Explorando o Uso de *Short-term Memory* na Construção de Métodos de Acesso Métricos Mais Eficientes *

Régis Michel dos Santos Sousa, Humberto Razente, Maria Camila N. Barioni [1]

[1] Faculdade de Computação (FACOM)
Universidade Federal de Uberlândia (UFU) – Uberlândia, MG

regismaicon@gmail.com, {humberto.razente,camila.barioni}@ufu.br

Abstract. *Similarity queries are fundamental operations for applications that deal with complex data. This work proposes a new approach to create the metric access method Slim-tree through a short-term memory. The strategy was evaluated in a dynamic context and the preliminary experiments resulted in less distance calculations, disk accesses and execution time to run k-nearest neighbor queries.*

Resumo. *Indexação e recuperação por similaridade são operações fundamentais para aplicações que lidam com dados complexos. Este trabalho propõe uma nova estratégia para indexação no método de acesso métrico Slim-tree que emprega uma estrutura auxiliar denominada short-term memory. A estratégia foi avaliada em um cenário dinâmico e os experimentos preliminares mostram uma redução significativa do número de cálculos de distância, do número de acessos a disco e do tempo de execução em consultas aos k-vizinhos mais próximos.*

1. Introdução

Aplicações do mundo real lidam com cenários dinâmicos nos quais dados são gerados ilimitadamente a cada foto tirada, ligação telefônica, mensagem enviada ou transação de comércio eletrônico efetuada. Praticamente toda ação do cotidiano gera uma nova informação que atualiza um banco de dados. Esses dados são conhecidos como dados complexos e podem assumir uma série de formatos, tais como: dados vetoriais, dados multimídia, séries temporais, entre outros. Nesse contexto, surge a necessidade de formular estratégias de armazenamento, organização e recuperação destes dados de maneira eficiente.

As estruturas que permitem indexar e recuperar rapidamente dados complexos são conhecidas como Métodos de Acesso Métricos (MAM). Estas estruturas reduzem o número de cálculos de distância e o número de acessos a disco durante as operações de consulta por similaridade em relação à comparação sequencial dos dados. No entanto, o custo computacional dos algoritmos que determinam a similaridade entre pares de objetos tornam as buscas por similaridade operações de custo elevado. Estes eventos têm motivado estudos com o objetivo de tornar a busca por similaridade eficiente sobre grandes conjuntos de dados.

Em diversos trabalhos encontrados na literatura, por exemplo [Oliveira et al. 2015, Souza et al. 2014, Skopal 2006, Traina et al. 2002,

*Trabalho realizado com apoio financeiro da Capes e do CNPq.

Ciaccia et al. 1997], conjuntos de dados são indexados para posterior realização de lotes de consultas para avaliação do desempenho com relação ao número de cálculos de distância, acessos a disco e outros parâmetros. Essas abordagens fazem o uso de bases de dados estáticas apesar dos métodos de acesso propostos serem dinâmicos e não consideram a realização de novos testes após a inserção de novos elementos.

A medida que novos objetos são inseridos os nós da estrutura de indexação tendem a aumentar e, consequentemente, seus raios de cobertura também aumentam. Assim, eleva-se a probabilidade de ocorrer sobreposição entre as regiões de cobertura dos nós, o que, em geral, resulta na degradação do desempenho durante as consultas por similaridade. Em um cenário dinâmico de geração de dados a distribuição dos dados pode mudar ao longo do tempo [Gama 2012] o que pode contribuir para o crescimento dos raios de cobertura e aumentar a sobreposição nas estruturas de indexação.

Nesse contexto, este trabalho propõe uma nova técnica de indexação que tem como finalidade tornar mais eficiente MAM tanto durante o processo de construção quanto durante o processo de consultas por similaridade. A técnica proposta pode ser aplicada a estruturas métricas ou multidimensionais, como a M-tree [Ciaccia et al. 1997] e a R-tree [Guttman 1984]). No trabalho apresentado aqui, ela foi implementada no MAM Slim-tree [Traina et al. 2002] por meio da modificação do processo de construção da árvore, sem alterar a estrutura de dados e os algoritmos de busca.

Este artigo está organizado da seguinte maneira. Na Seção 2 é apresentada a fundamentação teórica. A Seção 3 apresenta os métodos e técnicas utilizados no trabalho. Na Seção 4 são discutidos os resultados experimentais. As considerações finais são descritas na Seção 5.

2. Fundamentação Teórica

Um espaço métrico é um par $< W, d() >$, onde W é um domínio de objetos e $d()$ é uma função de distância (métrica) que satisfaz os seguintes axiomas para qualquer elemento $x, y, z \in W$: $d(x, x) = 0$ (identidade); $d(x, y) = d(y, x)$ (simetria); $0 \leq d(x, y) < \infty$ (não-negatividade); e $d(x, y) \leq d(x, z) + d(z, y)$ (desigualdade triangular).

Uma consulta por similaridade é um processo para obter um conjunto de objetos ordenados pelas distâncias para um dado objeto de consulta. Entre as principais técnicas utilizadas em consultas por similaridade estão os MAM. Esses métodos criam hierarquias com base na relação de distância entre os objetos. Entre eles destacam-se os MAM dinâmicos *M-Tree* [Ciaccia et al. 1997], *Slim-Tree* [Traina et al. 2002], *DBM-Tree* [Vieira et al. 2010] e DSAT [Navarro and Reyes 2016]. Uma vez que objetos estejam indexados em alguma dessas estruturas a recuperação é realizada por meio de consultas por abrangência (*range query*) ou consultas aos k vizinhos mais próximos (*k-nearest neighbors query*) [Ciaccia et al. 1997].

3. Indexação Assistida por Memória Auxiliar com Capacidade Limitada

A Slim-Tree é um MAM com construção incremental. Essa característica permite seu uso para indexação por similaridade em cenários dinâmicos. Este trabalho apresenta um nova abordagem de inserção de objetos no qual uma memória auxiliar com capacidade limitada de armazenamento é utilizada para processamento temporário dos dados.

O processo de inserção da Slim-tree inicia com o percurso da raiz até uma folha por meio de heurísticas para escolha de sub-árvore (*ChooseSubtree*). A inserção ocorre em uma folha e em caso de *overflow* um algoritmo de divisão é empregado, gerando uma nova folha, resultando na distribuição dos objetos entre o nó corrente e o novo nó. A inserção na folha pode propagar alterações recursivamente até a raiz. Tal mecanismo é conhecido por construção *bottom-up* e garante que a estrutura seja sempre balanceada. Entretanto, esse processo pode resultar em grande sobreposição não somente nos nós folha como também nos nós índice. Essa sobreposição impacta diretamente a eficiência das consultas, pois nós sobrepostos não podem ser podados durante o percurso na árvore.

A estratégia proposta neste artigo partiu da suposição de que ao permitir o adiamento da inserção de um objeto no local definitivo pode-se agrupá-lo com novos objetos mais similares que ainda serão inseridos, contribuindo para minimização dos raios de cobertura. Assim a operação de inserção é realizada da seguinte maneira. Se a operação de inclusão no nó folha resultar em um aumento do raio da sub-árvore, o novo processo de inserção inclui o objeto na *short-term memory*, postegando a inserção de objeto que cause o aumento da sobreposição imediata no MAM. Desse modo, uma análise posterior será realizada para possibilitar que esse objeto seja indexado. O processo completo é apresentado na Figura 1.

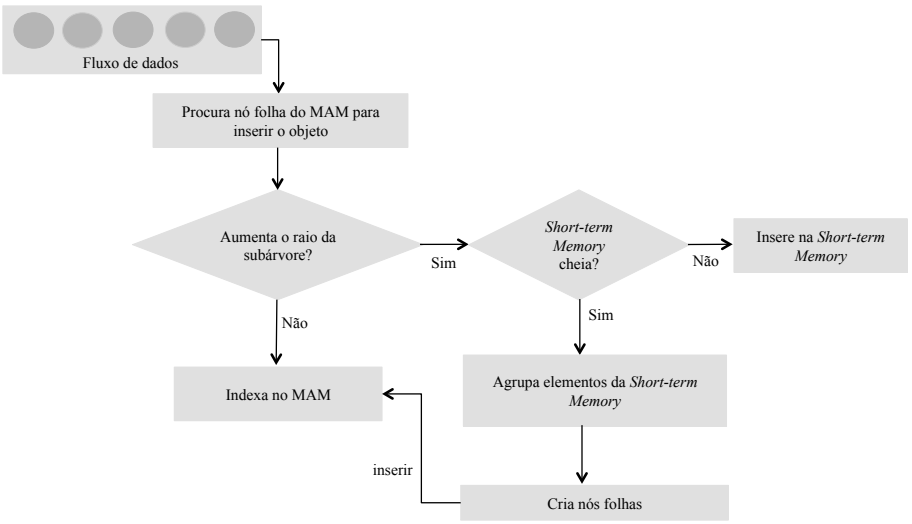

Figura 1. Visão geral do processo de inserção de elementos no MAM.

Este processo ocorre até que a *short-term memory* esteja cheia. Nesse caso um novo nó folha é criado com custo $O(n)$, onde n é o tamanho da *short-term memory*, respeitando-se a taxa de ocupação para ser inserido na estrutura de indexação. As seguintes etapas são executadas quando a capacidade da *short-term memory* é atingida:

- criação de um nó folha de tamanho fixo e escolha aleatória de um objeto da *short-term memory* para ser o representante;
- inserção dos elementos da *short-term memory* mais próximos do representante até o limite de espaço do nó respeitando-se a taxa de ocupação;
- exclusão na *short-term memory* dos elementos inseridos no novo nó;
- novo percurso da raiz até o nível das folhas para inclusão do novo nó folha, resultando na atualização recursiva dos nós índices até a raiz, caso necessário.

A estratégia permite que um novo objeto que não esteja coberto por alguma folha aguarde na *short-term memory* e se agrupe com outros objetos próximos formando uma nova folha, que terá capacidade para receber novas inserções de objetos dentro da sua região de cobertura devido à fragmentação resultante da taxa de ocupação.

4. Experimentos e Análise dos Resultados

A técnica foi implementada na linguagem C++ e utiliza a biblioteca Arboretum[1]. Os experimentos foram executados em um microcomputador com sistema operacional Linux 64 bits, i7-4770 @3.40GHz, 8GB de RAM e disco rígido de 1TB. Com o objetivo de explorar de forma eficaz a técnica proposta neste trabalho foram realizados experimentos com bases de dados reais disponíveis no UCI Machine Learning Repository [Lichman 2013], denominadas *Covertype* e *KDD Cup 1999 Data*. Os seguintes parâmetros de configuração foram empregados nos experimentos: taxa de ocupação dos nós criados a partir da *short-term memory*: 75%; função de distância: Euclidiana; tamanho da *short-term memory*: 500 objetos; tamanho das páginas de disco: 8 KB; método para *split*: algoritmo MinMax; e escolha aleatória de 100 objetos do conjunto para realização das consultas. Os experimentos consideram um cenário dinâmico com medições realizadas pela chegada dos objetos a cada 10% da base de dados. Para realização das consultas todos os objetos da *short-term memory* são inseridos no MAM. Desse modo é possível analisar o desempenho dos MAM ao longo do tempo de acordo com as novas inserções.

A seguir são apresentados os resultados obtidos para a base de dados *Covertype* que contém 581.102 objetos com 54 atributos numéricos que foram reescalados para o intervalo [0,1]. Como pode ser observado na Figura 2, durante a construção da Slim-Tree com o método proposto (SM Slim-tree), a mesma apresenta um ganho no tempo de execução de 55% (Figura 2c), uma diferença de 59% em cálculos de distância (Figura 2a) e quantidade similar de acessos a disco (Figura 2b) ao se atingir 100% da base de dados, ou seja, 581.102 objetos. É importante observar que mesmo com os cálculos de distância para operação da *short-term memory* o total de cálculos de distância para construção do índice foi menor. Isto pode ser explicado pelo fato de melhorar a distribuição dos elementos entre os nós folha, ou seja, retendo o efeito de elementos outliers que trariam aumento imediato aos raios de cobertura, causando aumento da sobreposição entre os nós.

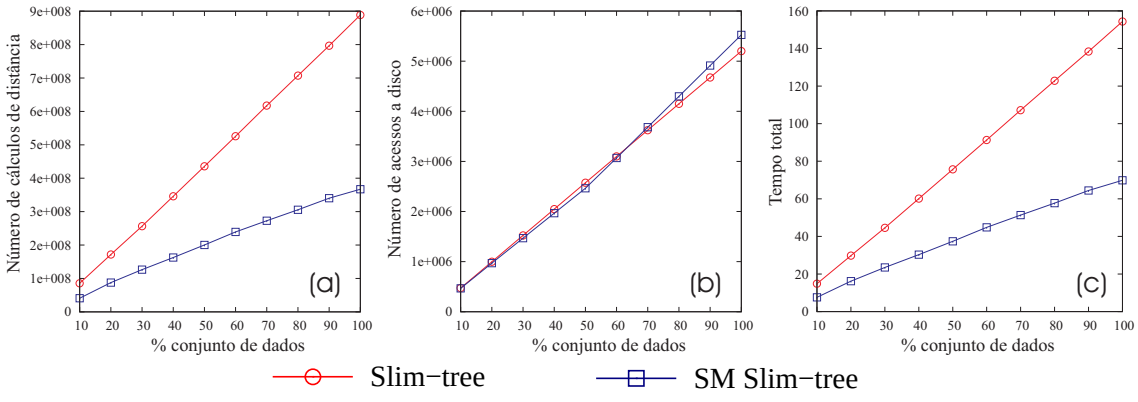

Figura 2. Construção com a base *Covertype*. (a) Número de cálculos de distância. (b) Número de acessos a disco. (c) Tempo total.

[1]Disponível em http://gbdi.icmc.usp.br/arboretum

Para avaliação do processamento de consultas, os gráficos apresentados são relativos ao processamento de 100 consultas, onde cada consulta recuperou os 100 vizinhos mais próximos. No gráfico da Figura 3a pode-se observar o desempenho de até 48% menos cálculos de distância, 52% menos acessos a disco na Figura 3b e tempo de execução 51% menor (Figura 3c).

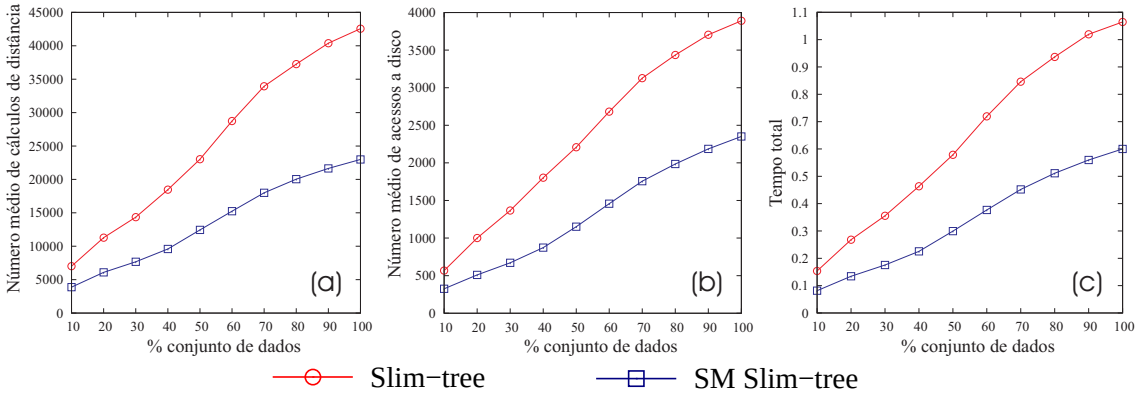

Figura 3. Consultas com a base *Covertype*. (a) Número médio de cálculos de distância. (b) Número médio de acessos a discos. (c) Tempo total.

A seguir são apresentados os resultados obtidos para a base de dados *KDD Cup 1999* que contém 4.898.431 objetos com 34 atributos numéricos que foram reescalados para o intervalo [0,1]. Como pode ser observado na Figura 4, durante a construção da Slim-Tree com o método proposto (SM Slim-tree), a mesma apresenta um ganho no tempo de execução de 66% (Figura 4c), uma diferença de 67% em cálculos de distância (Figura 4a) e ganho de até 11% em acessos a disco (Figura 4b) ao se atingir 100% da base de dados.

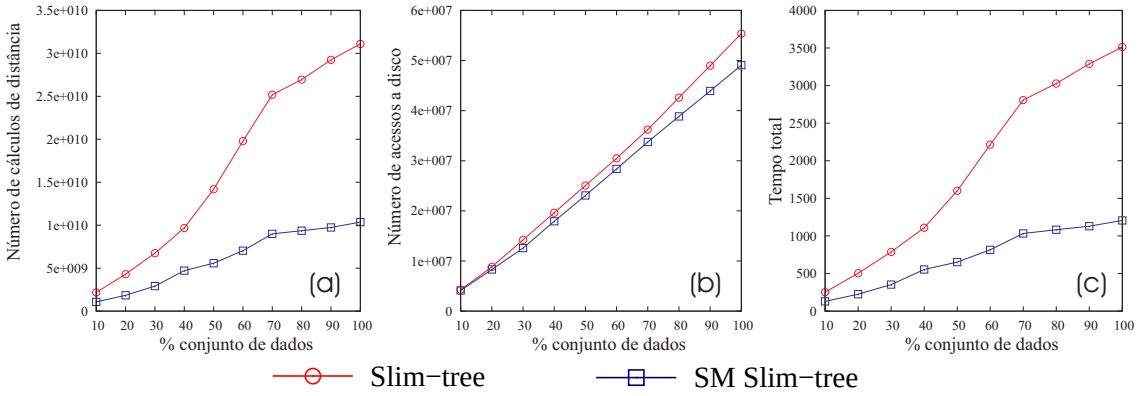

Figura 4. Construção com a base *KDD Cup 1999*. (a) Número de cálculos de distância. (b) Número de acessos a disco. (c) Tempo total.

No gráfico da Figura 5a pode-se observar o desempenho de até 14% menos cálculos de distância, 63% menos acessos a disco na Figura 5b e tempo de execução até 22% menor (Figura 5c).

5. Considerações Finais

O desenvolvimento de métodos de acesso métricos dinâmicos eficientes é fundamental para o processamento de consultas por similaridade em grandes bases de dados. Assim, este trabalho apresenta uma nova estratégia para melhorar o desempenho do método

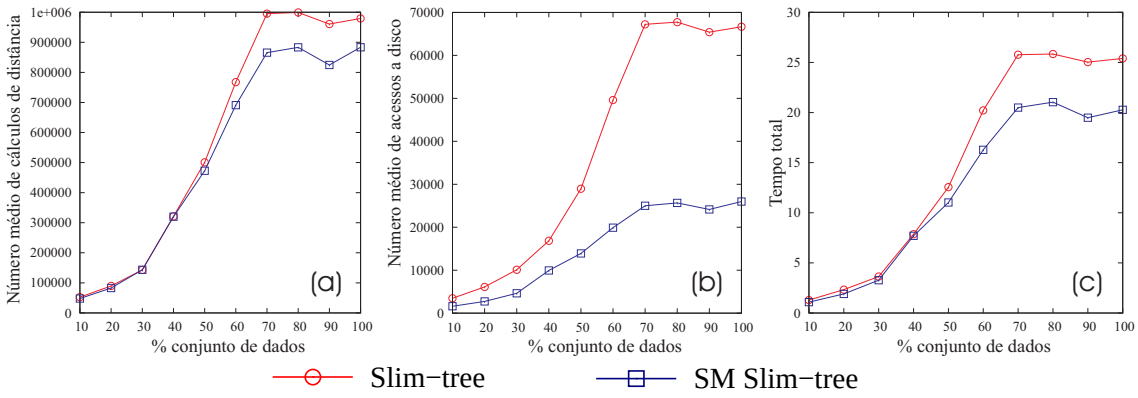

Figura 5. Consultas com a base *KDD Cup 1999*. (a) Número médio de cálculos de distância. (b) Número médio de acessos a discos. (c) Tempo total.

de acesso métrico Slim-Tree. Por meio dos experimentos realizados verificou-se que o método proposto é mais eficiente durante a construção e consultas por similaridade em relação ao número de cálculos de distância, de acessos a disco e ao tempo de execução quando comparado a abordagem padrão da Slim-Tree.

Dentre os trabalhos futuros planeja-se: investigar estratégias baseadas em agrupamento para inclusão dos elementos da *short-term memory* no índice; avaliar a utilização dessas estratégias na construção de outros métodos de acesso métricos e multidimensionais; e avaliar a sobreposição resultante da utilização dessas estratégias.

Referências

Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, Atenas, Grécia.

Gama, J. (2012). A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, Boston, Massachusetts.

Lichman, M. (2013). UCI Machine Learning Repository, University of California, Irvine, http://archive.ics.uci.edu/ml.

Navarro, G. and Reyes, N. (2016). New dynamic metric indices for secondary memory. *Information Systems*, 59:48–78.

Oliveira, P., Traina, C., and Kaster, D. (2015). Improving the pruning ability of dynamic metric access methods with local additional pivots and anticipation of information. In *ADBIS, LNCS 9282*, pages 18–31, Poitiers, França. Springer.

Skopal, T. (2006). On fast non-metric similarity search by metric access methods. In *EDBT, LNCS 3896*, pages 718–736, Munique, Alemanha. Springer.

Souza, J., Razente, H., and Barioni, M. C. (2014). Optimizing metric access methods for querying and mining complex data types. *J. Braz. Comput. Soc.*, 20(1):1.

Traina, C., Traina, A., Faloutsos, C., and Seeger, B. (2002). Fast indexing and visualization of metric data sets using slim-trees. *IEEE Trans Knowl Data Eng*, 14(2):244–260.

Vieira, M. R., Jr., C. T., Chino, F. J. T., and Traina, A. J. M. (2010). Dbm-tree: A dynamic metric access method sensitive to local density data. *JIDM*, 1(1):111–128.