um crescimento rápido assim que mais sementes são usadas.

Uma observação mais profunda desse momento inicial pode ser feita analisando a quantidade de páginas coletadas e os domínios visitados por mais tempo e a quantidade de tempo de espera para que os 30 segundos entre requisições seja respeitado, um dos principais fatores que deixam os primeiros momentos tão lentos. Na Figura 3, observamos um começo lento seguido por um crescimento de páginas coletadas que alcançam trezentas mil extremamente rápido. A Figura 4 mostra quanto tempo de espera as *Threads* tiveram que esperar normalizado pela quantidade de páginas vistas. Fica evidente que no começo cada página teve uma espera muito alta que decresce rapidamente para um tempo razoável de espera por página.

Visto que o tempo inicial é ultrapassado rapidamente, pode-se fazer uma análise da

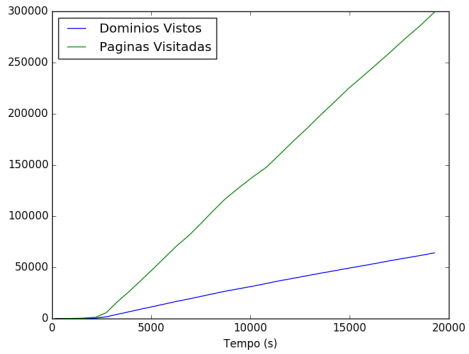

Figura 3. Coleta Inicial

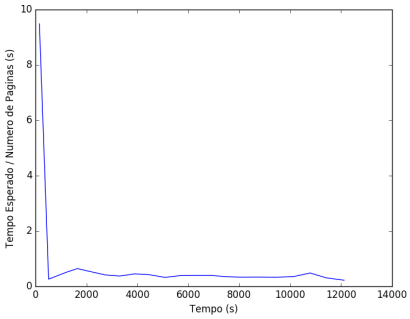

Figura 4. Tempo de Espera Inicial

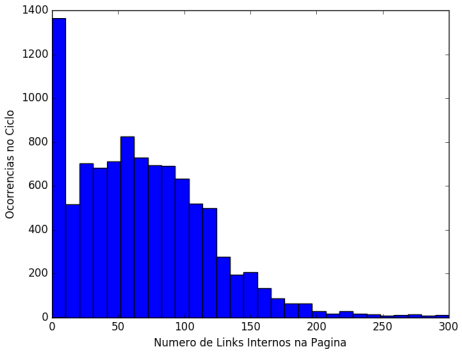

Figura 5. Distribuição de Links Internos

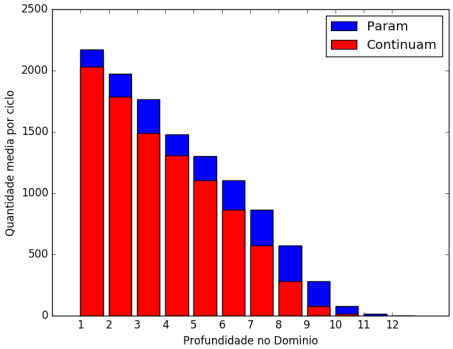

Figura 6. Links Usados

qualidade do modelo e dos parâmetros. Para isso, um *Checkpoint* foi salvo para que todos os testes a partir daqui tenham partido de um mesmo ponto e suas diferenças sejam mais justas. Nesse ponto, pouco mais de quatrocentas mil páginas já haviam sido coletadas com aproximadamente cem mil domínios vistos. Procurou-se usar uma quantidade um pouco mais elevada para ver o comportamento do modelo mesmo com uma lista de domínios visitados maior.

Antes de apresentar a eficiência, algumas características gerais resultantes das decisões de projeto devem ser levadas em conta. No primeiro ciclo do *Checkpoint*, a quantidade de *links* para o mesmo domínio tem uma distribuição apresentada pela Figura 5. Como o objetivo é encontrar, principalmente, o máximo de domínios, o critério de parada apresentado em 2.2.1 foi usado para fazer esse controle. Ao observar a Figura 6, pode-se ver a quantidade média por ciclo de páginas que satisfazem o critério e continuam se aprofundando no mesmo domínio e aquelas que não satisfazem e não precisam continuar. Dessa forma, serve como uma seleção das páginas que trazem maior benefício para o *Crawler*.

Por fim, uma análise do quanto o número de *Threads* e de domínios enviados para cada uma pode influenciar na eficiência. Do mesmo *Checkpoint* citado acima, foram

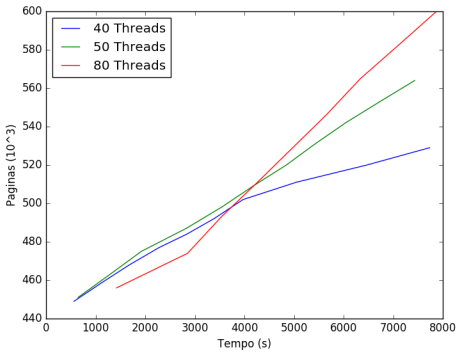

Figura 7. Páginas ao variar Threads

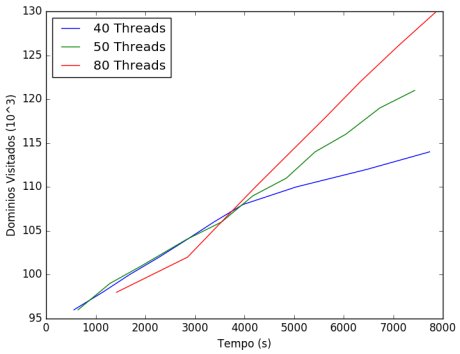

Figura 8. Domínios ao variar Threads

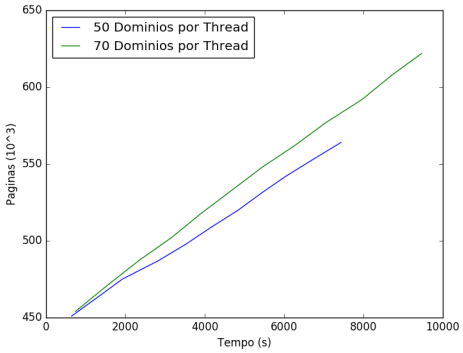

Figura 9. Páginas ao variar número de domínios

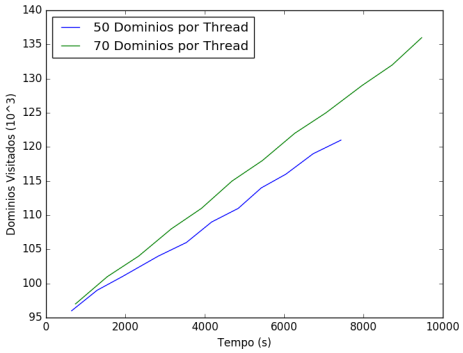

Figura 10. Domínios ao variar número de domínios

feitas variações em ambos.

Figuras 7 e 8 mostram a quantidade de páginas e domínios à medida que a quantidade de *Threads* variou, mantendo a quantidade de domínios em cada. O máximo usado foi 80 pelo fato de ter sido usada uma tabela de comprimento máximo de 4000 URLs. Quanto mais *Threads* são adicionadas, mais páginas são coletadas em um mesmo espaço de tempo. Não se alcançou um máximo onde a organização da quantidade delas prejudique a eficiência por não haver necessidade para os parâmetros escolhidos. Coletou aproximadamente 150×10^3 páginas após pouco mais de 2 horas sem apresentar uma notável redução de velocidade, o que o torna um resultado válido.

A quantidade de domínios enviados para cada *Thread* também influencia diretamente na velocidade em que as páginas são coletadas. As Figuras 9 e 10 apresentam os resultados com a variação desse valor. Vemos que o aumento de domínios apresenta uma melhora considerável tanto na quantidade de páginas vistas como na de novos domínios.

Com essas análises, pode-se entender melhor e controlar os parâmetros com maior eficiência para alcançar qualquer objetivo proposto. De uma forma geral, o *Crawler* consegue percorrer um alto número de páginas e domínios novos em pouco tempo. Além

disso, uma busca satisfatória de forma horizontal, visitando diversos domínios diferentes de uma forma eficaz.

4. Conclusão

Máquinas de buscas são usadas a todo o momento e cada uma é feita especialmente para realizar a função que propõe, podendo ser mais gerais ou específicas. Este projeto tem como principal objetivo a construção da primeira etapa de um máquina busca, o *Crawler*, com um modelo especializado, principalmente, em um caminhamento em largura e análise do seu comportamento.

Foi apresentada uma solução que respeita as regras de etiqueta da *Web* ao mesmo tempo que mantém uma velocidade de coleta suficiente para percorrer uma quantidade elevada de páginas. Além disso, procura manter sua eficiência mesmo depois de ativo por muito tempo, acompanhado por técnicas que possibilitem o uso constante mesmo após ser interrompido.

Algumas estratégias tiveram que ser escolhidas para focar nos objetivos propostos, como o cálculo de *scores* e critérios de parada ao caminhar dentro de um mesmo domínio. A forma de atuação das *Threads* também foram essenciais para dar rapidez sem desrespeitar os servidores.

Por fim, a análise dos resultados foi importante para definir como a forma escolhida e a variação dos parâmetros influenciam de uma forma geral. Isso levou a um melhor entendimento do comportamento visto e como cada fator de influência afeta todo o sistema desenvolvido.

5. Referências

1. Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. Modern information retrieval. Vol. 463. New York: ACM press, 1999.
2. http://homepages.dcc.ufmg.br/ nivio/br/teaching-ri-17.php