Investigação e criação de um algoritmo de bulk loading para métodos de acesso métricos, utilizando técnicas de agrupamento

Eduardo Leal[1], Mauro Biajiz[1] e Luis Carlos Trevelin[1]

[1] Universidade Federal de São Carlos, Departamento de Computação, Rodovia Washington Luis, sn. São Carlos, SP, Brasil
{leal, mauro, trevelin}@dc.ufscar.br

Abstract. Métodos de acesso são usados na computação para otimizarem o tempo de recuperação dos dados armazenados e são classificados de acordo com características bem definidas, de acordo com o domínio de dados que suportam. Neste trabalho foi abordado o método de acesso métrico denominado Slim-tree e investigado como incrementar seu algoritmo de bulk loading com a utilização de uma técnica de agrupamento.
O objetivo deste trabalho é comprovar a possibilidade do aumento de desempenho do carregamento de dados usando algoritmos de agrupamento. Testes foram realizados para validarem o algoritmo e demonstrarem seu desempenho.

Keywords: bulk loading, métodos de acesso, métodos de acesso métricos.

1 Introdução

As operações de bulk são definidas como operações executadas em lote sobre uma estrutura de índice em um mesmo tempo, sem a interrupção de outras ações. Elas podem ser de carregamento (loading), inserção, atualização ou remoção.

Existem basicamente duas formas distintas de popular a estrutura do índice, sendo a mais conhecida, o uso do algoritmo de inserção padrão (inserção um-a-um), onde um registro é inserido de cada vez na estrutura do índice. Outra forma é usando um algoritmo de bulk loading: onde o índice é construído a partir de um conjunto de dados pré-conhecidos que são inseridos na estrutura em uma série de operações executadas em um único processo, sem interrupção de outras ações.

Dentro do contexto de aplicação dos métodos de acesso, o carregamento eficiente (rápido e preciso) dos dados é um fator primordial para seu bom desempenho. Este artigo investiga a modificação de um algoritmo de bulk loading existente através da inclusão de uma técnica de agrupamento. Para validar sua adequação e corretude, o algoritmo foi implementado usando a estrutura de indexação métrica Slim-tree. Foram efetuadas medidas de desempenho do algoritmo proposto e estas foram comparadas com as medidas obtidas pela execução de um algoritmo básico existente. Os resultados obtidos mostraram a eficiência e escalabilidade do novo algoritmo, uma vez que ele faz até 20% menos acessos a disco e até 15% menos cálculos de distância do que o algoritmo básico correspondente.

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 689-698, 2008.

Este trabalho está organizado da seguinte forma: na Seção 2 é apresentada uma revisão da literatura dos trabalhos relacionados; na Seção 3 são descritos como a SlimTree faz a inserção dos dados e a proposta de um novo algoritmo BulkSlim; a Seção 4 descreve os experimentos realizados e os resultados obtidos; a Seção 5 finaliza o artigo apresentando as conclusões e os trabalhos futuros a serem realizados.

2. Estruturas de indexação e bulk loading

As estruturas de indexação podem ser divididas em quatro grandes categorias: as estruturas básicas, os Métodos de Acesso Espacial Puntuais (MAEP), os Métodos de Acesso Espacial Não-Puntuais (MAENP) e os Métodos de Acesso Métricos (MAM). O primeiro grupo compreende as estruturas a partir das quais as outras estruturas foram derivadas. As técnicas de hashing e a B-tree, dentre outras são exemplos deste grupo. O segundo e terceiro grupos consistem de estruturas que fazem indexação em espaços Euclidianos n-dimensionais. A diferença entre os dois grupos está no fato de os MAEP indexarem pontos, enquanto que os MAENP serem capazes de indexar também objetos não-puntuais. A Grid File, a K-D-B-tree, dentre outras estruturas, são exemplos de MAEP, enquanto que a família R-tree (R-tree, R+-tree, Greene R-tree e R*-tree), dentre outras, são exemplos de MAENP. O último grupo é aquele composto por estruturas para indexação em espaços determinados por uma função de distância. Os principais exemplos deste grupo são a M-tree, a Slim-tree e a OMNI-Family.

Uma variedade de algoritmos de bulk loading têm sido propostos para B+trees [8], Quadtrees [5], R-trees [1] [2] [3] e M-trees [4] com o objetivo de otimizar o tempo de criação ou recriação de um índice a partir de um conjunto de dados pré-conhecidos. A aplicação desses algoritmos está diretamente relacionada à estrutura de dados usada no método de indexação.

O processo de bulk loading para a B+tree ocorre primeiramente ordenando o conjunto de dados que será inserido na árvore. Em seguida é alocada uma página vazia para servir de raiz e é inserido um ponteiro para a primeira página das entradas ordenadas. Até que a raiz esteja cheia, adiciona-se uma nova entrada, que consiste de ({menor valor na página, ponteiro para a página}) para cada página dos dados de entrada ordenados. Quando a raiz estiver cheia, divide-se a página da raiz antiga e cria-se uma nova página para a raiz. As entradas nas páginas folhas são sempre inseridas na página de índice de posição mais à direita. Quando uma página de índice está cheia, divide-se o nó. Esta divisão pode se propagar para os nós de níveis superiores.

A operação de bulk loading para Quadtrees [5] se faz em duas fases distintas sendo:

a) Utiliza-se um buffer para armazenar os nós recentemente usados da B-tree e usa-se a política de LRU (least recently used) para prover espaço para os novos nós.

b) Constrói-se uma quadtree baseada na memória principal. À medida que a memória principal disponível se esgota, partes da quadtree são escritas no disco. É utilizado um conjunto de heurísticas para escolher quais dessas partes são escritas no disco, sendo que a meta é escolher as partes que não serão necessárias posteriormente.

Esse processo faz uso de um conjunto de estatísticas que são mantidas nos nós da quadtree.

No caso da R-tree, três tipos de bulk loading podem ser aplicados [1] [2] [3].

Em [1], a operação de bulk baseia-se na criação e utilização de um buffer acoplado a estrutura para a inserção dos dados, onde os retângulos (estrutura básica da R-tree) correspondentes às MBB são inseridos um a um e executa-se um descarregamento final de todos os buffers na árvore. O algoritmo de inserção em bulk é conceitualmente idêntico ao algoritmo de inserções repetidas um-a-um.

Em [2] propõe-se um algoritmo chamado STLT (Small Tree Large Tree) que consiste na inserção dos dados de entrada em small trees que posteriormente serão inseridas na large tree. Esse processo é feito em três passos: no primeiro, os novos dados de entrada são inicialmente inseridos em uma nova R-tree vazia, small tree. No passo seguinte é feita uma busca que indicará a posição que conterá a nova árvore e finalmente ela é inserida na large tree. A proposta de [3] segue essa mesma idéia, diferindo no ponto em que considera as diferenças do conjunto dos dados de entrada, como tamanho do novo conjunto em relação ao existente e se a distribuição de seus dados é uniforme. Essa nova técnica particiona o conjunto dos dados de entrada em conjuntos de agrupamentos, constrói a R-tree desses agrupamentos, identifica e prepara o local na R-tree original para a inserção e por fim executa a inserção da small tree na large tree.

O bulk loading da M-tree [4] parte do suposto que k é a quantidade máxima de elementos que um nó pode armazenar. Em uma fase inicial, o algoritmo escolhe aleatoriamente k elementos, chamados de samples. Os elementos do conjunto de entrada são então divididos em grupos de acordo com a distância entre eles e o sample. Para cada um desses 'subconjuntos' é repetido esse mesmo procedimento até que cada sample tenha um conjunto de elementos que seja menor ou igual à sua capacidade de armazenamento. Então é construída uma árvore com o sample como raiz e o conjunto de elementos associados como seus filhos. Em uma segunda fase, a árvore é construída seguindo o mesmo processo, só que tendo o conjunto de raízes das sub-árvores da fase anterior como conjunto de entradas inicial.

De acordo com o conhecimento dos autores, nenhum trabalho foi publicado visando a disponibilidade de um algoritmo de agrupamento acoplado a uma estrutura métrica. Portanto, este é o objetivo deste trabalho.

3. Descrição do trabalho

Diversos métodos de indexação espacial têm sido propostos com a finalidade de manipular tipos de dados que apresentam características espaciais, como pontos, linhas e polígonos. Eles gerenciam o espaço geométrico onde os objetos são representados de forma vetorial. Outros métodos de acesso, chamados de métodos de acesso métricos propõem uma representação do espaço de acordo com um espaço métrico, mais amplo e que engloba o espaço vetorial. Esses métodos se caracterizam por representar os objetos de acordo com sua distribuição no espaço. A Slim-tree [9] é uma estrutura de indexação métrica, baseada na M-tree, balanceada, que possui nós índice que direcionam o acesso aos nós folha, nos quais os objetos se encontram. Ela

foi criada para medir e diminuir o grau de sobreposição entre os nós da árvore métrica. Para fazer a medição foram propostos o fat-factor e o bloat-factor, que representam o grau absoluto e relativo (à árvore ótima) de sobreposição de uma árvore, respectivamente. Para fazer a diminuição da sobreposição, foi proposto o Slim-down cuja idéia é fazer a reinserção de objetos de forma conveniente a diminuir o raio de cobertura dos nós. A Slim-tree também propôs o algoritmo de inserção por mínima ocupação e o algoritmo MST (Minimal Spanning Tree) de divisão de nó e escolha dos representativos.

A Slim-tree é também uma árvore dinâmica para a organização métrica de conjuntos de dados em páginas de tamanho fixo. Ela é balanceada e cresce de forma bottom-up (das folhas para a raiz). Como outras árvores métricas, os objetos dos conjuntos de dados são agrupados em páginas de disco de tamanho fixo, cada uma correspondendo a um nó da árvore, sendo que cada nó possui também uma taxa de ocupação mínima. A intenção principal é organizar os objetos em uma estrutura hierárquica usando um objeto representativo como centro de cada região mínima limitada que cobre os objetos da subárvore.

O algoritmo padrão de inserção trabalha da seguinte forma: iniciando do nó raiz, localiza-se o nó que pode conter o novo objeto. Se nenhum nó for qualificado, seleciona o nó cujo centro é o mais próximo do novo objeto. Se houver mais do que um nó qualificado, executa o algoritmo ChooseSubtree para selecionar um deles. Esse processo é recursivamente aplicado para todos os níveis da árvore. Se ocorrer um overflow em um nó, um novo nó é alocado no mesmo nível e os objetos são distribuídos entre eles. Quando um nó raiz é dividido, uma nova raiz é alocada e a árvore cresce um nível.

Basicamente, os passos adotados na inserção de um elemento k na árvore são:

a) encontre o local correto (folha) L para inserir k;

b1) se L pode conter k, insira-o e ajuste o raio do nó índice dessa folha;

b2) senão, é necessária uma divisão de L (em L e um novo nó L2)

redistribua os elementos dos nós folhas entre L e L2

obs: esses passos podem acontecer recursivamente

b3) para dividir um nó do tipo índice, redistribua as entradas deste nó entre os novos nós de índices

b4) as divisões podem levar a um crescimento da árvore; a divisão da raiz aumenta a altura da árvore.

3.1 Técnica de bulk loading para a Slim-tree

O algoritmo básico bulk loading da Slim-tree foi proposto por [7] e utiliza uma abordagem bottom-up construindo uma árvore a partir da leitura de um conjunto de dados existente, particionando e agrupando sucessivamente os elementos desse conjunto de acordo com sua posição em um espaço métrico, e por fim executando uma nova versão do algoritmo SlimDown sobre a estrutura de indexação gerada. A execução do Slim-Down visa otimizar a árvore gerada através da redução do overlap entre os seus nós.

O algoritmo pode ser decomposto em duas fases de construção distintas, como indicado na figura 1. Basicamente ele se compõe de duas fases, sendo que na fase 1 procura-se construir as sub-árvores que vão conter os objetos de dados, nós_folhas, com a granularidade desejada. Na fase 2: constrói-se os níveis de indexação para as subárvores geradas na fase 1.

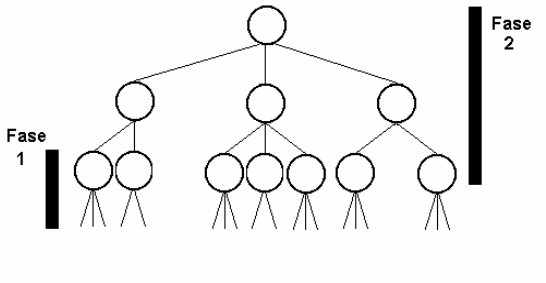
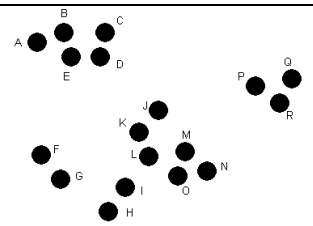
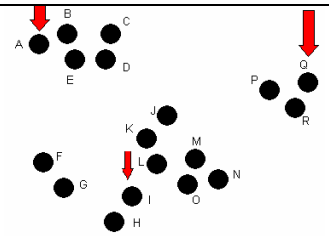
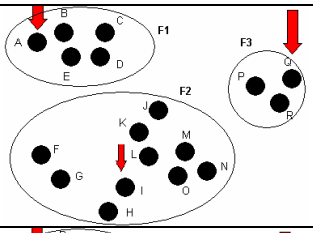
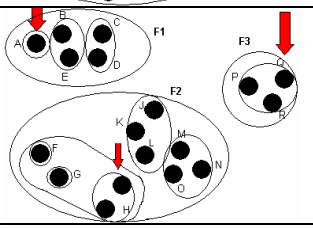
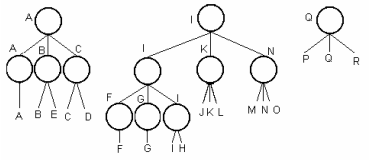
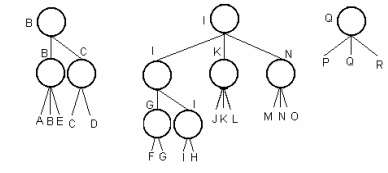


Fig. 1. Fases do algoritmo de bulk loading.

Descrição resumida do algoritmo: Dado um conjunto de objetos S com n objetos Oi, S = {O1, O2, ..., On}, escolhe-se aleatoriamente k de S e os insere em um conjunto F. O k corresponde à taxa de ocupação máxima de um nó. Para cada objeto de S, compara-se sua distância em relação a cada um dos objetos de F e o associa ao que tiver menor distância, produzindo k conjuntos F (F1, F2, ..., Fk). Então se invoca recursivamente o algoritmo de bulk loading para cada um dos elementos do conjunto F até que cada conjunto tenha uma quantidade de elementos que seja menor ou igual a k. O efeito dessas chamadas recursivas é que nas folhas de cada sub-árvore, tem-se uma partição do conjunto de dados com o nível desejado de granularidade. Chama-se o algoritmo novamente para construir a superárvore a partir das sub-árvores encontradas. As seguintes estratégias foram definidas para preservar as propriedades de taxa de ocupação e balanceamento da Slim-tree: A **taxa de ocupação mínima dos nós** é garantida verificando em cada sub-árvore gerada a quantidade de filhos que ela possui. Caso essa quantidade seja menor que a taxa de ocupação mínima, realoca-se cada um dos seus filhos para a sub-árvore irmã dessa que tenha a menor distância desse filho, e elimina-se essa sub-árvore. Esse procedimento garante que cada nó da sub-árvore tenha uma taxa mínima de ocupação e conduz a um número menor de elementos no conjunto F. Se após esse procedimento o número de elementos do conjunto F for reduzido a 1 (apenas um elemento), ele é descartado e retorna-se para a fase de escolha dos k elementos aleatórios para esse conjunto. Para garantir o **balanceamento da árvore** procede-se da seguinte forma: se a sub-árvore tem alturas diferentes, divide-se a sub-árvore de maior altura, produzindo um número de sub-árvores menores: as raízes obtidas dessas sub-árvores serão inseridas no conjunto F, trocando o objeto que anteriormente estava nesse conjunto. Esse procedimento garante que todas as sub-árvores em um conjunto F tenham a mesma altura, embora aumente o número de elementos desse conjunto F.

	Considere 18 objetos distintos distribuídos em um espaço métrico.
	Aleatoriamente são escolhidos k objetos que irão compor nosso conjunto. Nesse caso 3 objetos (A, I, Q).
	Para cada objeto de S, é comparada sua distância para cada um dos objetos de F. Cada objeto analisado é associado ao que tiver menor distância, produzindo k (=3) conjuntos F.
	Para cada um dos conjuntos F, chamamos recursivamente o algoritmo de bulk-loading, gerando novas partições e agrupamentos.
	Cada um dos agrupamentos corresponde a uma sub-árvore. Para cada uma das sub-árvores, os passos são descritos em (a) e (b):
	a) verificamos se a quantidade de filhos está abaixo do limite inferior da taxa de ocupação mínima. Se necessário realocamos seus filhos e eliminamos essa sub-árvore. Observe que o nó A foi eliminado e seu filho transferido para o nó B, o mesmo valendo para o nó F e G.

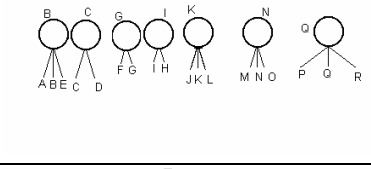
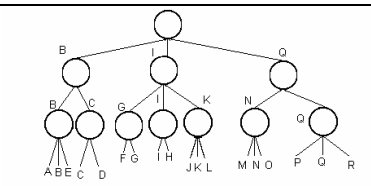
	b) verificamos a altura da sub-árvore em relação às outras sub-árvores. Se necessário, dividimos a sub-árvore mais alta, gerando duas novas sub-árvores menores. Repetimos esse passo até termos todas as sub-árvores com a mesma altura.
	Chama-se novamente o algoritmo de bulk loading para construir a superárvore a partir do conjunto das raízes das sub-árvores.

Fig. 2. Exemplo de bulk loading com k = 3 e taxa de ocupação mínima de 2.

3.2 O algoritmo de bulk loading com técnica de agrupamento

A técnica de algoritmos de agrupamento envolve o cálculo de elementos representativos (centróides) de cada agrupamento e são frequentemente referenciados como algoritmos k-means. Para verificar a viabilidade de nossa proposta, acoplamos o algoritmo EAC [6], pelo fato de os autores já terem um contato prévio com ele. Esse algoritmo, basicamente, particiona um conjunto de dados de N objetos em k agrupamentos. O valor de k, que representa o número de agrupamentos é especificado pelo usuário. Este algoritmo minimiza os cálculos das distâncias (Euclidianas) entre os objetos de um agrupamento e o seu centróide.

Este novo algoritmo de bulk loading substitui a escolha aleatória dos elementos representativos por uma lista de centróides escolhida através do EAC. Com isso, se ganha em processamento ao evitar as comparações necessárias ao posterior agrupamento interno no algoritmo padrão.

A linha 6 do algoritmo (apresentado abaixo) mostra a chamada da função de agrupamento.

```
Algoritmo BulkSlim
Comentários
// |S| : quantidade de elementos em uma lista S
// MinEntries : quantidade mínima de elementos em um nó
// MaxEntries : quantidade máxima de elementos em um nó
Entradas: S: lista_de_elementos, MaxEntries: inteiro,
MinEntries: inteiro
Saída: nó raiz
1 : SE | S | <= MinEntries ENTAO
2:          cria um nó de indice X
3:          aloca todos os elementos de S como filhos de X
4:          RETORNA X
5: SENAO
6:          F := EAC(S)
7:          PARA cada elemento de F FACA
8:            SE |lista de elementos associados de F| <
MinEntries ENTAO
```

```
9:            remova todos os elementos de sua lista de
associados
10:           associe esses elementos a outro F de menor
distância
11:           remova o F que agora está sem lista de
associados
12:         FIM-SE
13:       FIM-PARA
14:       SE |F| = 1 ENTAO
15:         remova esse elemento
16:         volte para a fase de escolha dos k elementos
17:       FIM-SE
18:       PARA cada F FACA
19:         Tj = BulkSlim( lista de elementos associados
de F, MaxEntries, MinEntries );
20:         SE a raiz de Tj não contiver um numero de
elementos mínimo ENTAO
21:           Divida Tj em sub-árvores // uma para cada
filho
22:         FIM-SE
23:         h  = a altura da menor sub-árvore de Tj
24:         T' = ∅ um conjunto de sub-árvores
25:         PARA cada Tj FACA
26:           SE altura de Tj > h ENTAO
27:             Remova o objeto de F
28:             Divida Tj em p sub-árvores de altura h
29:             Insira essas sub-arvores em T'
30:             Insira as raízes dessas sub-árvores em F
31:           SENAO
32:             insere Tj em T'
33:           FIM-SE
34:         FIM-PARA
35:         Tsup = BulkSlim( F, MaxEntries, MinEntries)
36:         anexe cada Tj ∈ T'como folha de Tsup
37:         atualize o raio da região de T
38:         retorna T
39:       FIM-PARA
40: FIM-SE
```

4. Resultados e Testes

Nesta seção são apresentados os resultados obtidos pelos algoritmos propostos versus os obtidos pelas composições dos algoritmos básicos. Para se determinar a eficiência do método de inserção foram considerados principalmente dois fatores: o **número de acessos a disco**, e o **custo da função de distância**.

O algoritmo foi implementado em C++, como já era adotado na Slim-tree. Na implementação foram adicionados contadores para acesso aos nós e para as computações dos cálculos de distância.

Dois conjuntos de dados foram usados nos testes, conforme descrito na tabela 1. Em cada caso, as métricas foram construídas pela inserção dos objetos um-a-

um e através do bulk loading. A escolha destes conjuntos foi feita para atender o nosso objetivo de se comparar o desempenho com o mesmo conjunto apresentado em [7]. Em um próximo trabalho, confrontaremos nosso desempenho com o apresentado em [10].

Tabela 1. Descrição dos dois conjuntos de dados reais utilizados nos experimentos.

Nome	Nro. de objetos	Descrição
MGCounty	15.559	pontos geográficos em um espaço bidimensional descrevendo as coordenadas das intersecções das estradas de Montgomery County em Mariland/USA
Sierpinsky	15.000	pontos em um espaço bidimensional do triângulo de Sierpinsky

Nos testes realizados, comparamos o algoritmo BulkSlim com o algoritmo padrão de bulk loading da Slim-tree na operação de carregamento dos dados para a estrutura da árvore. As tabelas 2 e 3 mostram o desempenho dos algoritmos para dois conjuntos distintos de dados, com variação na quantidade de registros carregados.

Tabela 2. Testes sobre o conjunto MgCounty

	dimensão	métrica	método de divisão	método de escolha na inserção	
MGCounty	2	L2	SpanningTree	MinOccupancy	
	N	Acesso a páginas do disco		Cálculos de distância	
		BulkSlim	bulk loading	BulkSlim	bulk loading
	1000	901	1002	23845	26495
	3000	2967	3225	82179	89326
	5000	4316	5200	135457	163201
	7000	6574	7780	216900	256686
	10000	9354	11005	343012	403544

Tabela 3. Testes sobre o conjunto Sierpinsky

	Dimensão	métrica	divisão	escolha na inserção	
SIERPINSKy	2	L2	SpanningTree	MinOccupancy	
	N	Acesso a páginas do disco		Cálculos de distância	
		BulkSlim	bulk loading	BulkSlim	bulk loading
	1000	1041	1184	23324	26505
	3000	2672	3108	76647	89125
	5000	4487	5099	146872	166900
	7000	74610	8290	2304045	256005

Como pode ser visualizado nos resultados, houve um ganho médio de 10 a 20%. Um dos possíveis motivos dessa variação é a escolha aleatória dos elementos representativos feitos pelo algoritmo de bulk loading tradicional.

5. Conclusões e trabalhos futuros

Este artigo apresenta o algoritmo BulkSlim desenvolvido para investigar o emprego de técnicas de agrupamento em algoritmos de *bulk loading*. Os testes foram realizados sobre o método de acesso métrico *Slim tree* e mostram que nosso algoritmo proposto é mais eficiente e escalável do que o algoritmo padrão. Os resultados obtidos mostraram a eficiência e escalabilidade do novo algoritmo, com até 20% menos acessos a disco e até 15% menos cálculos de distância.

Pontos a serem estudados com mais cuidado são o desempenho do algoritmo BulkSlim com uma maior variação do tamanho do conjunto de dados e com dados de domínio específico, como os utilizados em simulações de dinâmica moleculares.

Referências

[1] Arge L.; Hinrichs K. H.; Vahrenhold J.; Vitter J. S.. Efficient bulk operations on dynamic R-trees. Proc of the 1st Workshop on Algorithm Engineering and Experimentation, Baltimore, 1999.

[2] Chen L.; Choubey R.; Rundensteiner E.A.. Bulk Insertions into R-Trees. WPI, Tech. Rep. CS-WPI-98-05, 1998.

[3] Rupesh Choubey; Li Chen; Elke ª Rundensteiner. GBI: A Generalized R-tree Bulk-Insertion Strategy. Symposium on Large Spatial Databases. p.91-108, 1999.

[4] Ciaccia P.; Patella M. Bulk loading the M-tree. Proc of the 9th Australasian Database Conference (ADC'98), p.15-26, Perth, Australia, 1998.

[5] G. R. Hjaltason, H. Samet, and Y. Sussmann. Speeding up bulk-loading of quadtrees. Proc. 5th ACM-GIS Workshop, p. 50--53, Las Vegas, NV, 1997.

[6] Hruschka, E. R., Castro, L. N., and Campello, R. J. 2004. Evolutionary Algorithms for Clustering Gene-Expression Data. In Proceedings of the Fourth IEEE international Conference on Data Mining (Icdm'04) - Volume 00 (November 01 - 04, 2004). ICDM. IEEE Computer Society, Washington, DC, 403-406.

[7] LEAL, EDUARDO. Definição e Implementação da Persistência e Construção de um Algoritmo de Bulk Loading para Slim-tree. Dissertação de Mestrado em Ciência da Computação. Universidade Federal de São Carlos. 2001

[8]. RAMAKRISHNAN, RAGHU. Database Management Systems. The McGraw-Hill Companies, Inc. 1997.

[9] Traina C.; Traina A. J. M.; Seeger B.; Faloutsos C.. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. Proc. Intl. Conf. on Extending Database Technology, Konstanz, Germany, p. 51-65, 2000.

[10] Vespa, T.G., Jr., C.T., Traina, A.J.M.: Bulk-loading Dynamic Metric Access Methods. In SBBD(2007) 160-174.