UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

KLAIRTON DE LIMA BRITO

ANÁLISE DE SENSIBILIDADE DO TEMPO DE EXECUÇÃO DO ALGORITMO TOP-K ACELERADO POR FILTRO DE CARDINALIDADE

QUIXADÁ
2014

KLAIRTON DE LIMA BRITO

ANÁLISE DE SENSIBILIDADE DO TEMPO DE EXECUÇÃO DO ALGORITMO TOP-K ACELERADO POR FILTRO DE CARDINALIDADE

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientador Prof. Dr. Críston Pereira de Souza

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

B875a Brito, Klairton de Lima
Análise de sensibilidade do tempo de execução do algoritmo TOP-K acelerado por filtro de cardinalidade / Klairton de Lima Nobre. – 2014.
36 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2014.
Orientação: Prof. Dr. Críston Pereira de Souza
Área de concentração: Computação

1. Algoritmos computacionais 2. Conjunto – Teoria – Aplicações 3. Recuperação da informação I. Título.

CDD 519.6

KLAIRTON DE LIMA BRITO

ANÁLISE DE SENSIBILIDADE DO TEMPO DE EXECUÇÃO DO ALGORITMO TOP-K ACELERADO POR FILTRO DE CARDINALIDADE

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / novembro / 2014.

BANCA EXAMINADORA

_____
Prof. Dr. Críston Pereira de Souza (Orientador)
Universidade Federal do Ceará-UFC

_____
Prof. MSc. Lucas Ismaily B. Freitas
Universidade Federal do Ceará-UFC

_____
Prof. MSc. Ticiana Linhares Coelho da Silva
Universidade Federal do Ceará-UFC

Aos meus pais, que sempre me apoiaram e me ensinaram a superar as dificuldades.

AGRADECIMENTOS

A Universidade Federal do Ceará, seu corpo docente, direção e administração pela oportunidade de fazer o curso.
Ao Prof. Dr. Críston Pereira de Souza, pela orientação, apoio, confiança e empenho dedicado à elaboração deste trabalho.
Aos Professores Camilo Camilo Almendra e Wladimir Araújo Tavares, por terem me dado a oportunidade de trabalhar com eles durante o período que fui bolsista de iniciação à docência.
A minha família, que sempre me apoiou e até hoje é minha fortaleza.
A Camila, pela amizade e companheirismo.
Aos amigos, que fiz antes e durante minha graduação e intercâmbio, meus irmãos na amizade que fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza.
A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

RESUMO

Neste trabalho realizamos uma análise de sensibilidade do tempo de execução do algoritmo Top-k acelerado por filtro de cardinalidade. O algoritmo Top-k é empregado por alguns motores de busca para informar os termos que são mais relevantes nos documentos retornados por uma consulta. Para executar esta tarefa, o Top-k precisa realizar a interseção de conjuntos grandes, que neste contexto é considerado uma tarefa computacionalmente custosa. O filtro de cardinalidade foi proposto na literatura para acelerar o Top-k, fornecendo de forma eficiente um limite superior para o tamanho da interseção, e com isso poupando o Top-k de computar o tamanho exato da interseção em muitos casos. Neste trabalho consideramos um algoritmo fictício cujo tempo de execução cai exponencialmente com a piora do limite superior, e que possui o mesmo tempo de execução do filtro de cardinalidade quando seu limite superior tem a mesma qualidade. Concluímos através dos experimentos que o algoritmo fictício, embora mais flexível, não melhorou significativamente o tempo de execução do Top-k, mostrando que não existe muita margem para abordagens alternativas ao filtro de cardinalidade.

Palavras chave: Algoritmos computacionais. Conjunto – Teoria – Aplicações. Recuperação da informação.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Um algoritmo é uma sequência de passos bem definidos que recebem dados, denominados de entrada, e executam um processamento que transforma os dados em uma saída, de acordo com uma especificação do problema (CORMEN et al., 2002, p 3). Os algoritmos estão presentes no dia-dia, desde em equipamentos de alta tecnologia, como em sistemas de controles de voos em aeroportos, passando pelos produtos domésticos, como controladores de temperatura em geladeiras, e chegando até a uma busca na Internet. Apesar de parecer simples e rápida uma busca na Internet, a quantidade de informação acessada na busca é enorme. Por isso existem trabalhos de pesquisa na área que visam melhorar o desempenho desse processamento.

Alguns motores de busca acrescentam ao resultado das consultas uma série de termos fortemente relacionados à consulta. O algoritmo Top-k é utilizado para a tarefa de identificar estes termos fortemente relacionados. Estes termos são justamente os que aparecem em mais documentos dentre os retornados como resultado da consulta. Para o funcionamento deste algoritmo é calculado o tamanho da interseção entre o conjunto de documentos que contém cada termo conhecido e o conjunto de documentos retornados pela consulta. Para isso são utilizados algoritmos para determinar o valor exato do tamanho da interseção entre conjuntos.

Takuma e Yanagisawa (2013) buscaram melhorar o tempo de execução do algoritmo Top-k com o auxílio de uma estrutura proposta por eles para o cálculo de um limite superior do tamanho da interseção entre conjuntos, chamada de “filtro de cardinalidade”. Neste trabalho foi reproduzido o experimento apresentado em Takuma e Yanagisawa (2013), que utiliza uma base de dados sintética para comparar o filtro de cardinalidade com um algoritmo para determinar o valor exato do tamanho da interseção entre conjuntos, proposto por Ding e König (2011) (chamado “interseção via partições”). Foi construída uma base de dados real que possibilita a aplicação do algoritmo Top-k utilizando as estruturas propostas por Takuma e Yanagisawa (2013) e Ding e König (2011), com o objetivo de simular a execução de um algoritmo fictício que permita realizar análise de sensibilidade do algoritmo Top-k com filtro de cardinalidade. Estes resultados podem motivar outros pesquisadores a buscar melhor desempenho para o algoritmo Top-k através de algoritmos melhores para estimar os limites superiores para o tamanho da interseção entre conjuntos.

Para alcançar os resultados, o desenvolvimento do trabalho foi dividido em três etapas. A primeira etapa é a reprodução do estudo de Takuma e Yanagisawa (2013) com base de dados gerada aleatoriamente. Resumidamente, nesta etapa as estruturas "filtro de cardinalidade" e "interseção via partições" são implementadas e comparadas utilizando uma base de dados gerada a partir de algumas especificações fornecidas por Takuma e Yanagisawa (2013). Na segunda etapa foi construída uma base de dados real, que consiste em páginas web de sites de universidades brasileiras. Nesta etapa também consta a implementação do algoritmo Top-k com a utilização das estruturas "filtro de cardinalidade" e "interseção via partições", e a execução do mesmo utilizando a base de dados real. A terceira etapa é onde a função que simula a execução de um algoritmo fictício é criada e aplicada ao algoritmo Top-k, e seu resultado é comparado com sua execução utilizando as estruturas "filtro de cardinalidade" e "interseção via partições".

É descrita agora como a monografia está estruturada. Na Seção 2 são contextualizados alguns trabalhos da literatura para determinar o tamanho exato ou limite superior para o tamanho da interseção entre conjuntos. São descritos alguns conceitos chaves, e alguns algoritmos utilizados neste trabalho são explicados mais detalhadamente para uma melhor compreensão. Na Seção 3 são descritos detalhadamente todos os processos realizados e ferramentas utilizadas na execução deste trabalho. Na Seção 4 são descritos e comparados os resultados. Na última seção são descritos os pontos de melhorias, objetivos atingidos, as principais conclusões e dificuldades encontradas neste trabalho.

2 TRABALHOS RELACIONADOS E FUNDAMENTAÇÃO TEÓRICA

Como o algoritmo Top-k apresenta uma estreita relação com tamanho da interseção entre conjuntos, trabalhos que apresentem melhoria nesta área podem impactar na melhoria de desempenho do Top-k. Um trabalho que pode ser citado é o de Ding e König (2011), que aborda o problema de determinar o tamanho exato da interseção entre conjuntos de maneira eficiente. Para isso, foram propostas abordagens que dividem os conjuntos em partições, utilizando um mapeamento de bits para cada partição, e um mapeamento inverso para recuperar elementos específicos das partições. Essas abordagens serão descritas mais detalhadamente na Seção 2.2.

A interseção de grandes conjuntos é um problema comum no âmbito da avaliação de consultas booleanas a um motor de busca (BARBAY et al., 2006, p. 146). Computar o tamanho da interseção entre conjuntos pequenos pode ser feito em tempo satisfatório. Porém,

quando esses conjuntos são grandes, como os que são geralmente encontrados em motores de buscas ou em grandes bases de dados públicas ou privadas, torna-se uma tarefa que demanda um grande recurso computacional, e por isso melhorias pequenas podem proporcionar grande redução no tempo de execução.

Essas melhorias vêm sendo propostas ao decorrer do tempo, como podem ser vistas nos trabalhos de Blelloch e Reid-miller (1998) e Brown e Tarjan (1979), que propõem o uso de árvores balanceadas para determinar a interseção entre conjuntos, e nos trabalhos de Tatikonda et al. (2009) e Tsirogiannis et al. (2009) que abordam o uso de multi-processadores para agilizar o processamento.

O trabalho desenvolvido por Takuma e Yanagisawa (2013) demonstra a utilização de um algoritmo que fornece um limite superior para o tamanho da interseção entre conjuntos, que pode ser utilizado para reduzir o tempo de processamento do algoritmo Top-k, que é um algoritmo que fornece os K termos que apresentam a maior interseção com os documentos retornados pelo motor de busca após uma consulta.

Alguns algoritmos propostos que podem ser utilizados para determinar o limite superior e o tamanho exato da interseção entre conjuntos são apresentados nas Seções 2.2 e 2.3. Na Seção 2.4 fornecemos a descrição do funcionamento do algoritmo Top-k.

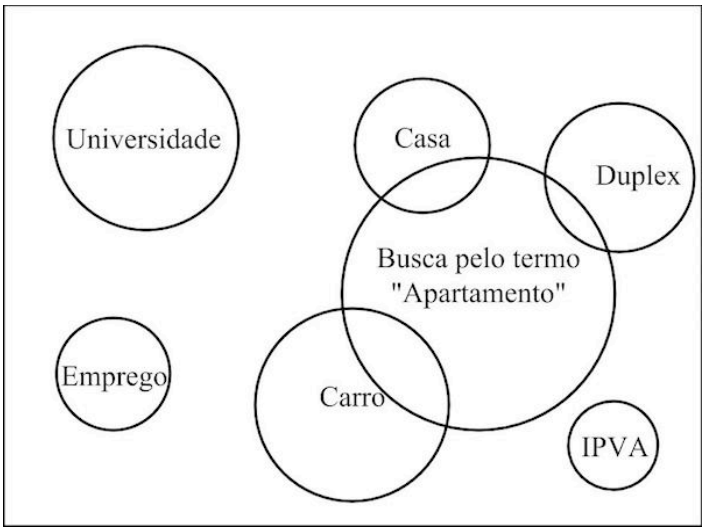
2.1 Interseção de conjuntos

Rosen (2009, p. 121) define interseção de conjuntos da seguinte forma: “Sejam A e B conjuntos. A *interseção* dos conjuntos A e B, denotada por $A \cap B$, é o conjunto de todos os elementos que estão em A e B, simultaneamente”. Corrobora Gersting (2004, p. 136): “a interseção de A e B, denotada por $A \cap B$, é $\{x|x \in A \text{ e } x \in B\}$”.

Um exemplo de utilização de interseção de conjuntos na área da computação é quando efetuamos uma pesquisa em algum motor de busca na Internet. Juntamente ao resultado da pesquisa, também são apresentados outros termos relacionadas com a consulta, e esses termos podem ser obtidos através da interseção de conjuntos. Por exemplo, efetuando uma busca pela palavra “Apartamento”, como ilustra a Figura 1, é retornado um conjunto que contém todos os documentos onde a palavra “Apartamento” ocorreu. Para determinar os termos relacionados, basta selecionar os termos com mais ocorrências dentre os documentos retornados pela busca. Para obter estes termos é necessário calcular, para cada termo T, o

tamanho da interseção entre o conjunto dos documentos contendo T e o conjunto dos documentos retornados pelo motor de busca.

Figura 1 – Interseção de conjuntos para uso de buscas relacionadas



Fonte: Elaboração própria.

2.2 Algoritmos para cálculo exato do tamanho da interseção entre conjuntos

"Algoritmos criados para resolver o mesmo problema muitas vezes diferem de forma drástica em sua eficiência. Essas diferenças podem ser muito mais significativas que as diferenças relativas a hardware e software" (CORMEN et al., 2002, p 7). A eficiência torna-se algo indispensável quando é necessário determinar o tamanho da interseção entre conjuntos grandes. O valor exato do tamanho da interseção entre dois conjuntos pode ser obtido, por exemplo, a partir dos algoritmos a seguir:

Força Bruta é um algoritmo que, para cada elemento presente em um conjunto A, efetua uma comparação com todos os elementos de um conjunto B; sendo assim um algoritmo com tempo de execução de ordem O(|A|*|B|). Entretanto, este tempo de execução pode ser quadrático quando os tamanhos dos conjuntos são da ordem do tamanho da entrada. Por exemplo, se N= |A|+|B| é o tamanho da entrada e |A|=N/2 e |B|=N/2, teremos um tempo de execução da ordem de $|A|*|B| = N^2/4$.

Merge Linear é um algoritmo que inicialmente ordena os conjuntos de forma crescente utilizando algoritmos de ordenação que apresentam tempo de execução de ordem O(N log N). Supondo que tais conjuntos sejam A e B, o próximo passo é a utilização de dois ponteiros p_1 e p_2, que inicialmente apontam para o menor elemento de A e B, respectivamente. Esses ponteiros são utilizados para que os dois conjuntos sejam percorridos

de forma simultânea (do menor elemento para o maior elemento). Caso p_1 e p_2 apontem para elementos com mesmo valor, ou seja, o elemento de A que p_1 aponta é igual ao elemento de B que p_2 aponta, esse elemento é adicionado na interseção, já que está presente tanto no conjunto A quanto no conjunto B; e então um dos dois ponteiros será incrementado. Caso p_1 e p_2 apontem para elementos de valores distintos, o ponteiro que apontar para o elemento de menor valor será incrementado. Quando p_1 ou p_2 percorrer todo seu respectivo conjunto, o algoritmo é encerrado. Como inicialmente os conjuntos foram ordenados, nenhum elemento que esteja no conjunto que não foi totalmente percorrido pertencerá à interseção. Como a etapa de percorrer os conjuntos pode ser feita em tempo linear, concluímos que o Merge Linear tem tempo de execução O(N log N).

Busca Binária é um algoritmo que primeiramente ordena o maior conjunto (assuma que este maior conjunto seja B), e então, para cada elemento em A, é feita uma busca binária no conjunto B para determinar se o elemento pertence ou não à interseção. O tempo de execução desse algoritmo tem ordem $|A|*\log_2 |B| + |B|*\log_2 |B|$, e portanto um desempenho melhor que o algoritmo força bruta quando os tamanhos dos conjuntos A e B não apresentam grande diferença. Entretanto quando A e B apresentam tamanhos muito diferentes, o algoritmo perde sua eficiência comparado com o força bruta.

DK é um algoritmo proposto por Ding e König (2011) que entre os algoritmos para determinar o tamanho exato da interseção é o que apresenta melhor desempenho na prática, embora não forneça a melhor complexidade de tempo na análise teórica. O melhor limite teórico é fornecido por Bille, Pagh e Pagh (2007), e vale $O((|A|+|B|)\log^2 w/w + |A \cap B|)$, (onde w representa o tamanho da palavra de máquina). Descrevemos a seguir duas variações do DK: com partições fixas e com partições randomizadas.

O algoritmo de interseção utilizando partições fixas funciona da maneira explicada a seguir. Supondo que L_1 e L_2 são dois conjuntos quaisquer. Estes conjuntos são ordenados e divididos em partições no pré-processamento em partições L_i^j de tamanhos $\sqrt{w}$ (onde w representa o tamanho da palavra de máquina), exceto a última partição, onde o tamanho pode ser menor que $w^{1/2}$. Então, para cada partição L_i^j , temos uma representação por uma palavra de w bits, denotada por $h(L_i^j)$, cada elemento x desta partição torna o bit h(x) igual a 1, h é uma função de *hash* que mapeia os elementos da partição no intervalo [0,w-1]. Os outros bits mantêm valor zero. Para cada bit y com valor 1 em $h(L_i^j)$, armazenamos também uma lista de todos os elementos cujo valor da função de *hash* é igual a y (ou seja, lista de colisões). Com esta lista, podemos fazer um mapeamento inverso da função de *hash*,

ou seja $h^{-1}(y, L_i^j) = \{ x \mid x \in L_i^j \text{ e } h(x) = y\}$. O estágio conhecido como *processamento online* consiste no algoritmo 1 descrito abaixo.

Algoritmo 1 – Algoritmo de partições fixas

```
1: p ← 1, q ← 1, Δ ← ∅
2: while p ≤ n1 and q ≤ n2 do
3:     if inf(L2^q) > sup(L1^p) then
4:         p ← p + 1
5:     else if inf(L1^p) > sup(L2^q) then
6:         q ← q + 1
7:     else
8:         compute (L1^p ∩ L2^q) using IntersectSmall
9:         Δ ← Δ ∪ (L1^p ∩ L2^q)
10:        if sup(L1^p) < sup(L2^q) then p ← p + 1 else q ← q + 1
11: Δ is the result of L1 ∩ L2
```

Fonte: Ding e König (2011)

Como no Merge Linear, são utilizados dois ponteiros, *p* e *q,* que apontam para as partições dos conjuntos L_1 e L_2 respectivamente. Inicialmente os ponteiros apontam para a partição do seu respectivo conjunto que apresenta o menor elemento. Caso a partição que *p* aponte apresente pelo menos um intervalo em comum com a partição que *q* aponte, chamamos o algoritmo IntersectSmall (que é explicado abaixo). Para calcular o tamanho da interseção destas duas partições, depois que a interseção das duas partições é calculada, o ponteiro que referencia a partição com menor elemento é incrementado. Caso a partição que *p* aponte não tenha nenhum intervalo em comum com a partição que *q* aponte, o ponteiro que referencia a partição com o menor elemento é incrementado. Quando *p* ou *q* percorre totalmente sua respectiva lista de partições, o algoritmo é encerrado. Como inicialmente os conjuntos foram ordenados, nenhum elemento presente nas partições que não foram percorridas pertencerá à interseção.

No Algoritmo 1 pode ser visto que as variáveis n_1 e n_2 representam o número de partições nos conjuntos L_1 e L_2, respectivamente. Na linha 2 está a condição de parada do algoritmo, que acontece quando $p > n_1$ ou $q > n_2$, ou seja, quando alguma das duas listas de partições foi totalmente percorrida. Na linha 3 é verificado se o menor elemento da partição L_2^q é maior que o maior elemento da partição L_1^p. Caso seja verdadeiro, a variável *p* é incrementada e o algoritmo continua, pois como inicialmente os conjuntos foram ordenados de maneira crescente isso garante que não existe nenhum elemento das partições L_1^p e L_2^q que pertençam à interseção. Na linha 5 é verificado o caso contrário (se o menor elemento da

partição L_1^p é maior que o maior elemento da partição L_2^q). Neste caso, a variável *q* é incrementada e o algoritmo continua. A linha 7 é executada quando as condições das linhas 3 e 5 não forem satisfeitas, ou seja, existe pelo menos um elemento com o mesmo intervalo nas partições, e então é computado a interseção das partições L_1^p e L_2^q utilizando o algoritmo IntersectSmall. Na linha 9 a variável delta armazena o valor dos elementos que fazem parte da interseção, e na linha 10 é verificado qual partição apresenta o menor elemento, e incrementa a variável *p* ou *q*.

Algoritmo 2 – Algoritmo IntersectSmall

IntersectSmall(L_1^p, L_2^q): computing $L_1^p \cap L_2^q$
1: Compute $H \leftarrow h(L_1^p) \cap h(L_2^q)$
2: **for** each $y \in H$ **do**
3: $\quad \Gamma \rightarrow \Gamma \cup (h^{-1}(y, L_1^p) \cap h^{-1}(y, L_2^q))$
4: Γ is the result of $L_1^p \cap L_2^q$

Fonte: Ding e König (2011)

O objetivo do algoritmo IntersectSmall, apresentado no Algoritmo 2, é calcular o tamanho da interseção das duas partições recebidas como parâmetro. Inicialmente calculamos em O(1) a interseção das representações binárias destas partições, ou seja, $h(L_1^p) \cap h(L_2^q)$, e o resultado é armazenado em H. Se o i-nésimo bit de H tem valor 1, concluímos que existem elementos nas duas partições cujo o valor de *hash* é i. Note que todos os elementos da interseção das partições estão associados a um bit 1 em H. Em seguida, para cada posição y com valor 1 em H (linha 2), calculamos a interseção dos mapeamentos inversos nesta posição utilizando o merge linear, ou seja, $h^{-1}(y,L_1^p) \cap h^{-1}(y,L_2^q)$. Como todos os elementos nessa interseção pertencem à interseção das partições, armazenamos estes elementos em Γ (variável que armazena a interseção).

Outra variação do DK é o algoritmo utilizando partições randomizadas que, diferente do o algoritmo de interseção utilizando partições fixas, utiliza uma função *g* para particionar cada conjunto em pequenos grupos (não necessariamente com o mesmo tamanho). O algoritmo utiliza os *t* bits mais significativos de *g*(x) para determinar a partição de cada elemento, onde *g*(x) é uma função de *hash*. O *processamento online* é bastante similar ao descrito anteriormente e é apresentado no algoritmo 3.

Algoritmo 3 – Algoritmo de partições randomizadas.

1: **for** each $z_2 \in \{0,1\}^{t_2}$ **do**
2: Let $z_1 \in \{0,1\}^{t_1}$ be the t_1-prefix of z_2
3: Compute $L_1^{z_1} \cap L_2^{z_2}$ using IntersectSmall$(L_1^{z_1}, L_2^{z_2})$
4: Let $\Delta \leftarrow \Delta \cup (L_1^{z_1} \cap L_2^{z_2})$
5: Δ is the result of $L_1 \cap L_2$

Fonte: Ding e König (2011)

T1 e T2 representam a quantidade de bits significativos de cada grupo e Z1 e Z2 armazenam uma sequência de bits que representam cada grupo. Para exemplificar, supondo que T1 apresente o valor igual a 2 e que os bits de Z1 e Z2 sejam respectivamente 1101 e 1110. É feita a verificação se os dois bits mais significativos de Z1 são prefixo de Z2, que no caso do exemplo é verdade, já que os dois bits mais significativos de Z1 são 11 que por sua vez é prefixo de Z2. Isso quer dizer que podem existir elementos em comum nos grupos Z1 e Z2, ou seja, elementos que fazem parte da interseção.

Como veremos na seção a seguir, estimar um limite superior para o tamanho da interseção pode ser feito de forma mais eficiente que obter o tamanho exato da interseção. Portanto, algumas aplicações podem ser aceleradas utilizando limites superiores ao invés dos tamanhos exatos, como é o caso do Top-k descrito na seção 2.4

2.3 Algoritmos para cálculo de limite superior para o tamanho da interseção de conjuntos

Segundo Demana (2009, p. 72) "Uma função *f* é limitada superiormente se existe algum *B* que seja maior ou igual a todo número da imagem de *f*. Qualquer que seja o número *B* esse é chamado de *limite superior* de *f*."

Existem algoritmos que podem determinar um limite superior para o tamanho da interseção entre conjuntos. O valor retornado por esses algoritmos nem sempre é o valor exato do tamanho da interseção, mas como visto na definição de limite superior, o valor retornado sempre será um valor maior ou igual ao tamanho exato da interseção. Alguns desses algoritmos são:

Filtro de Bloom é uma estrutura de dados proposta por Bloom (1970) utilizada para determinar se um elemento pertence a um conjunto, e pode ser utilizada para determinar um limite superior para o tamanho da interseção de conjuntos devido ao fato da abordagem apresentar somente falsos positivos, ou seja, um valor retornado pelo algoritmo nunca será menor do que o tamanho exato do tamanho da interseção.

Filtro Conjuntivo é uma estrutura de dados proposta por Hokanohara e Yoshida (2010) que pode ser utilizada para determinar um limite superior da interseção entre conjuntos. Entretanto, o filtro conjuntivo tem seu foco voltado para eficiência de espaço, e por isso, seu cálculo não é tão rápido quando comparado com os demais.

Filtro de Cardinalidade é uma estrutura de dados proposta por Takuma e Yanagisawa (2013) e é utilizada para calcular um limite superior para o tamanho da interseção entre dois conjuntos quaisquer. A estrutura Filtro de Cardinalidade apresenta duas variações, sendo elas o Filtro de Cardinalidade Simples e o Filtro de Cardinalidade Recursivo, descritos abaixo.

Definição:

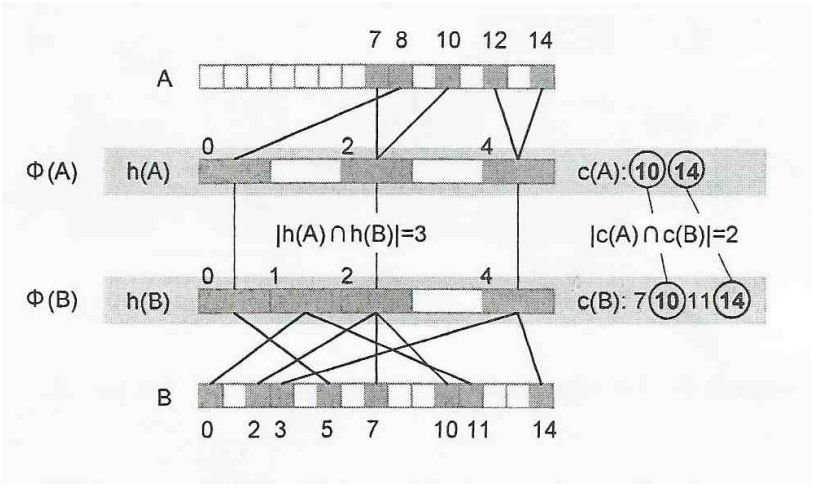
$X = \{E_1, E_2, E_3 ... E_n\}$

$A \subseteq X$

$B \subseteq X$

h é uma função *Hash* universal.

Figura 2 – Filtro de cardinalidade simples.



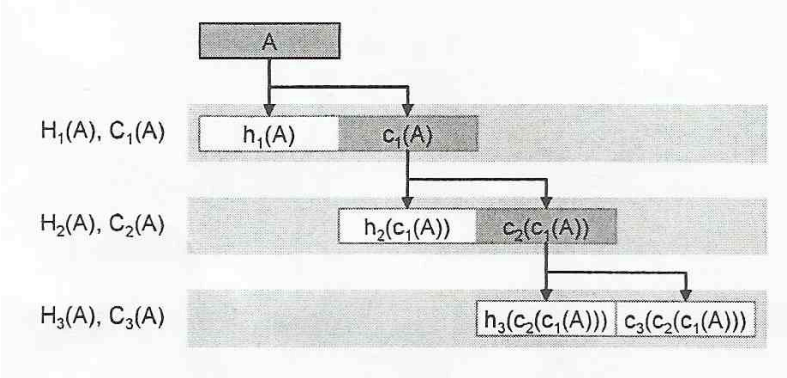
Fonte: Takuma e Yanagisawa (2013)

O Filtro de Cardinalidade Simples consiste em um par de conjuntos (h(A),c(A)), onde h(A) contém os valores do *hash* dos elementos de A, e c(A) que armazena os elementos que apresentam o mesmo valor da função *hash* que outros elementos (colisão), exceto o primeiro elemento que provocou colisão. Por exemplo, supondo que o conjunto A={3,6,9} e o valor de *hash* dos elementos 3, 6 e 9 seja respectivamente 1, 2 e 1, consequentemente h(A) = {1,2} e c(A)={9}.

O mesmo processo é executado no conjunto B, e o valor para o limite superior do tamanho da interseção entre esses dois conjuntos pode ser obtido a partir de $|h(A) \cap h(B)| + |c(A) \cap c(B)|$, sendo a operação $|h(A) \cap h(B)|$ calculada rapidamente utilizando a operações AND bit a bit, e a operação $|c(A) \cap c(B)|$ sendo calculado a partir de um algoritmo de merge linear (note que o tamanho de $|c(A) \cap c(B)|$ é tipicamente muito menor que $|A \cap B|$).

O algoritmo garante o limite superior do tamanho da interseção pelo fato de que o menor elemento de A com uma função *hash* igual a função *hash* menor elemento de B será contado por $|h(A) \cap h(B)|$, e os elementos cuja função *hash* apresentou o mesmo valor que a função *hash* de outros elementos do seu conjunto (colisão) serão contados por $|c(A) \cap c(B)|$. O valor do tamanho da interseção não é exato, devido ao fato de elementos diferentes poderem apresentar o mesmo valor da função *hash*, e serem contados por $|h(A) \cap h(B)|$, como por exemplo os elementos 8 do conjunto A e 5 do conjunto B visto na Figura 2.

Figura 3 – Filtro de cardinalidade recursivo.



Fonte: Takuma e Yanagisawa (2013)

O Filtro de Cardinalidade Recursivo, como o próprio nome sugere, é um algoritmo recursivo que apresenta um parâmetro *l* que representa a quantidade de níveis que o algoritmo deve ser executado. A execução do algoritmo se resume nos seguintes passos: é construído no pré-processamento h(A) e o h(B) juntamente com suas listas de colisões c(A) e

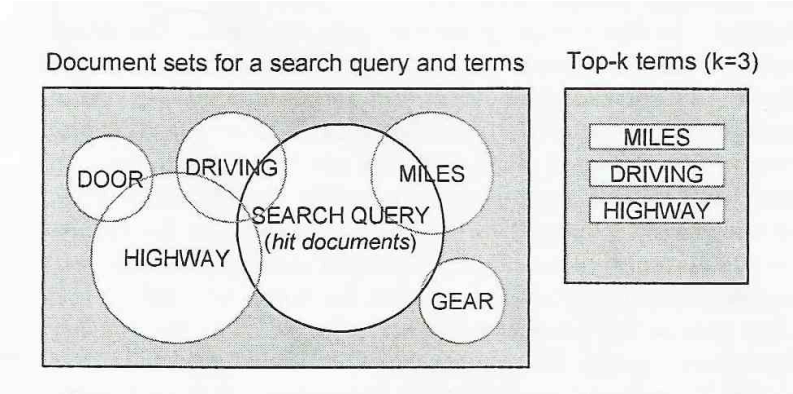
c(B), então as listas de colisões dos conjuntos passam a ser utilizados como um novo conjunto, onde A = c(A) e B = c(B), e uma nova lista de colisões é gerada. Isso é repetido *l* vezes, sendo gerado para todos o *l* níveis h(a) e h(b), juntamente com suas respectivas listas de colisões. O algoritmo então inicia o *processamento online* que, para cada nível *l,* computa $|h(A) \cap h(B)| + |c(A) \cap c(B)|$ da última iteração recursiva. Chegando no último nível, o algoritmo obtém o limite superior do tamanho da interseção. O Filtro de Cardinalidade Recursivo reduz o gasto com o algoritmo de merge linear pelo fato de que a cada iteração do algoritmo o conjunto de colisões é reduzido, e consequentemente aumenta o número de operações AND bit a bit. Porém, as operações AND bit-a-bit são computadas bem mais rápidas que o algoritmo de merge linear (w comparações em O(1)), e por isso o Filtro de Cardinalidade Recursivo apresenta uma leve vantagem sobre o Filtro de Cardinalidade Simples.

Existem outros algoritmos eficientes para determinar uma aproximação para o tamanho da interseção entre conjuntos, como os de Beyer *et al.* (2009), Simitsis *et al*. (2008), Krauthgamer *et al.* (2008) e Papapetrou, Siberski e Nejdl (2010) que não atendem à definição de limite superior pois, como esses algoritmos são baseados em amostragem, os valores obtidos podem ser menores que o tamanho exato da interseção, e por isso essas abordagens não serão consideradas nesse trabalho.

2.4 Top-k

A partir da explicação na Seção 2.1 sobre o uso da interseção entre conjuntos para determinar as informações relacionadas com uma consulta a um motor de busca, torna-se mais fácil compreender o problema Top-k. O Top-k é um algoritmo que retorna os k termos que apresentam a maior interseção com os documentos retornados por uma consulta (TAKUMA; YANAGISAWA, 2013). Um detalhe importante é que esses k termos não obrigatoriamente devem estar relacionados com os termos consultados. O Top-k é um problema que pode ser otimizado a partir do conhecimento do tamanho da interseção (ou um limite superior).

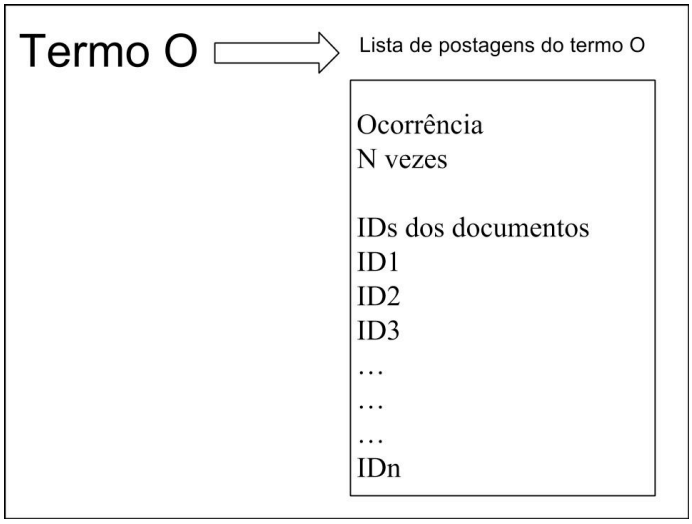
Figura 4 – Problema Top-k.



Fonte: Takuma e Yanagisawa (2013)

O algoritmo se caracteriza da seguinte maneira: é assumido que cada termo está associado a uma lista de *posts* que contém a quantidade de ocorrência desse termo e os identificadores dos documentos em que esse termo está presente, conforme ilustrado na Figura 5.

Figura 5 – Lista de postagens.



Fonte: Elaboração própria.

Algoritmo 4 – Algoritmo Top-k.

Input:
P: posting lists sorted in descending order of list size
$|P|$: the number of posting lists
$P[i]$: the posting list for term i $(i = 0, 1, \ldots, |P| - 1)$
S: the documents retrieved by a search query
k: the maximum number of terms to retrieve
1: $Q \leftarrow \emptyset$ // a set of the temporary top-k terms
2: **for** $i = 0, 1, \ldots, |P| - 1$ **do**
3: **if** $|Q| = k$ and $|P[i]| \leq |S \cap P[\min(Q)]|$ **then**
4: break // early out
5: **end if**
6: **if** $|Q| < k$ **then**
7: insert $(i, |S \cap P[i]|)$ into Q
8: **else if** $|S \cap P[i]| > |S \cap P[\min(Q)]|$ **then**
9: insert $(i, |S \cap P[i]|)$ into Q
10: remove the pair $(\min(Q), |S \cap P[\min(Q)]|)$ from Q
11: **end if**
12: **end for**
13: **return** Q

Fonte: Takuma e Yanagisawa (2013)

Um algoritmo para a tarefa Top-k é apresentado no Algoritmo 4. A estrutura P é um vetor de listas de postagens. Os elementos em P estão ordenados em ordem decrescente de tamanho de lista de postagem, ou seja, P[0] é a lista de postagem com mais elementos. A estrutura S é o conjunto dos documentos retornados pela máquina de busca. A estrutura Q é uma heap que armazena os termos escolhidos para o Top-k, cuja chave de um termo é tamanhos da interseção de S com sua lista de *posts*. Neste caso, P[min(Q)] representa a lista de postagem com menor interseção com S, dentre as listas de postagem dos termos em Q. Se Q tem menos de k termos, o próximo termo é inserido em Q (linhas 6 e 7). Caso contrário, verificamos se a lista de postagem do próximo termo i tem interseção com S maior do que a menor interseção das listas dos termos em Q (ou seja, P[min(Q)]). Se isto ocorrer, substituímos i por min(Q) em Q (linhas 8, 9 e 10). O algoritmo pode terminar assim que duas condições sejam satisfeitas (linha 3), sendo elas: a heap Q está cheia, e a próxima lista de postagem é menor que a menor chave em Q (ou seja, nenhuma outra substituição ocorrerá na heap Q, pois todas as listas de postagem restantes são menores que a menor interseção encontrada).

A aceleração do Top-k ocorre na linha 8, onde é calculado um limite superior de |S ∩P[i]|. Caso o limite superior para |S ∩ P[i]| seja menor que |S ∩ P[MIN(Q)]|, poupamos a execução das linhas 9 e 10 e o cálculo exato de |S ∩ P[i]|. Como o cálculo do limite superior é rápido, esta operação pode poupar o cálculo do valor exato da interseção em muitas iterações.

3 DESENVOLVIMENTO

Todos os experimentos foram executados em um PC com sistema operacional OS X 10.10 64 Bits, processador 2.9 GHz Intel Core i7 e memória 8 GB 1600 MHz DDR3. O desenvolvimento do trabalho foi dividido em três etapas, descritas nas seções 3.1, 3.2 e 3.3.

3.1 Implementação e avaliação do DK1H e SCF

Neste trabalho chamamos de DK1H o algoritmo proposto por Ding e König (2011), e chamamos SCF o algoritmo proposto por Takuma e Yanagisawa(2013). Denotamos por GAP a relação (L – E) / E, onde L é limite superior para o tamanho da interseção, e E é o valor exato do tamanho da interseção.

No trabalho de Takuma e Yanagisawa (2013) é apresentado um experimento utilizando uma base de dados sintética criada pelos autores. As subseções a seguir descrevem os procedimentos que realizamos para criação dessa base de dados sintética, e a implementação e execução dos algoritmos DK1H e SCF.

3.1.1 Geração do conjunto universo

Segundo a descrição do trabalho de Takuma e Yanagisawa (2013) o conjunto universo utilizado foi formado por 10MB de elementos. Adotamos como elementos números inteiros sem sinal. Foi criado um algoritmo na linguagem de programação C++ que aleatoriamente escolhe um número entre zero e o maior número inteiro sem sinal representado em 32 bits, e em seguida insere este número no conjunto universo (caso ainda não tenha sido adicionado). Ao final da execução o conjunto universo apresentou um total de 2.621.440 elementos, totalizando 10MB. Todos os outros conjuntos gerados no experimento são subconjuntos deste conjunto universo.

3.1.2 Geração de uma base de dados sintética e implementação dos algoritmos DK1H e SCF

O trabalho de Takuma e Yanagisawa (2013) apresenta seis especificações para a criação de instâncias de conjuntos, que pode ser visto na Figura 6.

Figura 6 – Especificações para instâncias de conjuntos sintéticos.

Name	$\|A\|$	$\|B\|$	Cr
(A) Large / no correlation	1M	1M	1.0
(B) Middle / no correlation	100K	100K	1.0
(C) Small / no correlation	10K	10K	1.0
(D) Asymmetric / no correlation	1M	10K	1.0
(E) Middle / positive correlation	100K	100K	10.0
(F) Middle / negative correlation	100K	100K	0.1

Fonte: Takuma e Yanagisawa (2013)

Cada instância é formada por dois conjuntos com seus respectivos tamanhos em disco, e um fator de correlação Cr que é um dado de entrada previamente definido utilizado para determinar o tamanho da interseção entre os conjuntos da instância. O tamanho da interseção entre os conjuntos de uma instância é definido pela Equação 1, onde A e B são os conjuntos da instância, X é o conjunto universo e Cr é o fator de correlação.

$$| A \cap B | = \left\lceil \frac{Cr \bullet | A | \bullet | B |}{| X |} \right\rceil \quad (1)$$

Mil instâncias foram criadas para cada uma das seis especificações de instância, com o auxílio de um algoritmo escrito na linguagem de programação C++. O algoritmo para a criação dessas instâncias calcula a quantidade de elementos dos conjuntos A e B com base no tamanho em disco fornecido na especificação da instância, e então calcula o tamanho da interseção utilizando a Equação 1. Com o tamanho da interseção definida, o algoritmo escolhe aleatoriamente um elemento do conjunto universo e adiciona nos conjuntos A e B, caso o elemento escolhido ainda não tenha sido inserido em nenhum dos conjuntos. Quando a quantidade de elementos de cada conjunto é igual ao tamanho da interseção, o algoritmo passa a escolher aleatoriamente dois elementos distintos do conjunto universo que ainda não tenham sido adicionados nos conjuntos, e então insere um elemento em cada conjunto. Portanto, no final da execução do algoritmo são gerados dois conjuntos seguindo exatamente as especificações de tamanho em disco e fator de correlação.

Os algoritmos DK1H e SCF descritos nas seções 2.2 e 2.3, respectivamente, foram implementados na linguagem de programação C++, seguindo as instruções dos trabalhos de Takuma e Yanagisawa (2013) e Ding e König (2011).

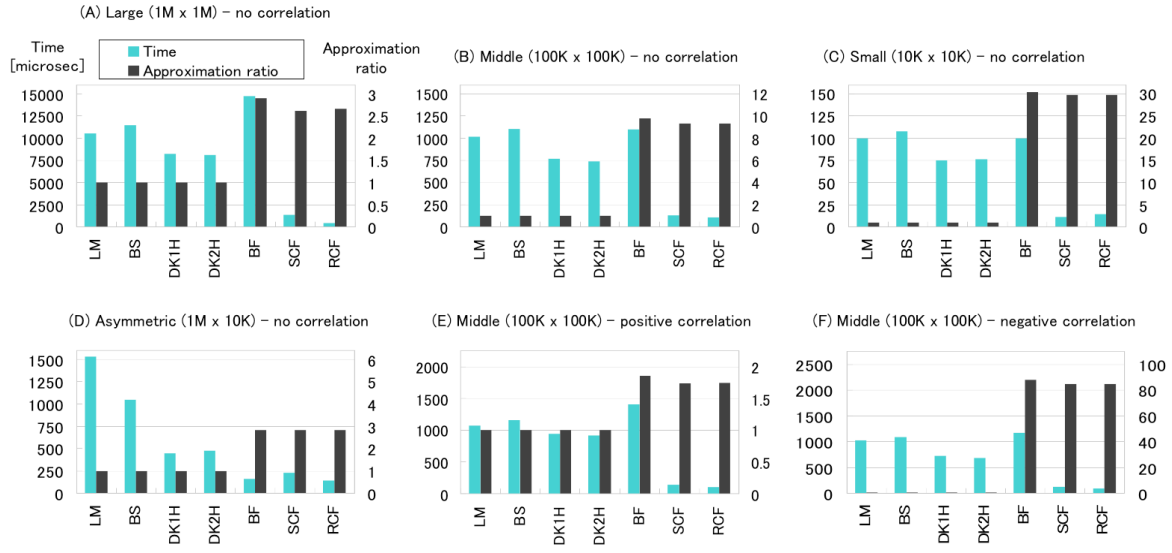
3.1.3 Execução dos algoritmos DK1H e SCF utilizando a base sintética

Para cada execução do algoritmo DK1H foi armazenado o tamanho da interseção e o tempo de execução do *processamento online*. Com esses dados foi calculado o tempo médio de execução para cada especificação de instância. Para cada execução do algoritmo SCF foi armazenado o limite superior do tamanho da interseção e o tempo de execução. Com esses dados, e com tamanho exato da interseção de cada instância fornecido pelo algoritmo DK1H, foram calculados o tempo médio de execução do *processamento online* e o GAP médio.

3.1.4 Comparação dos resultados obtidos

Takuma e Yanagisawa (2013) ilustram o tempo médio de execução e o *approximation ratio* (GAP+1) de alguns algoritmos, para cada especificação de instância, que pode ser visto na Figura 7. Entre esses algoritmos estão o DK1H e o SCF.

Figura 7 – Desempenho de algoritmos com base de dados sintéticas



Fonte: Takuma e Yanagisawa (2013)

Comparando os resultados apresentados na figura com os resultados obtidos por nossa implementação do SCF, verificamos que os resultados foram similares. O algoritmo DK1H apresentou GAP igual a zero em todas as instâncias, como esperado. Entretanto, o tempo médio de execução em 83.3% dos casos foi superior ao tempo reportado por Takuma e Yanagisawa (2013).

3.2 Execução do Top-k utilizando base de dados reais

Takuma e Yanagisawa(2013) compararam o tempo de execução do Top-k utilizando o DK1H e o SCF. Nas subseções a seguir é descrita nossa reprodução desse experimento, com a diferença que no trabalho de Takuma e Yanagisawa (2013) foram utilizados como dados reais as informações de tráfego de rodovias, e neste trabalho utilizamos páginas de sites de universidades brasileiras.

3.2.1 Obtenção de dados reais

Inicialmente extraímos uma lista contendo o endereço eletrônico de 804 universidades brasileiras do site vestibulandoweb (http://www.vestibulandoweb.com.br/sites-de-universidades.asp) acessado em 05 outubro 2014. Com essa lista, e com o auxílio da ferramenta wget, baixou-se todo conteúdo HTML dos sites dessas universidades. Entretanto, nem todos os endereços eletrônicos dessa lista estavam disponíveis, totalizando a extração de dados de 383 sites de universidades brasileiras.

3.2.2 Tratamento dos dados

No total foram obtidas 47.766 páginas web, onde cada página corresponde a um documento no nosso experimento. Um programa implementado na linguagem de programação ruby extraiu os termos de cada documento gerando os *posts*. Cada *post* é composto por um identificador único que é um termo, e tem como conteúdo os identificadores dos documentos onde esse termo tem ocorrência. Todos os termos extraídos foram forçados a passar para letra minúscula para que os algoritmos que utilizem esses dados não sejam sensíveis à caixa. Por exemplo, os termos “HOJE” e “hoje”, provenientes de páginas distintas após a tratamentos de dados, são salvos no mesmo *post*.

Depois que todos os posts foram gerados, um programa implementado na linguagem de programação ruby cria um documento chamado *post list*, que contém todos os identificadores dos termos, ordenados de maneira decrescente de acordo com a quantidade de identificadores de documentos presente em cada *post*. A extração e tratamento dos dados gerou 47.766 documentos (páginas), que por sua vez gerou 131.074 *posts*.

3.2.3 Geração de consultas de teste

Uma consulta a um motor de busca retorna os identificadores dos documentos onde todos os termos de uma busca estão presentes. Ou seja, se um documento não contém pelo menos um dos termo da consulta, seu identificador não estará presente no retorno da busca.

Para gerar as consultas de teste, foi utilizado um programa implementado na linguagem de programação ruby que escolhe três termos aleatoriamente entre os mil *posts* com mais documentos, e então calcula a interseção entre os identificadores de documentos desses três termos. A Tabela 1 informa os termos utilizados e o tamanho da interseção (número de documentos que contém todos os termos da consulta).

Tabela 1 – Lista de consultas de teste

Identificador	Termos	Tamanho da interseção
1	brasil, instituto, campus	10067
2	tecnologia, pesquisa, federal	12774
3	cursos, ensino, desenvolvimento	9684
4	professor, projetos, alunos	2573
5	biblioteca, universidade, trabalho	1336
6	informa, administra, mais	7152
7	profissional, concurso, documentos	671
8	institucional, acesso, social	3524
9	reitoria, publica, estudantes	1292
10	docente, vestibular, processo	765

Fonte: Elaboração própria

3.2.4 Implementação e execução do algoritmo Top-k com as variações DK1H e SCF

O algoritmo Top-k descrito na Seção 2.4 foi implementado seguindo as informações descritas no trabalho de Takuma e Yanagisawa (2013). Consideramos duas variações, sendo uma delas utilizando somente o algoritmo DK1H e a outra utilizando os algoritmos DK1H e SCF.

Para cada variação do algoritmo Top-k, e para cada consulta de teste descrita na seção anterior, o algoritmo foi executado com o limite de 50 termos. Ou seja, para cada execução foi armazenado o tempo gasto para determinar os 50 termos com maior interseção.

3.3 Execução o algoritmo Top-k com a função que simula a execução de um algoritmo fictício

Definimos um algoritmo fictício cujo tempo de execução cai exponencialmente com o aumento do GAP, e apresenta o mesmo tempo de execução do SCF quanto atinge o mesmo GAP do SCF. Com este algoritmo fictício podemos realizar uma análise de sensibilidade do tempo de execução do algoritmo Top-k. As subseções a seguir descrevem como foi parametrizada a função de tempo do algoritmo fictício, sua aplicação no algoritmo Top-k, e a criação e comparação dos resultados.

3.3.1 Parametrização do algoritmo fictício

Os dados de execução do algoritmo fictício são baseados no tempo de execução e qualidade do limite superior, e esses valores são obtidos a partir de uma função que utiliza os dados dos algoritmos DK1H e SCF. Para que esses dados fossem definidos, foi calculado para cada *post* e consulta de teste o tempo de execução e o valor do tamanho da interseção fornecido pelo algoritmo DK1H, além do tempo de execução e o limite superior do tamanho da interseção fornecido pelo algoritmo SCF.

Com os dados de execução dos algoritmos DK1H e SCF para cada consulta de teste e *post,* definiu-se uma relação exponencial entre o GAP e o tempo de execução, apresentada na Equação 2, onde T_DK é o tempo de execução do algoritmo DK1H, T_SCF é o tempo de execução do algoritmo SCF, e G_SCF é o GAP do algoritmo SCF.

$$tempo(GAP) = T\_DK * exp(-a * GAP) \quad (2)$$

$$a = - \ln(T\_SCF / T\_DK) / G\_SCF \quad (3)$$

Com a função para determinar o tempo de execução do algoritmo a partir de um GAP qualquer, definiu-se então duas variações de GAP para o experimento, G1 e G2, que utilizam o GAP fornecido pelo algoritmo SCF definido como G_SCF nas equações 4 e 5.

$$G1 = G\_SCF / 2 \quad (4)$$

$$G2 = 2 * G\_SCF \quad (5)$$

Para cada post e consulta de teste foram definidos e armazenados os valores de tempo de execução e limite superior para as variações G1 e G2, para serem utilizados no

algoritmo Top-k. Portanto, avaliamos o Top-k considerando um algoritmo fictício cujo GAP vale a metade ou o dobro do GAP do algoritmo SCF, e o tempo de execução é definido pela Equação 2. Note que o GAP de DK1H é nulo (algoritmo exato), e o tempo gasto pelo algoritmo fictício é igual ao tempo gasto pelo algoritmo SCF quando o algoritmo fictício tem GAP igual ao GAP do SCF. Ou seja, tempo(G_SCF) = T_SCF.

3.3.2 Avaliação do Top-k usando algoritmo fictício para obter o limite superior da interseção

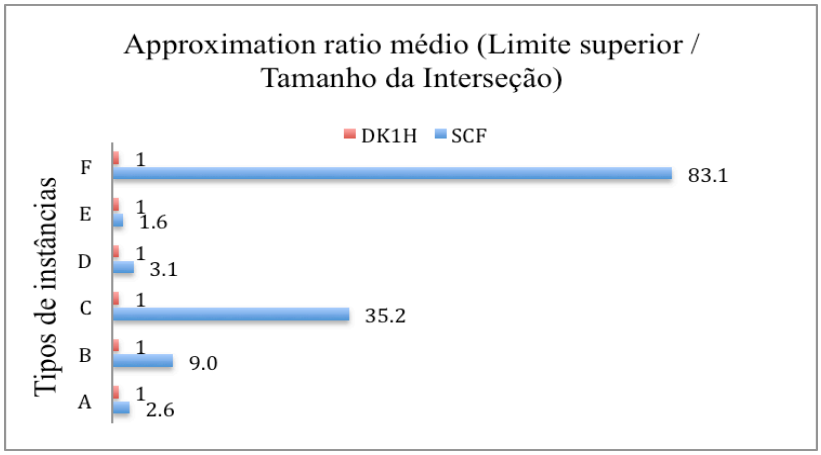
Similar à Seção 3.2.4, o algoritmo Top-k foi executado substituindo o algoritmo SCF pelos dados de execução do algoritmo fictício. O valor de limite superior do tamanho da interseção fornecido pelo algoritmo SCF foi substituído pelo valor do GAP definido previamente (G1 ou G2), e o algoritmo Top-k foi colocado em espera por tempo(G1) ou tempo(G2) milissegundos. Para todas as consultas, o algoritmo Top-k foi executado com o mesmo limite de 50 termos para as variações de GAP G1 e G2.

4 RESULTADOS EXPERIMENTAIS

Durante o início dos procedimentos, na etapa de análise dos algoritmos DK1H e SCF com dados sintéticos, foi possível constatar a notória vantagem em relação ao tempo de execução do algoritmo SCF sobre o algoritmo DK1H.

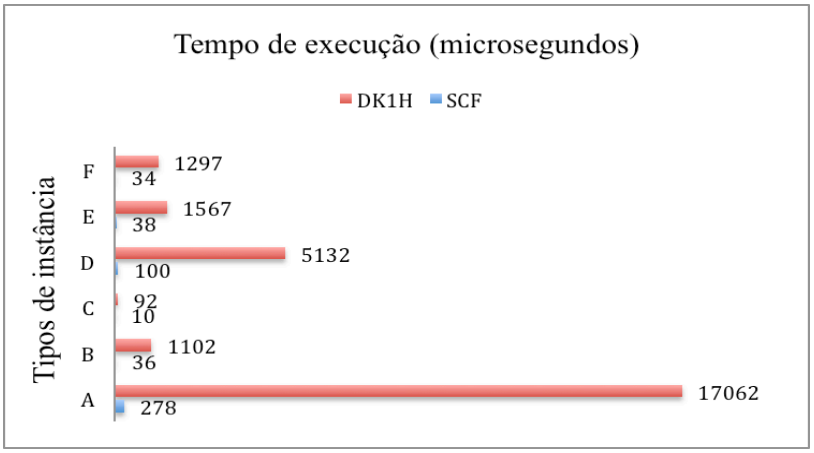
Um dos pontos de dificuldade foi tentar reproduzir o tempo de execução do algoritmo DK1H com o que estava descrito no trabalho de Takuma e Yanagisawa (2013), pois em todas as instâncias nossa implementação do algoritmo DK1H apresentou tempo de execução acima do descrito em Takuma e Yanagisawa (2013). Os dados médios de tempo de execução e *approximation ratio* para cada tipo de instâncias podem ser vistos nas figuras 8 e 9.

Figura 8 – Comparativo de *approximation ratio* médio entre os algoritmos DK1H e SCF.



Fonte: Elaboração própria

Figura 9 – Comparativo de tempo de execução entre os algoritmos DK1H e SCF.

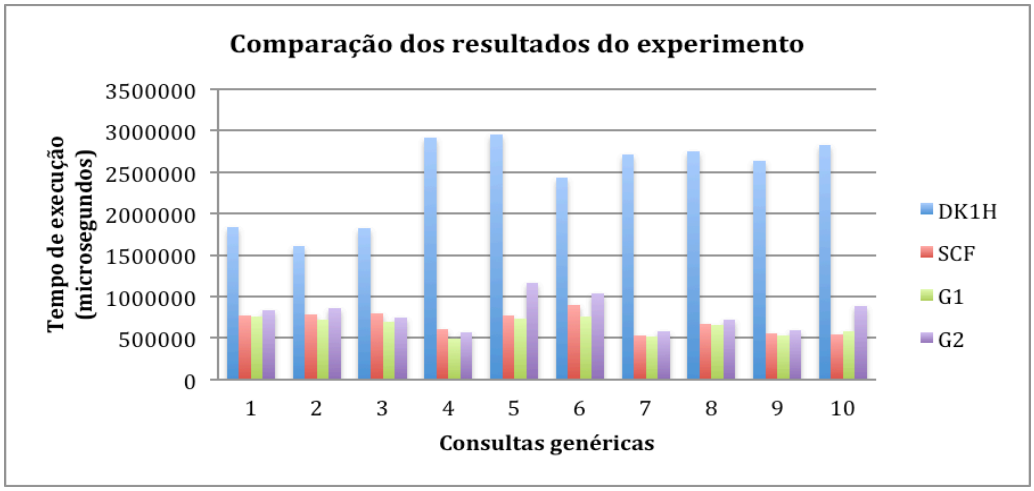


Fonte: Elaboração própria

Nossa implementação do algoritmo SCF apresentou um desempenho satisfatório, em alguns casos superando resultados descritos no trabalho de Takuma e Yanagisawa (2013) (reproduzidos na Figura 7).

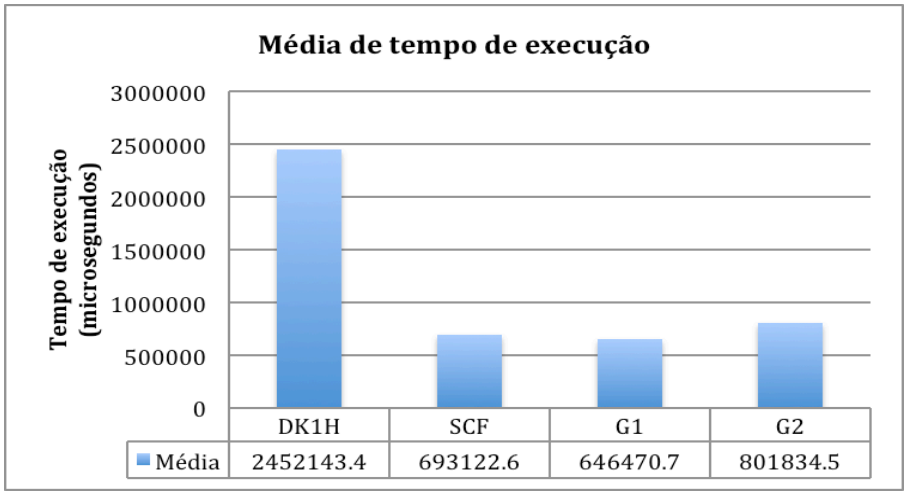
Na etapa de utilização do algoritmo Top-k com os algoritmos DK1H, SCF e com algoritmos fictícios G1 e G2, o algoritmo Top-k com algoritmo fictício G1 apresentou vantagem em 90% dos testes realizados. As informações na Figura 10 mostram um comparativo entre a execução do algoritmo Top-k com suas variações, para cada consulta. Na Figura 11 é apresentada uma média de tempo de execução do algoritmo Top-k entre todas as consultas sintéticas, para cada variação do Top-k.

Figura 10 – Comparativo entre as execuções do algoritmo Top-k.



Fonte: Elaboração própria

Figura 11 – Média de tempo entre as execuções do algoritmo Top-k



Fonte: Elaboração própria

O algoritmo Top-k com o GAP G1 apresentou o melhor tempo de execução (93% do tempo médio de execução do SCF). Por outro lado, o GAP G2 gerou um aumento de 15% no tempo médio quando comparado com o tempo médio do SCF.

5 CONSIDERAÇÕES FINAIS

Constatamos nos experimentos a boa qualidade do limite superior fornecido pelo SCF, visto que em algumas consultas o algoritmo SCF acertou precisamente o tamanho exato da interseção para todos os *posts*. Essa informação fez com que o número de dados para o experimento fosse aumentado em treze vezes para garantir um GAP significativo para o SCF.

Uma das abordagens que não foi apresentada neste trabalho, e que pode ser tema de trabalhos futuros, é a criação de funções que simulem a execução de algoritmos fictícios para cada um tipo de instância. Outra abordagem é propor e avaliar algoritmos para melhorar o tempo de execução do Top-k.

O principal objetivo do trabalho foi alcançado, que foi avaliar se é possível reduzir o tempo de execução do algoritmo Top-k, assumindo uma relação exponencial entre GAP e tempo de execução. Apesar de G1 ter apresentado uma leve vantagem sobre o algoritmo SCF, não é possível afirmar que essa vantagem possa deixar uma margem significativa para a melhoria do algoritmo Top-k.

REFERÊNCIAS

BARBEY, J.; LÓPEZ-ORTIZ, A.; LU, T. Faster Adaptive Set Intersections for Text Searching. **WEA**. Menorca, p. 146-157, maio. 2006. Disponível em:<http://link.springer.com/chapter/10.1007/11764298_13>. Acesso em: 15 set. 2013.

BEYER, K et al. Distinct-value synopses for multiset operations. **Communications of ACM**. v. 52, p. 87-95, out. 2009. Disponível em:< http://dl.acm.org/citation.cfm?id=1562787>. Acesso em: 10 out. 2013.

BILLE, P; Pagh, A.; Pagh, R. Fast Evaluation of Union-Intersection Expressions. **ISAAC 2007**. Sendai, p. 739-750, dec. 2007. Disponível em:<http://link.springer.com/chapter/10.1007/978-3-540-77120-3_64>. Acesso em: 7 set. 2013.

BLELLOCH, G. E.; REID-MILLER, M. Fast Set Operations Using Treaps. **SPAA**. Barcelona p. 16-26, jun. 1998. Disponível em:<http://dl.acm.org/citation.cfm?id=277660 >. Acesso em: 10 set. 2013.

BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. **Communications of ACM**. v. 13, p. 422-426, jul. 1970. Disponível em:<http://dl.acm.org/citation.cfm?id=362692>. Acesso em: 7 set. 2013.

BROWN, M. R.; TARJAN R. E. A fast merging algorithm. **Journal of the ACM**. v. 26, p. 211-226, abr. 1979. Disponível em:<http://dl.acm.org/citation.cfm?id=362692>. Acesso em: 10 set. 2013.

CORMEN, Thomas H. **Algoritmos**: teoria e prática. Rio de Janeiro: Elsevier, 2002.

DEMANA, Franklin D. **Pré-cálculo**. São Paulo: Addison-Wesley, 2009.

DING, B.; KÖNIG, C. Fast set intersection in memory. **VLDB Endowment**. v. 4, p. 255-266, jan 2011. Disponível em:<http://dl.acm.org/citation.cfm?id=1938550>. Acesso em: 6 set. 2013.

GERSTING, Judith L. **Fundamentos matemáticos para a ciência da computação**: um tratamento moderno de matemática discreta . 5. ed. Rio de Janeiro: Livros Técnicos e Científicos, c2004.

HWANG, F. K.; LIN, S. A simple algorithm for merging two disjoint linearly ordered sets. **SIAM Journal on Computing**. v. 1, p. 31-39, 1972 . Disponível em:<http://epubs.siam.org/doi/pdf/10.1137/0201004>. Acesso em: 17 set. 2013.

KRAUTHGAMER, R. et al. Greedy List Intersection.**ICDE 2008**. Cancun, p. 1033-1042, abr. 2008. Disponível em:<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4497512&tag=1>. Acesso em: 5 out. 2013.

MOFFAT, A.; WITTEN I. H.; BELL T. C. **Managing gigabytes**: compressing and indexing documents and images. [S.l.]: Morgan Kaufmann, 1999. Disponível em:<

http://books.google.com.br/books?hl=pt-BR&lr=&id=2F74jyPl48EC&oi=fnd&pg=PR23&dq=MOFFAT,+A.+et+al.+Managing+Gigabytes:+Compressing+and+Indexing+Documents+and+Images.+Morgan+Kaufmann,+1999.&ots=5QhQFp8Q4d&sig=OT8kYWdd5BFm5vUd3bjqCdgvz4I#v=onepage&q&f=false>. Acesso em: 5 out. 2013.

OKANOHARA, D.; YOSHIDA, Y. Conjunctive filter: Breaking the entropy barrier. **ALENEX10**. Texas, p. 77-83, jan. 2010. Disponível em:<http://siam.org/proceedings/alenex/2010/alx10_008_okanoharad.pdf>. Acesso em: 18 set. 2013.

PAPAPETROU O.;SIBERSKI W.;NEJDL W. Cardinality estimation and dynamic length adaptation for Bloom filters. **Distributed and Parallel Databases**. v. 28 p. 119-156, dec. 2010. Disponível em:< http://link.springer.com/article/10.1007/s10619-010-7067-2>. Acesso em: 7 out. 2013.

ROSEN, Kenneth H. **Matemática discreta e suas aplicações**. 6. ed. São Paulo, SP: McGraw-Hill, 2009.

SIMITIS, A. et al. Multidimensional Content eXploration. **VLDB Endowment**. v. 1, p. 660-671, ago. 2008. Disponível em:<http://siam.org/proceedings/alenex/2010/alx10_008_okanoharad.pdf>. Acesso em: 7 set. 2013.

TAKUMA, D.; YANAGISAWA, H. Faster upper bounding of intersection size. **SIGIR '13**. Dublin, p. 703-712, jul. 2013. Disponível em:< http://dl.acm.org/citation.cfm?id=2484065>. Acesso em: 4 set. 2013.

TATIKONDA, S. et al. On efficient posting list intersection with multicore processors. **SIGIR '09**. Bonston, p. 738-739, jul. 2009. Disponível em:< http://dl.acm.org/citation.cfm?id=1572104>. Acesso em: 25 set. 2013.

TSIROGIANNIS, D; GUHA S.; KOUDAS N. Improving the performance of list intersection. **VLDB Endowment**. v. 2, p. 838-849, ago. 2009. Disponível em:< http://dl.acm.org/citation.cfm?id=1687722>. Acesso em: 19 set. 2013.

VESTIBULANDOWEB. Sites de Universidades. Disponível em:< http://www.vestibulandoweb.com.br/sites-de-universidades.asp>. Acesso em: 05 out. 2014.

APÊNDICES

APÊNDICE A – Lista de sites de universidades que tiveram conteúdo extraído

abaetetuba.ifpa.edu.br	www.facens.br	www.ifsp.edu.br
aedsfachusc.com.br	www.facimed.edu.br	www.ifsul.edu.br
aracaju.ifs.edu.br	www.facoparana.com.br	www.ifsuldeminas.edu.br
araguaia.ufmt.br	www.facos.edu.br	www.iftm.edu.br
barreiros.ifpe.edu.br	www.facsal.br	www.impacta.edu.br
belem.ifpa.edu.br	www.faculdadebatista.com.br	www.imperatriz.ufma.br
belojardim.ifpe.edu.br	www.faculdadedevinhedo.com.br	www.ims.ufba.br
braganca.ifpa.edu.br	www.faculdadeguaianas.com.br	www.inapos.edu.br
breves.ifpa.edu.br	www.faculdademodulo.com.br	www.inesul.edu.br
cantareira.br	www.faculdadepitagoras.com.br	www.inhumas.ifg.edu.br
castanhal.ifpa.edu.br	www.faculdadeunida.com.br	www.inspirar.com.br
cefet-rj.br	www.faculdadevascodagama.edu.br	www.inta.edu.br
ces.ufcg.edu.br	www.fadire.edu.br	www.iprj.uerj.br
conceicaodoaraguaia.ifpa.edu.br	www.fae.br	www.itumbiara.ifgoias.edu.br
curitibanos.ufsc.br	www.fae.edu	www.jatai.ifgoias.edu.br
faahf.edu.br	www.faeca.com.br	www.jatai.ufg.br
facisaba.com.br	www.fafic.com.br	www.jc.iffarroupilha.edu.br
facol.com	www.fafism.com.br	www.jf.ifsudestemg.edu.br
faculdade.cnecbento.com.br	www.faflor.com.br	www.jna.ifmt.edu.br
faculdadefamesp.com.br	www.fai.com.br	www.joinville.ifsc.edu.br
faculdades.sp.senai.br	www.fainor.com.br	www.lusiada.br
faculdadesaofrancisco.com.br	www.faj.br	www.machadosobrinho.com.br
fafibe.br	www.fam2011.com.br	www.mackenzie.br
famariana.edu.br	www.famac.com.br	www.marilia.unesp.br
fametro.com.br	www.famam.com.br	www.materdei.edu.br
fan.edu.br	www.famath.com.br	www.mch.ifsuldeminas.edu.br
fatecaracatuba.edu.br	www.fanan.esab.edu.br	www.medcariri.ufc.br
favip.edu.br	www.fap.com.br	www.mg.senac.br

fis.edu.br	www.fapacleopoldina.com.br	www.mundomackenzie.com.br
formiga.ifmg.edu.br	www.fapb.edu.br	www.muriae.ifsudestemg.edu.br
ifgoiano.edu.br	www.fapce.com.br	www.muz.ifsuldeminas.edu.br
ipojuca.ifpe.edu.br	www.fasete.edu.br	www.nepomuceno.cefetmg.br
itabaiana.ufs.br	www.fat.uerj.br	www.newtonpaiva.br
itaituba.ifpa.edu.br	www.fatecba.com.br	www.niltonlins.br
joinville.ufsc.br	www.fatecbarueri.edu.br	www.novafapi.com.br
lagarto.ufs.br	www.fatecbpaulista.edu.br	www.novomilenio.br
laranjeiras.ufs.br	www.fateccarapicuiba.edu.br	www.osorio.ifrs.edu.br
maracanau.ifce.edu.br	www.fateccatanduva.edu.br	www.paragominas.ufra.edu.br
metalurgia.sp.senai.br	www.fatecfranca.edu.br	www.parauapebas.ufra.edu.br
pagead2.googlesyndication.com	www.fatecgarca.edu.br	www.pb.iffarroupilha.edu.br
palmas.ifto.edu.br	www.fatecitapetininga.edu.br	www.poa.ifrs.edu.br
paraiso.ifto.edu.br	www.fatecitaqua.edu.br	www.policamp.edu.br
pelotas.ifsul.edu.br	www.fatecitu.edu.br	www.portal.fmu.br
pesqueira.ifpe.edu.br	www.fatecjahu.edu.br	www.portal.ifba.edu.br
portal.anhembi.br	www.fatecjd.edu.br	www.portal.ufpa.br
portal.cefet-rj.br	www.fatecmaua.com.br	www.portal.ufra.edu.br
portal.faculdadedacidade.edu.br	www.fatecmm.edu.br	www.portalmouralacerda.com.br
portal.faesa.br	www.fatecmococa.com.br	www.porto-ifto.edu.br
portal.sje.ifmg.edu.br	www.fatecosasco.edu.br	www.portoseguro.ifba.edu.br
portal.ufam.edu.br	www.fatecpg.com.br	www.puc-rio.br
portal.ufes.br	www.fatecpp.edu.br	www.puccamp.br
portal.unifeob.edu.br	www.fatecpr.edu.br	www.pucminas.br
portal.urisantiago.br	www.fatecriopreto.edu.br	www.pucpr.br
portal2.ifrn.edu.br	www.fatecsantoandre.com.br	www.pucrs.br
porteiras.unipampa.edu.br	www.fatecsorocaba.edu.br	www.quixada.ufc.br
recife.ifpe.edu.br	www.fatecsp.br	www.rc.unesp.br
s7.addthis.com	www.favag.edu.br	www.redentor.edu.br
scristovao.ifs.edu.br	www.favip.edu.br	www.riopomba.ifsudestemg.edu.br
sesp.edu.br	www.fcc.edu.br	www.sa.iffarroupilha.edu.br
sites.txt	www.fcjp.edu.br	www.san.uri.br
spo.ifsp.edu.br	www.fcmscsp.edu.br	www.santanna.br
uniamerica.br	www.febf.uerj.br	www.santarita.br

unibave.net	www.feg.unesp.br	www.saoboaventura.edu.br
universidades.txt	www.feliz.ifrs.edu.br	www.scelisul.com.br
www.aeso.br	www.fema.edu.br	www.serradocarmo.edu.br
www.aisi.edu.br	www.fesjf.estacio.br	www.sertao.ifrs.edu.br
www.al.iffarroupilha.edu.br	www.feso.br	www.simoesfilho.ifba.edu.br
www.anapolis.ifg.edu.br	www.fespmg.edu.br	www.site.uft.edu.br
www.brazcubas.br	www.fespsp.org.br	www.sjdr.ifsudestemg.edu.br
www.camacari.ifba.edu.br	www.fev.edu.br	www.slmandic.edu.br
www.campuscastanhal.ufpa.br	www.ffassis.edu.br	www.sorocaba.unesp.br
www.campusdosertao.ufal.br	www.ffp.uerj.br	www.sp.senac.br
www.campussobral.ufc.br	www.fgv.br	www.sp.senai.br
www.camtuc.ufpa.br	www.fiap.com.br	www.sr.ifes.edu.br
www.canoas.ifrs.edu.br	www.fic.br	www.sr.iffarroupilha.edu.br
www.car.uem.br	www.finan.com.br	www.sumare.edu.br
www.cariri.ufc.br	www.fipecafi.org	www.suprema.edu.br
www.cas.ifmt.edu.br	www.fiponline.com.br	www.svc.ifmt.edu.br
www.catolicadeanapolis.com.br	www.fit.br	www.svs.iffarroupilha.edu.br
www.catolicasc.org.br	www.fiu.com.br	www.timoteo.cefetmg.br
www.caxias.ifrs.edu.br	www.fiu.com.br:82	www.toledo.br
www.cba.ifmt.edu.br	www.fizo.edu.br	www.uast.ufrpe.br
www.ccjs.ufcg.edu.br	www.fjav.com.br	www.ubafupac.com.br
www.ccp.uenp.edu.br	www.flated.edu.br	www.uc.br
www.cefetmg.br	www.fmc.br	www.ucdb.br
www.cefetsp.br	www.fmj.br	www.ucp.br
www.centropaulasouza.sp.gov.br	www.foa.org.br	www.ucpel.tche.br
www.cesumar.br	www.fog.br	www.ucs.br
www.cetai.com.br	www.formosa.ifg.edu.br	www.ucsal.br
www.ceul.ufms.br	www.frb.edu.br	www.udesc.br
www.ceunes.ufes.br	www.fsa.br	www.ueap.ap.gov.br
www.ceunsp.br	www.fsd.edu.br	www.uece.br
www.cfp.ufcg.edu.br	www.ftec.com.br	www.uefs.br
www.cidadesp.edu.br	www.fumec.br	www.ueg.br
www.cienciasmedicas.com.br	www.fundacaojau.edu.br	www.uel.br
www.cj.uenp.edu.br	www.funecsantafe.edu.br	www.uem.br
www.claretiano.edu.br	www.funjob.edu.br	www.uemg.br
www.clm.uenp.edu.br	www.fupacnovalima.com.br	www.uems.br
www.cnp.ifmt.edu.br	www.furb.br	www.uenf.br
www.codo.ufma.br	www.furg.br	www.uepa.br
www.conquista.ifba.edu.br	www.fw.uri.br	www.uepb.rpp.br

www.cotemig.com.br	www.goiania.ifg.edu.br	www.uepg.br
www.crg.uem.br	www.granbery.edu.br	www.uergs.edu.br
www.crp.ufv.br	www.ibhes.edu.br	www.uerj.br
www.cstr.ufcg.edu.br	www.ibilce.unesp.br	www.uern.br
www.damasio.edu.br	www.ibiruba.ifrs.edu.br	www.uerr.edu.br
www.div.cefetmg.br	www.ibmec.br	www.uesb.br
www.eafa-to.gov.br	www.ideau.com.br	www.uesb.br:8080
www.eafcol.gov.br	www.ifac.edu.br	www.uesc.br
www.eafcrato.gov.br	www.ifam.edu.br	www.uespi.br
www.eafs.gov.br	www.ifap.edu.br	www.ufabc.edu.br
www.ebah.com.br	www.ifbaiano.edu.br	www.ufac.br
www.eca.cefetmg.br	www.ifc.edu.br	www.ufal.br
www.enem.vestibulandoweb.com.br	www.ifce.edu.br	www.ufal.edu.br
www.ensitec.com.br	www.ifes.edu.br	www.ufba.br
www.erechim.ifrs.edu.br	www.iff.edu.br	www.ufc.br
www.esamc.br	www.ifg.edu.br	www.ufcg.edu.br
www.esefap.edu.br	www.ifgoiano.edu.br	www.ufcspa.edu.br
www.eunapolis.ifba.edu.br	www.ifma.edu.br	www.uff.br
www.faa.edu.br	www.ifmg.edu.br	www.uffs.edu.br
www.faacz.com.br	www.ifms.edu.br	www.ufg.br
www.faag.com.br	www.ifnmg.edu.br	www.ufjf.br
www.faama.edu.br	www.ifpb.edu.br	www.ufla.br
www.faap.br	www.ifpi.edu.br	
www.facamp.com.br	www.ifrj.edu.br	
www.facape.br	www.ifro.edu.br	
www.facc.com.br	www.ifrr.edu.br	
www.faccar.com.br	www.ifs.ifsuldeminas.edu.br	
www.faccat.br	www.ifsc.edu.br	
www.facdesco.edu.br	www.ifsertao-pe.edu.br	

Fonte: Elaboração própria