Heurísticas para Aprimorar o Método BMW e suas Variantes

Lídia Lizziane Serejo Carvalho[1], **Edleno Silva de Moura**[1],
Caio Moura Daoud [1], **Altigran Soares da Silva** [1]

[1]Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)
Manaus – Am – Brasil

lidializz@gmail.com, {edleno, caiodaoud, alti}@icomp.ufam.edu.br

Abstract. *In this paper, we propose and evaluate heuristics to improve the performance of BMW and its variants. The proposed changes retain ownership to preserve the order of the first results at the end of processing, offering benefits in an attempt to both further reduces query processing times and the amount of memory required for processing queries.*

Resumo. *Neste artigo são propostas e experimentadas modificações nas heurísticas de descarte de documentos utilizadas para processar consultas no algoritmo BMW e suas variantes. As modificações propostas mantém a propriedade de preservar a ordem dos primeiros resultados obtidos ao final do processamento, oferecendo como benefícios a redução no tempo de processamento de consultas e na quantidade de memória utilizada durante o processamento.*

1. Introdução

Processadores de consulta em sistemas de busca são implementados visando minimizar custos em relação ao tempo de processamento e uso de memória. Seu principal objetivo é tomar uma consulta especificada pelo usuário e fornecer um ranking com os documentos considerados relevantes para a consulta. Para isso, utilizam modelos de Recuperação da Informação que produzem funções de *ranking* como o modelo vetorial [Salton et al. 1975] e o modelo BM25 [Robertson and Walker 1994], que computam um peso para cada documento dada uma consulta. Em geral os sistemas retornam um *ranking* com os top-k documentos com maior peso, onde k é um número predefinido que corresponde a quantidade de documentos que serão retornados.

Ding e Suel (2011) propuseram o algoritmo Block-Max WAND (BMW), um algoritmo para processamento de consultas que mostrou-se mais veloz que os propostos anteriormente na literatura. Variantes do método BMW têm sido propostas e estudadas na literatura, incluindo o BMW-CS [Rossi et al. 2013], o qual reduz o tempo de processamento de consultas quando comparado ao BMW, mas tem como desvantagem um aumento significativo na quantidade de memória necessária para processar consultas e a possibilidade de perder resultados ao final do processamento, alterando assim a lista de documentos que aparecem no topo da resposta dada a cada consulta.

Algoritmos como o BMW utilizam técnicas de descarte de documentos (técnicas de poda) que estão diretamente relacionadas às atuais heurísticas que visam a redução de custos de processamento. O algoritmo mantém uma estrutura de *heap* de mínimo que armazena os top-k documentos durante o processamento. O menor peso entre todos os

documentos presentes no *heap* é usado como um limiar de descarte. Assim durante o processamento só são computados os escores dos documentos que têm chance de superar o limiar de descarte.

Este trabalho tem como principal objetivo estudar e propor melhorias ao algoritmo BMW e seu variante BMW-CS com a finalidade de reduzir o tempo de processamento e quantidade de memória utilizada ao processar consultas. Como principais resultados, obteve-se as seguintes contribuições: (i) Redução no tempo de processamento, com uma heurística de adoção de limiar de poda inicial que garante a integridade dos resultados. (ii) Redução no uso de memória decorrente de redução significativa no tamanho de uma estrutura auxiliar empregada pelos algoritmos no processamento de consultas, as *skiplists*, que são utilizadas para acelerar o processamento de consultas. Nos dois casos, as mudanças realizadas apresentam a vantagem de não alterar o resultado inicial provido pelos métodos (BMW ou BMW-CS). Como o BMW é um algoritmo que preserva resultados do topo da resposta para cada consulta, nossas alterações sobre o BMW também possuem tal propriedade.

O presente trabalho está estruturado como segue. A Seção 2 expõe um resumo dos principais trabalhos relacionados presentes na literatura. Nas Seções 3 e 4, explicamos as estratégias propostas para a solução do problema apresentado. A Seção 5 descreve os experimentos realizados e os resultados obtidos. E por fim, na Seção 6, discutimos as conclusões e direcionamentos para trabalhos futuros.

2. Trabalhos Relacionados

Diversos pesquisadores têm proposto estratégias para melhorar o desempenho do processamento de consultas em sistemas de busca. Os métodos de processamento de consultas encontrados na literatura podem ser divididos em duas classes principais : os que utilizam estratégia TAAT (*Term-At-A-Time*) e os que utilizam estratégia DAAT (*Document-At-A-Time*) [Baeza-Yates et al. 2011]. Nos métodos que utilizam a estratégia TAAT, as listas invertidas são ordenadas pelo peso dos documentos. Com isso, a consulta é processada um termo por vez. Sua maior desvantagem é que esta estratégia requer muita memória para armazenar os escores alcançados por cada documento em cada termo da consulta. Nos métodos que utilizam estratégia DAAT, as listas invertidas são ordenadas pelo *id* do documento permitindo que as listas sejam percorridas em paralelo e que o escore completo de um documento seja computado em cada iteração. Outra característica desta estratégia é o uso de *skiplists* associadas a cada lista invertida. Estas estruturas armazenam ponteiros para entradas da lista invertida dividindo-a em blocos para facilitar o acesso aos documentos contendo tipicamente entre 128 e 256 entradas [Broder et al. 2003].

Um dos trabalhos mais importantes apresentados nos últimos anos propõe um método de processamento de consultas DAAT conhecido como WAND (*Weak* AND ou *Weighted* AND) [Broder et al. 2003]. Neste método, durante a indexação da coleção de documentos é armazenado o maior escore de cada lista invertida. Durante o processamento, saber o escore máximo que um documento pode atingir é importante para evitar o custo de computar escores de documentos que não têm chance de alterar o topo do *ranking* de respostas. Durante o processamento de uma consulta, documentos que estão sendo avaliados, chamados de pivôs, são analisados em duas etapas. Primeiro, obtém-se o *Max Score*, escore máximo de cada lista onde o pivô tem chance de ocorrer. Quando

o *Max Score* for maior que o limiar de poda atual, a lista invertida de cada termo será acessada para que o documento pivô tenha seu escore real computado. Quando o *Max Score* não supera o limiar de poda, o documento pivô é descartado e um novo pivô é selecionado.

O algoritmo BMW [Ding and Suel 2011] foi criado tendo como base o WAND [Broder et al. 2003], porém propõe uma solução otimizada que abrange principalmente uma modificação nas *skiplists*. No BMW, para cada entrada das *skiplists*, é armazenado o maior escore do bloco. A estratégia de poda é similar à adotada no método WAND, com a vantagem de possuir um escore máximo mais próximo do escore real de cada documento. Com essa vantagem, o método BMW descarta ainda mais documentos que não apresentam peso suficiente para serem inseridos no topo do *ranking* de respostas. A poda no BMW acontece em dois momentos, (i) quando o *Max Score* não supera o limiar de descarte, como no método WAND, e (ii) quando o *Block Max Score* não supera o limiar de descarte. O *Block Max Score* é computado com base no escore máximo dos blocos em que o documento pode ocorrer e só é verificado quando o *Max Score* supera o limiar de descarte.

O limiar de descarte é dinamicamente atualizado sempre que um documento tem escore maior que o do atual limiar. Este documento atualiza o *heap* de respostas e o limiar de poda é atualizado com o peso do menor documento do *heap*.

Para exemplificar, supondo um cenário como mostra a Figura 1, o limiar de poda indica que é necessário que um documento tenha um escore maior que $3,2$ para atualizar o atual *ranking* de respostas. A figura retrata um momento do processamento onde os documentos 20, 40 e 90 associados a cada termo da consulta "amarelo verde azul" serão avaliados, nesta ordem. A lista invertida do primeiro documento a ser processado tem um escore máximo de $2,5$, ou seja, o documento 20, que só ocorre na lista do termo amarelo, não tem peso suficiente para superar o limiar de poda. No entanto, o documento 40, que pode ocorrer na lista invertida do termo "amarelo", poderá ter um escore de até $4,0$ $(2,5+1,5 > 3,2)$, maior que o limiar de poda atual. Já que existe tal possibilidade, o algoritmo verifica o *Block Max Score* do pivô (documento 40) . Caso o pivô ocorra na lista invertida do termo "amarelo" e ele tenha o maior escore dentre os de seu bloco, nas duas listas, seu escore máximo será então $3,3$ $(2,3+1,0 > 3,2)$, portanto superior ao limiar de poda. Após tais verificações, o documento 40 tem seu escore completo avaliado. Neste exemplo, o escore completo do documento é 3.3 $(2,3+1,0)$, que ultrapassa novamente o limiar de poda, portanto o documento 40 entra no *heap* de respostas atualizando o limiar de poda. Nesta etapa, os ponteiros das listas invertidas são movidos para que apontem para o próximo documento maior que o pivô avaliado, e o algoritmo então prossegue repetindo o processo.

Alguns autores exploram a abordagem do particionamento da lista invertida em mais de uma camada para acelerar o processamento de consultas. Uma das mais recentes, e também mais bem sucedidas estratégias de particionamento é adotada pelo algoritmo BMW-CS [Rossi et al. 2013], o qual divide cada lista invertida em duas camadas, a primeira camada tem tamanho menor e é criada usando entradas que apresentam os maiores escores das listas invertidas, enquanto a segunda camada tem um índice muito maior e contém as entradas restantes. O processamento é dividido em duas fases. A primeira, chamada de seleção de candidatos, processa a primeira camada das listas dos termos da

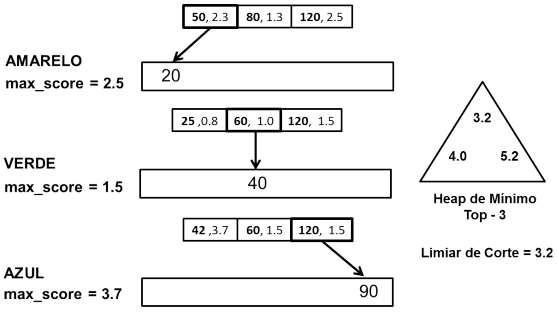


Figura 1. Listas invertidas para os termos da consulta "amarelo verde azul".

consulta. Nesta fase, o processo é similar ao utilizado no método BMW mas considera também o escore máximo do documento na segunda camada. O valor para a segunda camada é importante, pois podem existir documentos que estão na primeira camada mas também ocorrem na segunda camada para outros termos da consulta. Os documentos selecionados na primeira fase são chamados de candidatos à resposta. Na segunda fase do processo, a segunda camada é percorrida apenas para buscar ocorrências dos documentos candidatos na segunda camada e computar o escore real desses documentos.

O método BMW-CS apresenta a desvantagem de não preservar todas as top-k respostas de uma consulta, pois documentos que não ocorrem na primeira camada de pelo menos um dos termos da consulta não são avaliados. Nestes casos, mesmo que os documentos tenham escores suficientes para entrar na resposta, eles serão descartados pelo algoritmo BMW-CS.

3. Estudo de Limiares Iniciais para Poda

O limiar de poda é determinante para que as estratégias de descarte de documentos funcionem nos métodos apresentados. Em um primeiro momento do processamento, enquanto o *heap* de respostas não está cheio, o limiar de poda é igual a zero. Isso significa que a estratégias de poda não são aplicadas pelo BMW inicialmente. Assim que os primeiros k documentos são computados, também é obtido o primeiro limiar de poda diferente de zero, dando inicio ao processo de descarte de documentos que reduz o custo de processamento de consultas no algoritmo BMW. Na medida em que mais documentos são processados, o limiar de poda aumenta até atingir o valor mínimo do *ranking* final.

A presença de um limiar inicial baixo faz com que menos documentos sejam descartados no inicio do processamento de consultas do algoritmo BMW e seus variantes. Nossa primeira tentativa de melhorar os algoritmos foi estudar o desempenho de tais algoritmos ao utilizar como limiar de poda inicial uma estimativa de valor que seja ao mesmo tempo segura, sendo garantidamente menor que o limiar de poda obtido ao final do processamento, e grande o suficiente para ter impacto nos custos de processamento, tendo valor o mais próximo possível do limiar de poda final.

A estratégia para adoção de limiar inicial foi aplicada em dois cenários: quando estamos interessados em computar os top-10 resultados e os top-1000 para uma dada consulta. Os dois cenários são considerados típicos em sistemas de busca e são normalmente estudados em artigos que tratam do processamento eficiente de consultas [Rossi et al. 2013]. Para estimar o limiar inicial de poda, guardamos durante a

indexação o k-ésimo maior escore de cada palavra da coleção em sua entrada correspondente presente na estrutura de índice invertido. Dada uma consulta, estima-se um valor inicial de escore mínimo tomando-se o maior k-ésimo escore dentre os termos da consulta. Tal escolha garante que o limiar inicial não irá descartar respostas, pois sabe-se que há pelo menos k entradas com escore pelo menos igual ao limiar inicial escolhido. Por outro lado, a adoção de tal limiar permite que se descarte a priori entradas que seriam levadas em consideração quando adotado um limiar inicial igual a zero.

Além de propor e experimentar estratégias para determinar um limiar inicial, fez-se também um estudo para verificar quanto se pode ganhar em desempenho com esse tipo de estratégia. Para isso, foi armazenado o mínimo escore entre as k respostas obtidas no processamento de consultas, e depois as mesmas consultas foram novamente processadas, porém iniciando o limiar de poda com seu valor máximo. Denominamos este valor como limiar ótimo, por ser um indicativo do maior ganho que poderíamos obter com a utilização da estratégia de limiar de poda inicial. O limiar ótimo é apenas um valor de referência, pois para obtê-lo é necessário que a consulta seja primeiro processada integralmente.

4. Blocos de Tamanho Variáveis

Nos experimentos apresentados por [Ding and Suel 2011] e [Rossi et al. 2013] foram utilizados blocos fixos de 128 documentos na criação das *skiplists*. Isso quer dizer que a cada 128 documentos em uma lista invertida é criada uma entrada na *skiplist* para acelerar o acesso a documentos e guardar informações de descarte, como o maior escore de cada bloco (*Block Max Score*). Como vimos nas Figuras 3(a) e 3(b) e na Tabela 2, quanto menor o tamanho dos blocos, menor o tempo de processamento. Esse custo está diretamente relacionado à busca do documento nos blocos para computar o escore real.

Sempre que o *Max Score* e *Block Max Score* de um pivô superar o limiar de poda, os blocos correspondentes serão acessados em busca do documento pivô para que seu escore real seja computado. Blocos são totalmente descartados quando o *Block Max Score* do pivô não supera o limiar de poda. Com isso, percebe-se que blocos com *Block Max Score* baixo apresentam maior probabilidade de descarte. Neste caso, inferimos que blocos adjacentes que apresentam valor baixo de *Block Max Score* poderiam ser integrados para que mais documentos possam ser descartados durante uma iteração do processador de consultas.

Tabela 1. Desempenho do algoritmo BMW para blocos de tamanho fixo de 64, 128 e 256 documentos. A quantidade de blocos é apresentada com *.

		Pivôs	Tempo(ms)
Blocos de 64	Top-10	369.781.344	20
23.595.719*	Top-1000	1.002.528.576	51
Blocos de 128	Top-10	426.335.552	22
11.798.404*	Top-1000	1.116.841.472	55
Blocos de 256	Top-10	495.183.424	27
5.899.760*	Top-1000	1.223.783.040	59

Para determinar se o *Block Max Score* é baixo ou não, utilizou-se como referência o milésimo maior escore de cada termo da coleção, fazendo-se uma associação com a quantidade típica de entradas normalmente considerada nos resultados de busca. Logo,

blocos com *Block Max Score* menor que o valor de referência são considerados blocos de baixo impacto, e portanto podem se unir a outro bloco.

A Figura 2 ilustra uma lista invertida para um termo *A* que está dividida em n blocos, cada um composto por 128 documentos. No exemplo, o milésimo maior escore desta lista já foi computado e tem valor igual a 12. Neste cenário, os blocos A_2 e A_3 são combinados por apresentarem escore máximo menor que o valor de referência 12. Os blocos A_1, A_4 e An permanecem como estavam. O novo bloco $A_2 + A_3$ formado terá 256 documentos e *Block Max Score* igual a 8.

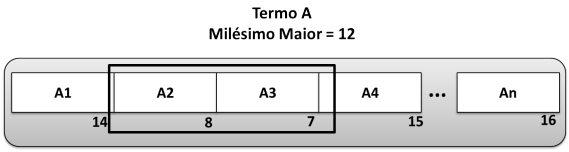


Figura 2. Exemplo de lista invertida com blocos variáveis.

Nossa hipótese é que a heurística de união de blocos descrita acima pode ser usada para aprimorar o processo de poda em sistemas de busca como o BMW e o BMW-CS. Na seção de experimentos, mostramos que nossa hipótese se comprova na prática e a heurística proposta, quando combinada com os benefícios de nossa proposta de estimar um limiar inicial de poda, resulta em melhora significativa nos indicadores de desempenho dos dois algoritmos.

5. Experimentos

Para os experimentos, utilizamos e modificamos o sistema de busca implementado por [Rossi et al. 2013]. O sistema adotado processa consultas utilizando o algoritmo BMW ou o variante BMW-CS. Utilizou-se nos experimentos a coleção de referência TREC GOV2. Esta é composta por $25.205.179$ milhões de páginas coletadas do domínio .gov no início de 2004. A TREC GOV2 possui 426 GB de texto, compostos por páginas HTML e texto extraído de páginas no formato PDF e postscript (PS). O índice inteiro possui aproximadamente 7 GB de listas invertidas e o vocabulário é composto por cerca de 4 milhões de termos distintos. A coleção TREC GOV2 foi adotada por ter se tornado um padrão para estudos que propõem melhorias de eficiência em sistemas de busca, tendo sido usada nos experimentos dos principais trabalhos encontrados na literatura.

Selecionou-se aleatoriamente um conjunto de 10.000 consultas extraídas da coleção TREC 2006 *efficiency queries*. Todos as *stopwords* dessas consultas foram retiradas. Durante o processamento de consultas, o índice inteiro é carregado para a memória. Todas as configurações foram escolhidas por serem similares às adotadas em trabalhos similares [Strohman and Croft 2007, Ding and Suel 2011], o que torna mais fácil a comparação entre os métodos estudados. Os experimentos foram executados em uma máquina Intel(R) Xeon(R), com processador X5680, 3.33GHz e 64GB de memória. Todos os experimentos foram executados em cenários que retornavam top-10 ou top-1000 respostas. A recuperação das top-1000 respostas foi incluída para simular um ambiente onde o conjunto top-1000 é utilizado como entrada para um método de *ranking* mais sofisticado, como por exemplo o uso de uma técnica de aprendizado de máquina. O cenário de top-10 respostas foi incluído para simularmos um cenário mais comum, onde o usuário

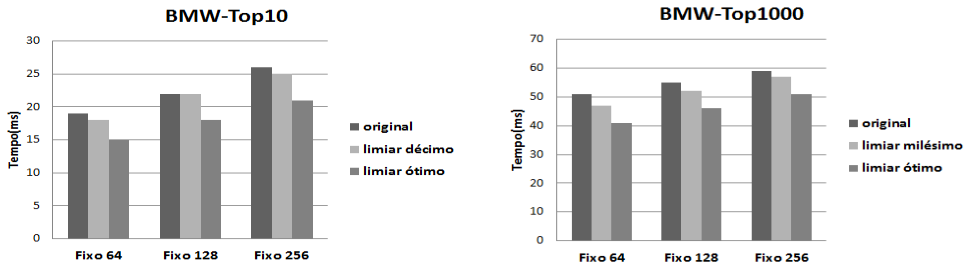
está interessado apenas em uma pequena lista de resultados para a sua consulta. Os algoritmos foram avaliados em termos de tempo de processamento e memória utilizada.

As estratégias foram implementadas no algoritmo BMW, proposto por [Ding and Suel 2011], e o algoritmo BMW-CS, proposto por [Rossi et al. 2013]. Estas foram desenvolvidas em C++ tendo como base trabalhos anteriores. Assim como os algoritmos originais, mantemos a utilização do modelo probabilístico Okapi BM25 como função de similaridade. Para o algoritmo BMW-CS, variamos o tamanho da primeira camada de 2 em 2% até 50% do índice completo nos experimentos.

5.1. Resultados Com Limiar de Poda Inicial

As Figuras 3(a) e 3(b) mostram ganhos nas estratégias propostas em relação ao algoritmo BMW sem limiar de poda inicial (BMW Original) para as top-10 e top-1000 consultas, respectivamente. As figuras apresentam os experimentos realizados também ao variarmos o tamanho dos blocos gerados nas listas invertidas. Foram experimentados tamanho fixo de 64, 128 e 256 documentos. Blocos fixos com 64 documentos resultam em tempos de processamento de consultas menores, porém como indicado na Tabela 2, formam mais blocos no índice invertido, requerendo mais memória para armazenar as *skiplists*. De maneira oposta, blocos fixos com 256 documentos formam menos blocos, no entanto resultam em tempo de processamento maior. Quanto maior a quantidade de blocos formados (número de entradas da estrutura *skiplist*), maior é o custo de memória usado no processamento de consultas, pois será necessário mais espaço para o armazenamento destes dados.

O tamanho de bloco utilizado tipicamente em trabalhos apresentados na literatura é de 128 entradas. Nesta configuração, observa-se que a escolha de um limiar inicial ótimo resultaria em ganho de cerca de 18% do tempo de processamento para as top-10 respostas e 16% para as top-1000 respostas, o que seria o limite para qualquer estratégia de estimativa de limiar inicial. Como podemos observar também na Figura 3, nossa heurística de estimativa de limiar teve impacto para top-1000 respostas, com uma redução aproximada de 5,5% em relação ao algoritmo original. Os maiores ganhos ocorreram com blocos de tamanho 64, onde temos reduções aproximadas de 5,2% e 7,8% para top-10 e top-1000 respostas, respectivamente.



(a) (b)

Figura 3. Desempenho do algoritmo BMW em diferentes abordagens para limiares de poda iniciais: BMW original, BMW com décimo/milésimo maior escore e BMW com o mínimo escore das top-k respostas para blocos fixos de 64, 128 e 256 documentos.

Tabela 2. Número total de entradas das *skiplists* para blocos fixos de 64, 128 e 256 documentos.

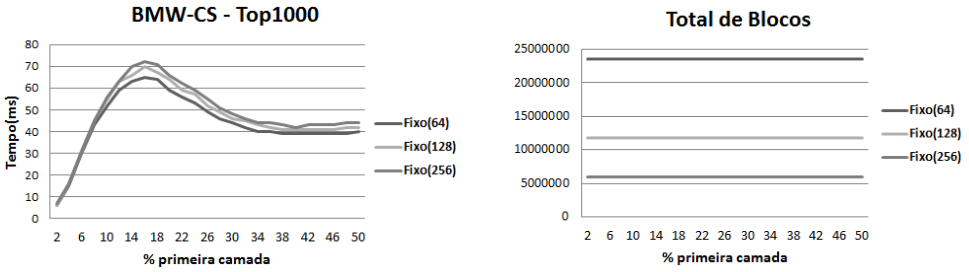
Fixo 64	Fixo 128	Fixo 256
23.595.719	11.798.404	5.899.760

No algoritmo BMW-CS, para blocos fixos de 64, 128 e 256 documentos, nas duas camadas, os maiores ganhos do método com estratégia de limiar inicial são obtidos para top-1000 respostas. Dentro deste cenário, para a análise de tempo de processamento, calculamos a média de tempo entre as variações de 18% e 50% da primeira camada para todas as opções de implementação, como mostra a Tabela 3. A tabela mostra que a introdução do milésimo como estratégia de limiar inicial reduz o tempo de processamento de consultas em até aproximadamente 17% .

Tabela 3. Média do tempo de processamento em milissegundos calculada a partir da variação de 18% da primeira camada para top-1000 respostas.

	Fixo 64	Fixo 128	Fixo 256
BMW-CS original	54,65	57,94	59,94
BMW-CS c/ limiar	45,12	47,82	49,88
Redução (%)	17,44	17,46	16,78

Temos nas Figuras 4(a) e 4(b) respectivamente, o comportamento da implementação da estratégia de limiar inicial e o total de blocos formados para os três cenários, quando variarmos o tamanho da primeira camada. Percebe-se que não há muita oscilação no total de blocos na medida em que aumentamos a porcentagem da primeira camada. Nota-se ainda que o uso de blocos fixos de 64 entradas gera custos de memória expressivos e dada sua semelhança em relação ao tempo de processamento com blocos fixos de 128 e 256 (Figura 4(a)), ele não se mostra como uma boa opção de implementação. Tal cenário, apresenta a média aproximada de 23,6 mil blocos, enquanto blocos de 128 e 256 documentos formam aproximadamente 12 mil e 6 mil blocos, respectivamente. Portanto, como verifica-se na Figura 4(a), visto que o tempo de processamento entre os cenários são bem similares, podemos inferir que a melhor implementação é aquela que apresenta o menor número de blocos criados, ou seja, o cenário de blocos fixos com 256 entradas.



(a) Processamento de consultas com limiar inicial

(b) Total de entradas nas skiplists

Figura 4. Desempenho e total de blocos fixos do algoritmo BMW-CS.

5.2. Variando o Tamanho Dos Blocos

Começamos com a avaliação do impacto da variação no tamanho dos blocos no BMW. Como primeira estratégia, estabelecemos um tamanho mínimo e máximo para variação do tamanho dos blocos. Variamos blocos com no mínimo 16, 32, 64, 128, 256 e 512 documentos, compondo ao máximo o total de 1024 documentos. A Figura 5 mostra um comparativo desta estratégia com o algoritmo BMW Original. Percebe-se o aumento no tempo de processamento para top-10 e top-1000 respostas. Para top-1000 respostas retornadas, ao aplicarmos a variação com blocos de 512 até 1024 documentos, o tempo aumenta de 52 milissegundos para um pouco mais de 60 milissegundos. Nota-se ainda na figura que, apesar do tempo de processamento aumentar, a utilização da estratégia diminui em até aproximadamente 80% o número total de blocos formados (Figura 5(c)) quando comparamos com o algoritmo BMW Original.

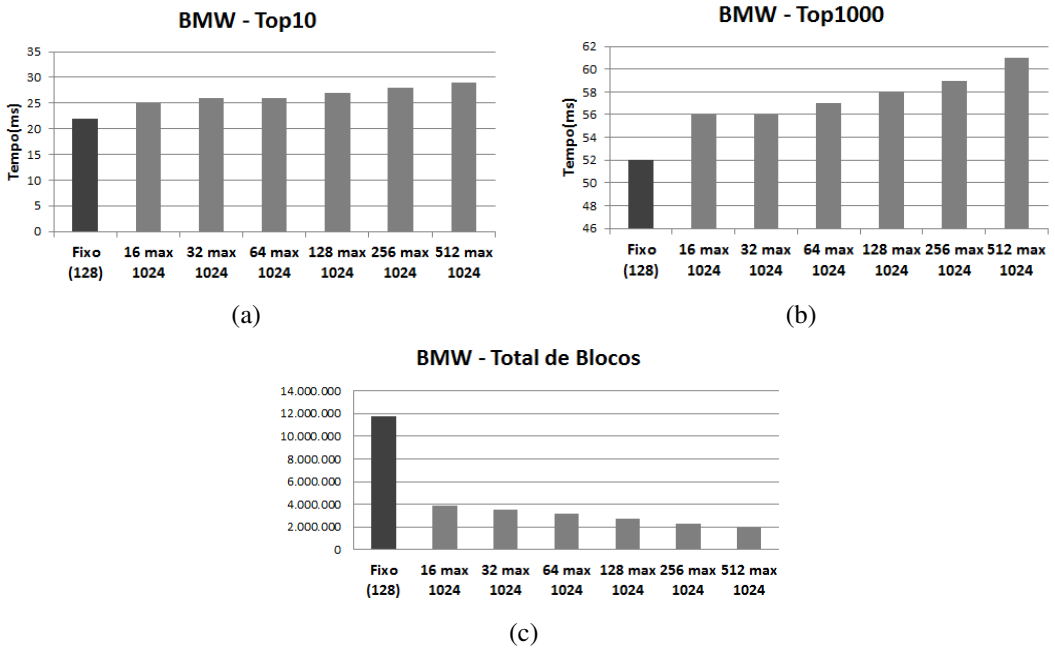


Figura 5. Desempenho do algoritmo BMW com a utilização das abordagem de variação do tamanho de blocos (máximo 1024 documentos).

Outra estratégia adotada com intuito de reduzir o tempo de processamento com blocos variados foi reduzir o tamanho máximo de 1024 para 128 ou 256 documentos. O gráfico apresentado na Figura 6 mostra os tempos alcançados com essas novas variações. Há redução no tempo de processamento para índices formados por blocos com no máximo 128 documentos para top-10 e top-1000 respostas. Ainda na figura, percebe-se que podemos manter o tempo de processamento similar ao BMW Original quando se utiliza o tamanho máximo de 256 documentos por bloco. Nota-se na Figura 6(c) que com esta opção de implementação reduzimos em até aproximadamente 44% o total de blocos formados. Por exemplo, gerar blocos de tamanho mínimo 32 e máximo 256 documentos, resultou em tempo de processamento igual ao obtido pelo BMW original (52 milisegundos). Contudo, o número de blocos gerados por tal opção foi 35% menor, gerando 7.676.292 blocos contra 11.798.404 do BMW original, o que significa um ganho significativo em termos de redução da memória extra gerada para armazenar informação sobre cada bloco.

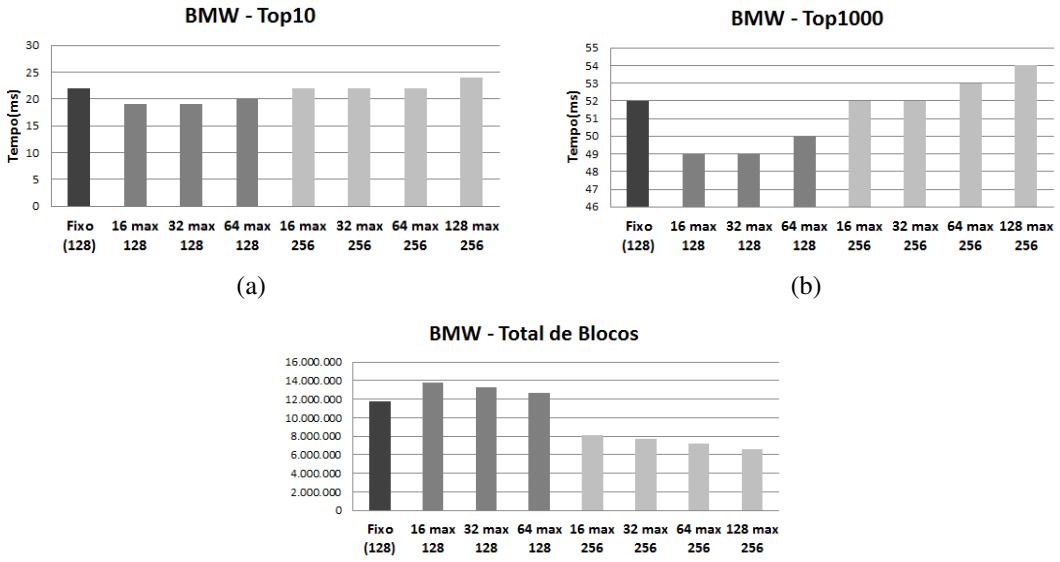


Figura 6. Desempenho do algoritmo BMW com a utilização das abordagem da variação do tamanho de blocos (máximo 128 e 256 documentos).

Alterações no tamanho dos blocos podem gerar efeitos colaterais no desempenho dos algoritmos, dado que elas também alteram os valores de escore máximo dos blocos e, consequentemente, afetam também as estratégias de poda de documentos, tanto do BMW, quanto do BMW-CS. No entanto, como foi visto nesta seção, podemos ter uma redução do número total de blocos formados sem ter impacto negativo no tempo de processamento. Unindo-se o uso de blocos de tamanho variável à estratégia de inserção de limiar inicial, conseguimos obter ganhos no tempo de processamento e, ao mesmo tempo, redução no tamanho das *skiplists* usadas para representar os blocos de documentos durante o processamento de consultas.

Também realizamos experimentos para estudar a aplicação de blocos de tamanho variável ao algoritmo BMW-CS. Mudamos o tamanho dos blocos de 64 até 1024 documentos nas duas camadas e analisamos o desempenho do algoritmo para cada porcentagem de variação da primeira camada, para top-10 e top-1000 respostas. Todas as opções de implementações (com a primeira, segunda ou ambas camadas variáveis) foram comparadas ao algoritmo BMW-CS usando blocos fixos com 128 documentos e a estratégia de limiar inicial proposta por nós. Os experimentos mostram que não obtivemos ganhos no tempo de processamento tanto para o cenário de top-10 como top-1000 respostas. A Tabela 4 mostra a média de tempo entre as variações de 18% e 50% da primeira camada para todas as opções de implementação no cenário de top-1000 respostas. Constata-se que os tempos de processamento obtidos pelas abordagens são bem similares.

Tendo em vista que o tempo de processamento entre as estratégias de blocos variáveis não apresenta grandes variações, podemos levar em consideração, como nas abordagens anteriores, a possibilidade de reduzir custos de memória. A Figura 7 faz um comparativo do total de blocos entre todas as opções de implementação, inclusive a utilizada no algoritmo BMW-CS original. Ainda na figura, notamos que o algoritmo BMW-CS fixo com 128 documentos configura a pior hipótese de implementação, visto

Tabela 4. Blocos Variáveis - Média do tempo de processamento em milisegundos calculada a partir da variação de 18% da primeira camada.

	1ª camada variável	2ª camada variável	1ª e 2ª camadas variáveis
Mín 64 Máx 256	48,47	48,88	49,18
Mín 64 Máx 512	48,76	49,65	50,41
Mín 64 Máx 1024	49,00	50,12	51,35
BMW-CS Original: 57,94			
BMW-CS Original c/ limiar: 47,82			

que o mesmo totaliza a média de aproximadamente $11,8$ milhões de blocos formados.

A opção de implementação de variar as duas camadas com o mínimo de 64 e máximo 256 documentos gera uma redução aproximada de 43% em média em relação ao BMW-CS original. Tais resultados são obtidos com uma mudança pequena no tempo de processamento. Por exemplo, em nossos experimentos, o tempo médio de processamento obtido ao considerarmos todos os limiares de tamanho estudados para a primeira camada (Tabela 4) no algoritmo BMW-CS original, que utiliza a estratégia de limiar inicial, é de $47,82$(ms), enquanto que temos um tempo médio de $49,18$(ms) ao variamos o tamanho dos blocos nas duas camadas com o mínimo de 64 e máximo 256 documentos.

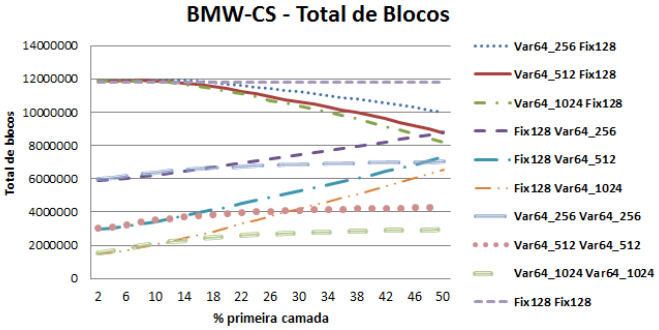


Figura 7. Número total de entradas das skiplists para cada porcentagem da primeira camada - blocos variáveis.

6. Conclusões e Trabalhos Futuros

Os resultados obtidos mostram que a estimativa de limiares iniciais para descartes de documentos pode melhorar o desempenho do algoritmo BMW e sua variante BMW-CS tanto para o cenário de top-10 como top-1000 respostas. Os maiores ganhos ao modificarmos o BMW para usar nossa estratégia de limiar inicial de poda ocorreram com blocos fixos de tamanho 64, havendo redução de cerca de 5,2% e 7,8% no tempo de processamento para top-10 e top-1000 respostas, respectivamente. Obteve-se ganhos ainda maiores ao modificarmos o algoritmo BMW-CS, com redução de aproximadamente 17% no tempo de processamento.

A estratégia de variação do tamanho dos blocos da estrutura de índices não resultou na redução de tempo de processamento, mas proporcionou reduções significativas no número de entradas das *skiplists*, tendo impacto portanto na quantidade de memória extra requerida por cada método para processar consultas. Para o algoritmo BMW, a melhor configuração encontrada nos experimentos empregou blocos com o máximo de 256

entradas. Nela mantivemos o mesmo tempo de processamento do algoritmo original e reduzimos em cerca de 35% o número total de blocos criados. No algoritmo BMW-CS, a opção de variar o tamanho dos blocos nas duas camadas com o mínimo de 64 e máximo 256 documentos gerou em nossos experimentos uma redução média de 43% em relação ao BMW-CS original, sem mudanças significativas no tempo de processamento.

Como trabalho futuro, seria interessante estudar o impacto das técnicas aqui implementadas em cenários onde o índice do sistema de busca não cabe em memória. Em tais situações, ganhos de espaço nas *skiplists* podem tornar-se mais relevantes e ter impacto positivo no tempo de processamento, uma vez que *skiplists* menores têm maior chance de serem colocadas completamente em memória. Também seria interessante o estudo do impacto das heurísticas de detecção de limiares quando aplicados a outros problemas práticos, como o da classificação de documentos [Cristo et al. 2003].

Referências

Baeza-Yates, R., Ribeiro-Neto, B., et al. (2011). *Modern information retrieval*. ACM press New York.

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. (2003). Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434. ACM.

Cristo, M., Calado, P., de Moura, E. S., Ziviani, N., and Ribeiro-Neto, B. A. (2003). Link information as a similarity measure in web classification. In *String Processing and Information Retrieval, 10th International Symposium, SPIRE 2003, Manaus, Brazil, October 8-10, 2003, Proceedings*, pages 43–55.

Ding, S. and Suel, T. (2011). Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 993–1002. ACM.

Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241. Springer-Verlag New York, Inc.

Rossi, C., de Moura, E. S., Carvalho, A. L., and da Silva, A. S. (2013). Fast document-at-a-time query processing using two-tier indexes. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 183–192. ACM.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Strohman, T. and Croft, W. B. (2007). Efficient document retrieval in main memory. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 175–182. ACM.