Trabalho Prático 2 - Recuperação da Informação

Matheus Aquino Motta[1]

[1]Bacharelado em Matemática Computacional
DCC - Universidade Federal de Minas Gerais

matheusaquino199@gmail.com.br 2018046513

Abstract. *In this report we will briefly discuss the implementation of the assignment 2 of the subject Information Retrieval. The problem consisted into implementing a web crawler algorithm to collect web pages from a given set of seed links and to crawl through at least 100000 different pages and its respective HTML codes, while maintaining statistic information about the average crawling time, web pages size and number of pages crawled from each level 0 page.*

Resumo. *Nesse relatório iremos discutir brevemente a implementação do Trabalho Prático 2 da disciplina Recuperação da Informação. O problema consistia em desevolver um coletor de páginas web para coletar a partir de um conjunto inicial de URLs pelo menos 100000 diferentes páginas e seus respectivos códigos HTML, mantendo informações estatíticas acerca do tempo de crawling médio, tamanho das páginas e o número de páginas coletadas a partir de cada página nível 0.*

1. Introdução

O problema proposto no Trabalho Prático 1 consistia no desenvolvimento de um *crawler* focado na *Web* brasileira. A busca deveria ser estruturada por um escalonador de longo prazo, *Long-Term Scheduler*, que controloria a partir de uma fila de prioridades quais páginas seriam coletadas de modo a favorecer o *crawling* de URLs de domínios variados e *homepages*.

Assim, seja p uma página selecionada pelo escalonador de longo prazo e d_p o seu respectivo domínio, descrevemos o processo de coleta das páginas por meio de um escalonador de curto prazo, *Short-Term Scheduler*, que realiza o crawling das páginas URL referenciadas em p pertencentes ao domínio d_p (páginas de nível 1), enquanto páginas pertencentes a domínios diferentes serão adicionadas ao escalonador de longo prazo (páginas de nível 0).

De um modo geral o programa é desenvolvido em torno desses dois escalonadores, que realizam esse processo até que sejam coletadas 100000 páginas e as informações de coleta sejam devidamente armazenadas. Destarte, iremos discutir brevemente as decisões de implementação tomadas e os resultados apresentados, assim como analises estatísticas dos dados coletados.

2. Implementação

O funcionamento do algoritmo pode ser visualizado a partir do fluxograma abaixo, onde com uso de *multi-threads* iremos crawlear páginas da *Web* de servidores distintos de forma

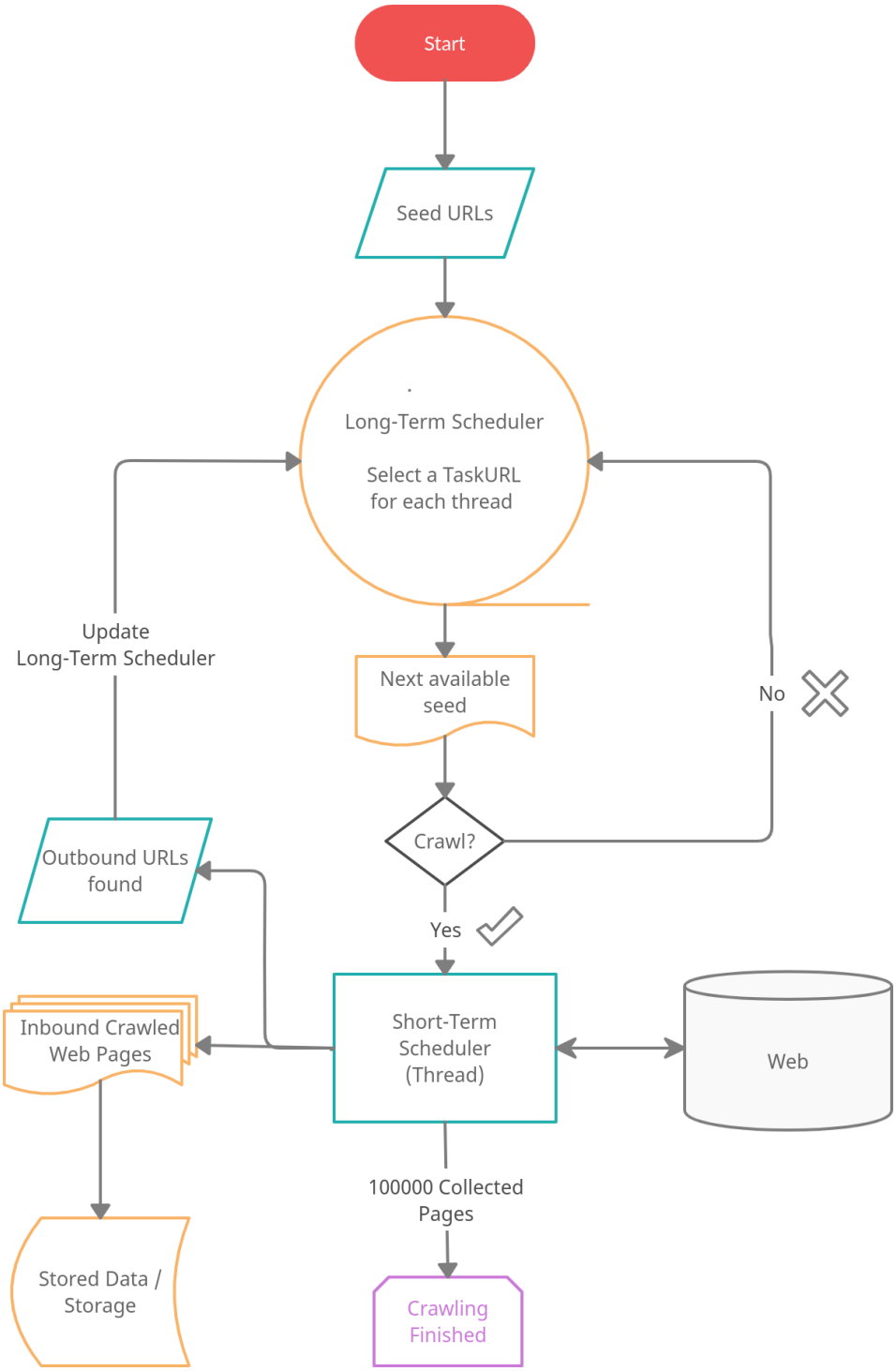

Figure 1. Fluxograma de funcionamento do Crawler.

concorrente com polidez e velocidade, respeitando tempos de requisição entre coletas de páginas para um mesmo domínio.

Assim, para descrever o funcionamento do algoritmo precisamos entender as funções centrais do coletor de *Long-Term Scheduling* e *Short-Term Scheduling*.

2.1. *Long-Term Scheduler*

A função de escalonamento de longo prazo de *Task* URLs, enlaces de nível 0, foi implementada a partir de uma fila de prioridade de tarefas, que recebe enlaces de páginas *Web* para serem coletadas e determina a ordem de coleta por meio de 3 fatores:

- Peso da URL, *URL Weight*, isto é, de acordo com o número de barras, "/", e pontos "." existentes na *string* do *link*, conseguimos inferir quais páginas são as mais próximas à *homepage* do domínio. E como nosso objetivo é priorizar uma busca em largura em servidores distintos, páginas de peso menor são priorizadas.

- Peso do domínio, *Domain Weight*, de modo a mitigar a possibilidade de que duas ou mais *threads* façam requisições de forma concorrente a um mesmo domínio aferimos ao dominio da página o número de URLs já visitadas nele. Maximizando a variedade de domínios visitados ao longo da execução do algoritmo.

- Localidade do domínio, *Brazilian Domains*, para priorizar a busca em domínios potencialmente brasileiros, URLs com o a *substring ".br"* tem seu peso diminuido, a fim de priorizar potenciais páginas da *Web* brasileira.

Assim, ao longo do algoritmo iremos priorizar a coleta de sítios *web* mais próximos a páginas raiz de domínios variados e preferencialmente pertencentes a servidores potencialmente brasileiros.

2.2. *Short Term Scheduler*

Diferentemente do escalonador de longo prazo que consiste fundamentalmente em uma fila de prioridades, o escalonador de curto prazo é a função que realiza efetivamente o *crawling* das páginas e continuidade do algoritmo.

Dessa forma, seja p uma página apropriadamente retirada do escalonador de longo prazo para ser coletada e d_p o seu respectivo domínio. No *Short-Term Scheduler* realizamos o *crawling* de p armazenando seu HTML em disco e obtemos listas dos enlaces referenciados na página separados em conjuntos *inbound links* e *outbound links*. Assim, definimos P_{in} e P_{out} como os conjuntos de *inbound* e *outbound* links respectivamente como

$$P_{in} = \{p_i \in p \mid d_p \neq d_{p_i}\}$$
$$P_{out} = \{p_j \in p \mid d_p \neq d_{p_j}\}$$

Onde $0 \leq i \leq |P_{in}|$ e $0 \leq j \leq |P_{out}|$.

Isto é, URLs *inbound* são aquelas que compartilham o mesmo domínio da URL retirada inicialmente do escalonador de longo prazo, enquanto as *outbound* pertencem a domínios externos.

Nesse sentido, a primeira sub-rotina do *Short-Term Scheduler* executada é a atualização, *update*, da fila de prioridades, a partir da apropriada inserção do conjunto de URLs do conjunto P_{out}, URLs de nível 0, ao *Long-Term Scheduler* para serem potenciais novas *seeds* de coleta para próximas execuções do *Short-Term Scheduler*.

Por conseguinte, a segunda etapa do escalonador de curto prazo é executada realizando o *crawling* das páginas pertencentes a P_{in} (páginas de nível 1), assim como seus códigos

HTML e *logs* de informação a respeito da coleta, como o tamanho da página e tempo de execução do *crawling*.

2.3. Fluxo de execução

Agora que entendemos a parte central do funcionamento do algoritmo podemos discutir o fluxo de execução e decisões de implementação tomadas com mais clareza, de acordo com o fluxograma apresentado acima.

Inicializamos o algoritmo incluindo ao escalonador de longo prazo as URLs seed dadas na especificação do projeto, exemplo: *https://www.cnnbrasil.com.br/*, "*https://ufmg.br/*", "*https://www.folha.uol.com.br/*". Onde aferimos o devido peso a URL e ao domínio ao qual o enlace pertence.

Nesse sentido, serão inicilizadas T threads, onde por limitações de hardware $T \leq 20$, que irão executar a função de *Short-Term Scheduling* de forma concorrente, que por sua vez irá retirar de forma apropriada uma nova *URL* de nível 0 do escalonador de longo prazo. Assim, seja u_s a URL retirada, verificamos antes de iniciar a busca se o domínio ao qual o enlace pertence não está sendo pesquisado em outra *thread*, caso esteja u_s é adicionado à uma pilha e tomamos o próximo enlace disponível, esse processo é repetido até que encontremos uma URL viável para busca ou não hajam mais URLs na fila. Além disso, caso a busca não possa ser iniciada mantemos a *thread* em *sleep* por 2 segundos e repetimos o processo por até 50 vezes, deletando a *thread* ociosa.

Agora, com sucesso na última etapa, inicializamos o *Spider* a partir da URL u_s e seu respectivo domínio, nesse sentido caso a página não tenha sido visitada anteriormente executamos o primeiro *crawling* da rotina. Desse modo, caso a coleta tenha sido executada com sucesso, salvamos o conteúdo HTML da página, assim como demais informações relevantes de tempo e espaço. Em seguida, realizamos o *update* do *Long-Term Scheduler* a partir dos enlaces *outbound* pertencentes a P_{out}, URLs de nível 0, encontrados e executamos a coleta das das páginas HTML de nível 1 referenciadas em u_s, *inbound links* pertencentes a P_{in}, assim como seus respectivos dados de coleta.

Repetimos essa rotina até que 100000 páginas sejam coletadas e todas as informações acerca da execução, códigos HTML e dados de tempo/espaço sejam armazenados.

2.4. Bibliotecas e Estruturas de Dados

Durante a implementação do projeto foi feito expressivo uso da biblioteca *Chilkat* e da *Standart Library (STL)* do C++, fundamentais para o *Crawling* das páginas *Web*, armazenamento e acesso de dados de maneira prática e minimamente eficiente. Dentre elas as estruturas *Map* e *Set* foram fortemente utilizadas, de modo a acessar informações relativas a URLs e domínios em complexidade de tempo da ordem $O(\log N)$, onde N é o número de elementos inseridos na estrutura, assim como a *priority queue* utilizada para a implementação do escalonador de longo prazo que possui complexidade de inserção e remoção também da ordem de $O(\log M)$ onde M é o número de elementos na fila.

Além disso, as biblioteca *Thread* e *Mutex* foram utilizadas para a realização do *crawling* em *multithreads*, de modo que cada *thread* executa um *Short-Term Scheduler* e tem o acesso a váriaveis globais estáticas e estruturas compartilhadas entre as múltiplas tarefas de acesso *thread-safe* assegurado por váriaveis *Mutex* que bloqueiam o acesso de mais de uma *thread* à uma estrutura ou váriavel compartilhada.

3. Resultados

Os resultados apresentados pelo algoritmo foram obtidos realizando a execução do *crawler* para a coleta de 100000 páginas *Web* a partir das seguintes *seeds* URLs.

1. *https://ufmg.br/*
2. *https://www.cnnbrasil.com.br/*
3. *https://www.folha.uol.com.br/*

O programa foi executado com o uso de 20 *threads*, devido à limitações de *hardware* de 2 núcleos de processamento, e duas amostragens de páginas foram coletadas

- Geral, onde não havia um grande enfoque em tomar páginas da *Web* brasileira, e o critério de priorização para potenciais páginas em domínio brasileiro do escalanador de longo prazo foi implementado para subtrair 1 unidade de peso de *Task* URLs que possuem a *substring ".br"*. Assim garantindo a priorização de uma URL com sufixo ".com.br/" comparada a uma URL de sufixo ".com/".

- Brasileira, nessa abordagem o enfoque em tomar páginas de domínios brasileiros foi tomado de forma mais agressiva, no qual o escalonador de longo prazo implementado foi modificado de modo a subtrair um número arbitrariamente grande do peso de *Task* URLs potencialmente brasileiras, isto é, as que possuem a *substring ".br"*. Desse modo garantindo priorização total de páginas potencialmente brasileiras na busca do coletor, ainda com respeito ao peso entre elas.

A diferença entre as amostragens torna-se notável quando comparados os números de páginas potencialmente brasileiras coletadas em cada variação. O coletor de enfoque "Geral" alcançou e coletou um total de 5574 páginas com o sufixo desejado *".br"*, enquanto o coletor de enfoque "Brasileiro", coletou todas as 100000 páginas desejadas em domínios potencialmente brasileiros.

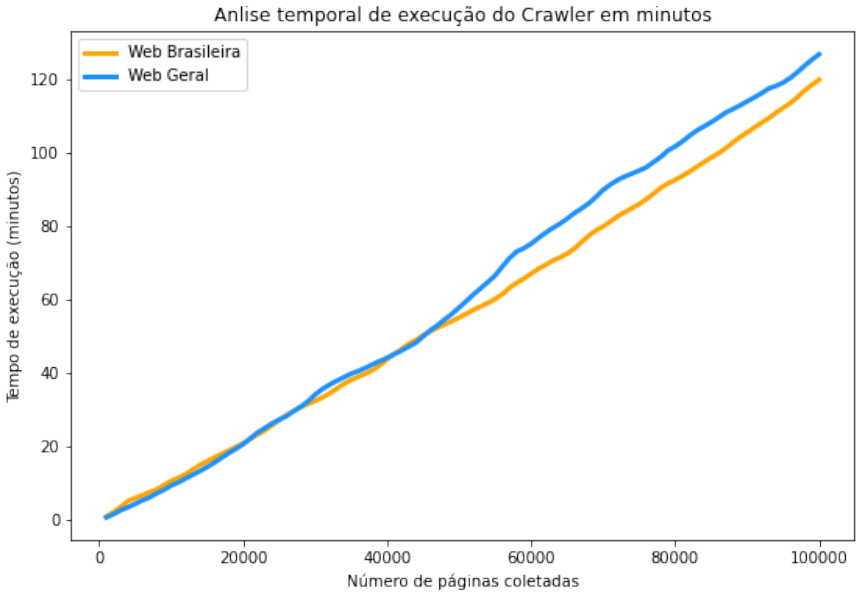

Figure 2. Gráfico tempo de execução total do algoritmo.

O gráfico acima apresenta resultados acerca do tempo de execução do algoritmo, que tomou cerca de 2 horas para ser executado em ambos os casos. Assim como a análise

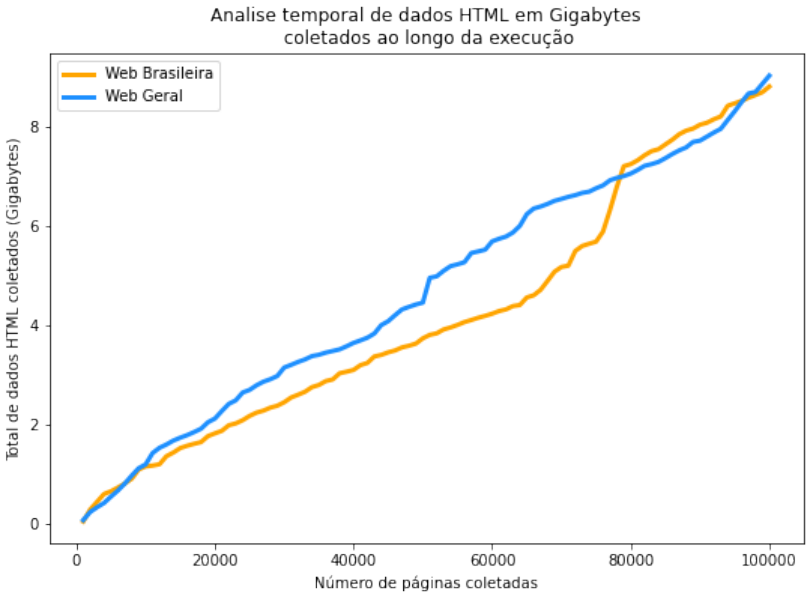

Figure 3. Dados HTML abaixados ao longo da execução do algoritmo.

temporal, a análise de tamanho dos dados HTML coletados também ocorreu de forma similar para as duas abordagens, onde foram coletados cerca de 9 GB de arquivos HTML.

Analisamos dados individuais de cada *Task URL*, páginas de nível 0, como por exemplo, o tamanho das URLs pesquisadas retiradas do *Long-Term Scheduler*, que de um modo geral apresentam pesos pequenos, indicando estarem mais próximas a *homepages*. Porém é notável a diferença entre as abordagens, haja vista que o caso ”Brasileiro” apresenta URLs maiores de sufixos/*substrings* *”.com.br”*, haja vista que prioriza páginas brasileiras, que por sua vez não são necessariamente as com menor peso.

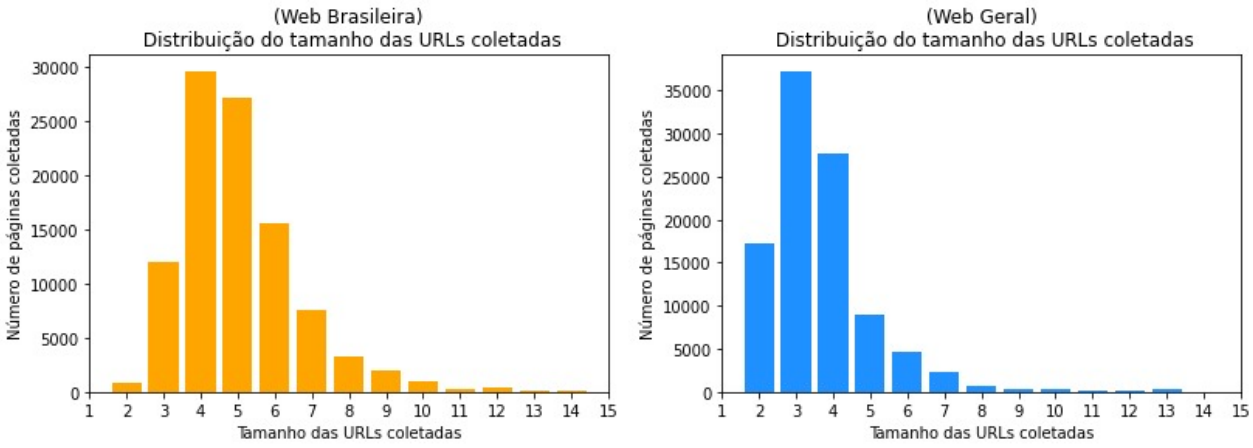

Figure 4. Distribuição do peso das URLs de nível 0 coletadas.

Os gráficos abaixo fazem uma análise mais específica acerca das distribuições médias de páginas nível 1 coletadas em cada página de nível 0, como informações relativas ao número de páginas encontradas por *Task*, tamanho médio, e tempo de coleta.

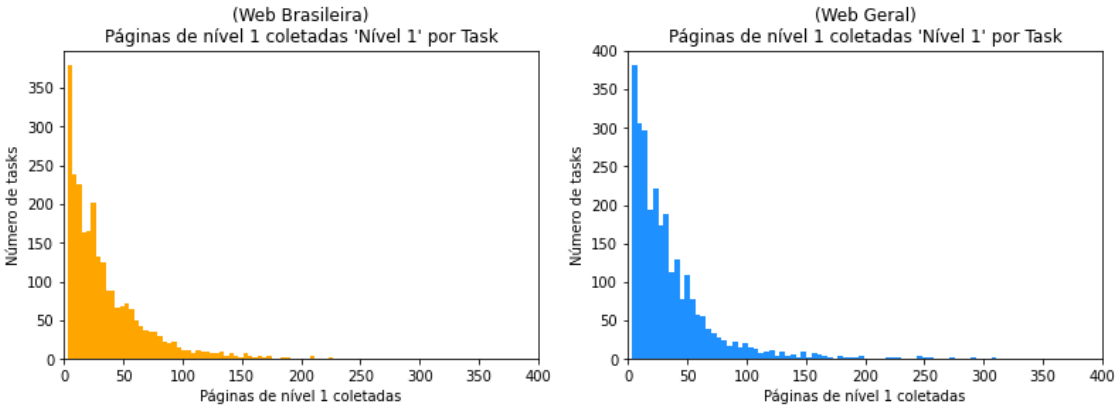

Figure 5. Distribuição do número de páginas nível 1 coletadas em cada Task, nível 0.

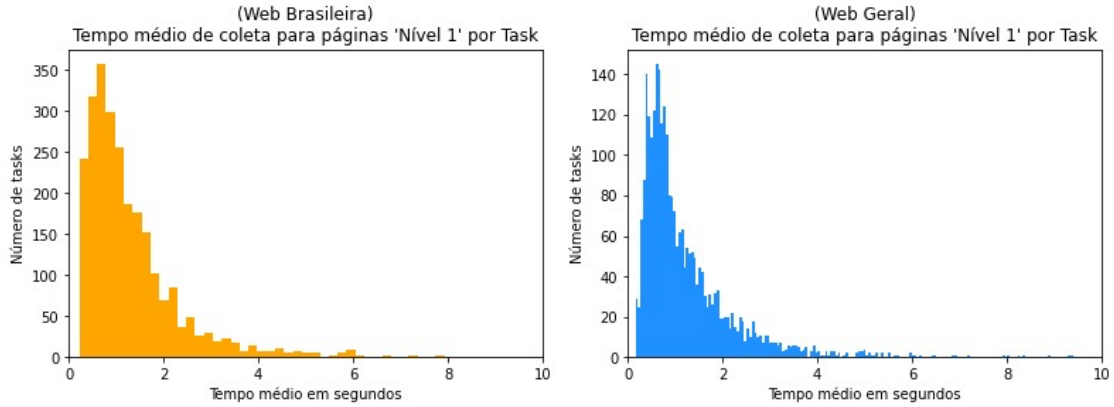

Figure 6. Distribuição do tempo médio de crawling para páginas de nível 1 em cada Task.

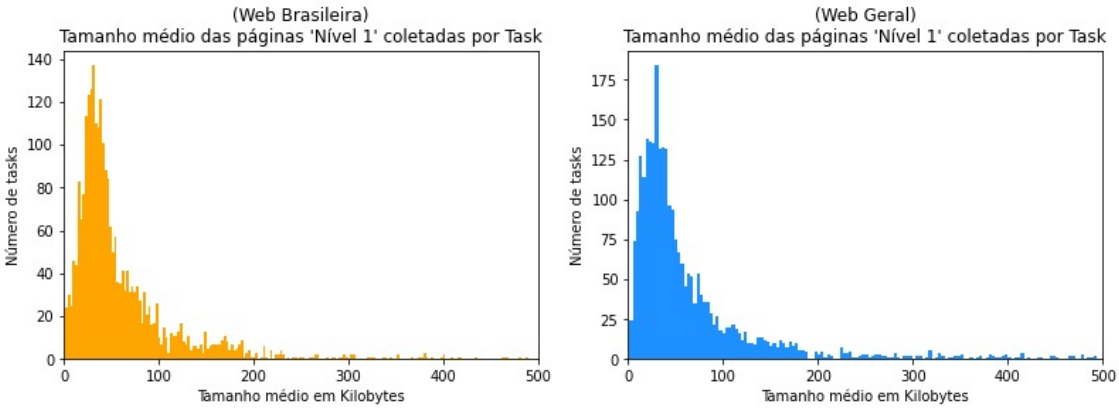

Figure 7. Distribuição do tamanho médio das páginas de nível 1 em cada Task.

Ao fim obtivemos uma média de aproximadamente $\mu = 35$ páginas de nível 1 coletadas para cada página de nível 0, e um desvio padrão de $\sigma = 41$ para ambas as abordagens.

Além disso, para o tempo de execução médio do *crawling* de páginas Nível 1, a partir de uma *Task* URL de Nível 0, é perceptível a formação de uma ”normal” próxima a 1 segundo para ambos os casos, como será enfatizado na análise individual das páginas.

Uma análise similar pode ser feita a respeito do tamanho médio das páginas coletadas

em cada *Task*, com a formação de uma ”normal” próxima a 50 kilobytes para cada página HTML de nível 1 coletada com sucesso.

Por fim, podemos visualizar de uma maneira mais ampla os resultados a partir de distribuições relativas ao tempo de coleta e tamanho individual de cada página.

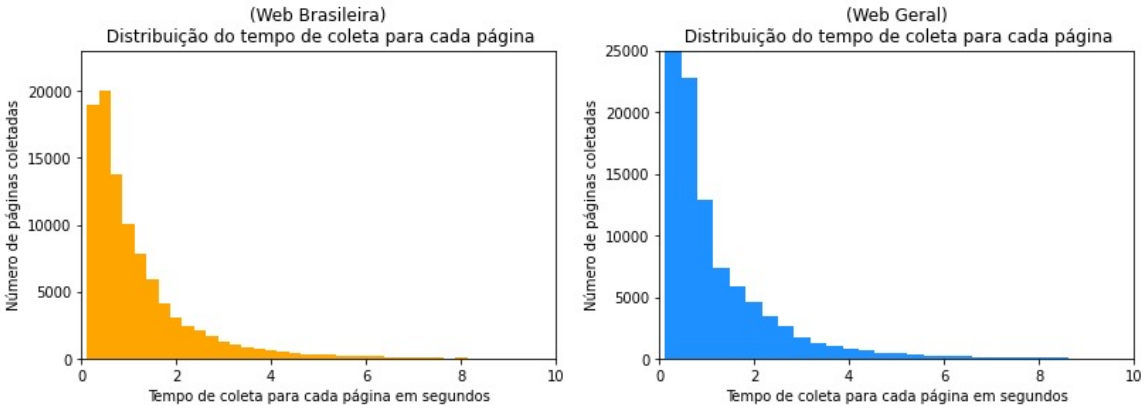

Figure 8. Distribuição parcial do tempo de crawling individual para páginas de nível 1 e nível 0.

Onde temos para a abordagem prunada para domínios potencialmente brasileiros um tempo médio de *crawling* e desvio padrão em segundos de

$$\mu_{t_{Br}} = 1.372$$
$$\sigma_{t_{Br}} = 2.829$$

entre 3711 páginas de nível 0 e 96289 páginas de nível 1. Enquanto para o caso ”geral” temos, entre 3446 páginas de nível 0 e 99654 de nível 1, e média e desvio padrão iguais a

$$\mu_{t_{Gr}} = 1.408$$
$$\sigma_{t_{Gr}} = 5.917$$

Assim, embora as diferenças parciais na distribuição sejam pequenas, a maior presença de páginas *outliers* na busca ”geral” provoca um tempo médio de coleta e desvio padrão superiores as do caso focado na *Web* brasileira, assim como possíveis maiores variações no tempo de requisição e latência de servidores externos.

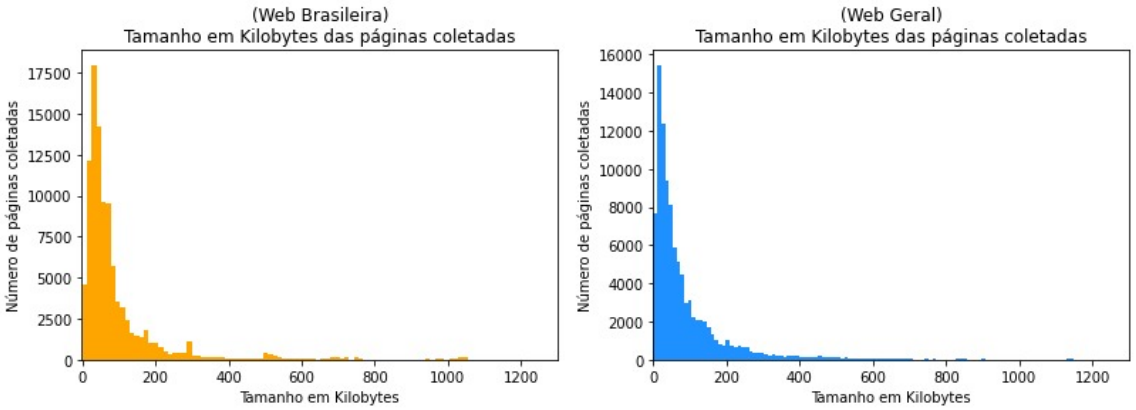

Figure 9. Distribuição parcial do tamanho individual das páginas de nível 1 e nível 0 em kilobytes.

E por fim, temos uma média e desvio padrão para o tamanho das páginas em kilobytes para a abordagem ”brasileira” e ”geral” respectivamente igual a

$$\mu_{s_{Br}} = 97.295 \qquad \sigma_{s_{Br}} = 163.708$$
$$\mu_{s_{Gr}} = 95.849 \qquad \sigma_{s_{Gr}} = 167.715$$

Que por sua vez são resultados similares, o que evidencia a semelhança no tamanho de páginas HTML em geral e a presença de *outliers* em ambos casos, esses notabilizadas pelo alto desvio padrão de ambas as coletas.

4. Conclusões

Nesse trabalho apresentamos o desenvolvimento de um *crawler* que deveria coletar 100000 páginas preferencialmente focadas na *Web* brasileira. Embora a proposta inicial do trabalho seja aparentemente simples a execução e implementação exigiram esforço e cuidados singulares, desde implementar e executar o programa de maneira limpa, eficiente e polida, fazendo o uso de *multi-threads*, até adquirir dados e informações a respeito de cada página de nível 1 e nível 0 coletada de forma apropriada.

Algumas dificuldades encontradas centraram-se principalmente em especificidades da biblioteca *Chilkat* e excessões que deveriam ser tratadas na busca, como o encontro de URLs quebradas ou não existentes. Além disso, o uso de *multi-threads* foi um desafio devido à limitações de *hardware* e necessidade de balanço entre polidez na busca e eficiencia. Dificuldades que foram superadas e contribuiram significativamente para o desenvolvimente do trabalho e aprendizado.

O primeiro resultado obtido efetivamente, caso ”geral”, apresentou sucesso na coleta das páginas *Web* e dados requeridos, entretanto falhou em coletar páginas focadas em domínios brasileiros, enquanto a segunda abordagem apresentou resultados relativamente superiores apresentando melhora no tempo médio de coleta por página e desvio padrão, coletando apenas páginas potencialmente pertencentes a *Web* brasileira. Assim, as comparações e análises realizadas demonstraram-se importantes para novas decisões de engenharia e futuras tentativas de melhora do algoritmo.

Algumas alternativas para a melhora na eficiencia do $crawler$ poderiam ser atingidas a partir do uso de técnicas como *String Hashing* para obter acesso as informações sobre URLs já pesquisadas, domínios visitados, etc em complexidade de tempo inferior a $O(\log N)$ dado pelas bibliotecas STL do C++ ou otimizações relativas a fila de prioridades do escalonador de longo prazo.