Estendendo o algoritmo *Partition Based Spatial Merge Join* com partições adaptáveis

MIGUEL RODRIGUES FORNARI[1]
CIRANO IOCHPE[2]
[1,2]UFRGS – Universidade Federal do Rio Grande do Sul, Caixa Postal 15064, 91501-970, Porto Alegre, RS, Brasil
{fornari,ciochpe}@inf.ufrgs.br

Abstract. The spatial join operation matches two sets of geometric descriptions by means of a geometric predicate. This work concentrates in algorithms based in space subdivision, presenting a new algorithm, based on PBSM, called PBSM-NRQB (No-Replication, Quadtree, Bucket). This algorithm uses statistical information, maintained in quadtrees, to adapt the space partitioning to object distribution and uses a file organization, based in buckets to store object descriptors in a more convenient way. The comparison with others algorithms in literature are made by performance analysis and experimental results obtained using synthetic and real data sets, showing that the algorithm is very competitive in all the situations, reducing the number of I/O operations and response time in almost every case.

1 Introdução

A operação de junção espacial, $A\ sp_q\ B$, combina dois conjuntos de feições geométricas, *A* e *B*, baseada no predicado espacial q, sendo essencial e, ao mesmo tempo, uma das mais caras, em bancos de dados geográficos. Pontos, linhas ou polígonos podem representar feições geométricas. O resultado é um conjunto de pares de elementos para os quais o predicado é verdadeiro. O predicado deve ser uma operação espacial binária com resultado booleano [1]. Como exemplo, considere um conjunto de dados descrevendo escolas e outro conjunto descrevendo indústrias químicas. Para responder a questão "Selecione todas escolas localizadas ao lado de uma indústria química" é necessário utilizar uma junção espacial, com o predicado "tocar".

Orenstein [2] sugeriu dividir a junção espacial em duas etapas. Na primeira, chamada filtragem, o predicado é testado para as aproximações de objetos (envelopes), resultando em uma lista de pares candidatos. Nesta etapa, o predicado deve ser adaptado, para garantir que todos os possíveis pares sejam incluídos na lista. Em uma segunda etapa, chamada refinamento, a geometria exata de cada objeto é utilizada nas comparações, considerando-se, apenas, os pares selecionados na etapa anterior.

Este artigo concentra-se em algoritmos que realizam uma subdivisão do espaço para realizar a etapa de filtragem, sem utilizar índices. Um novo algoritmo é apresentado e comparado aos demais algoritmos por eficiência. Este algoritmo, chamado PBSM-NRQB (*Partition Based Spatial Join-No Replication, Quadtree, Bucket*), baseia-se em informação estatística sobre a distribuição de objetos, armazenada em quadtrees, para ajustar a divisão do espaço de acordo com a distribuição dos objetos. Também utiliza uma organização de arquivos *hash* baseada em *buckets* para armazenar descritores de objetos. Além disso, um critério espacial impede a inclusão de pares duplicados no conjunto de resposta da primeira etapa. Todos algoritmos na fase de filtragem podem operar sobre descritores de objetos, compostos pelo identificador único (OID), o envelope (MBR) e um ponteiro para a descrição completa do objeto, armazenada em outro arquivo.

O artigo está organizado da seguinte maneira: na segunda seção, é feita uma revisão da literatura descrevendo os algoritmos de junção por partição já existentes. Na seção três, o PBSM-NRQB é descrito, e, na quarta seção, comparado a os demais, de maneira analítica. A quinta seção descreve os testes realizados, utilizando conjuntos de dados sintéticos e reais. A conclusão é apresentada na sexta seção.

2 Revisão da literatura

Esta seção revisa três algoritmos: *Partition Based Spatial Merge Join* (PBSM), proposto por Patel e DeWitt [3]; o SSBSJ, proposto por Arge et all [4]; e o SHJ, de Lo e Ravishankar [5].

O algoritmo chamado PBSM, em uma primeira etapa, divide o espaço em um conjunto de células, por exemplo, através de uma grade de 32x32 células. Um número mínimo de partições é calculado e cada célula é alocada a uma partição. Uma partição, em geral, abrange várias células não adjacentes. O objetivo é que células de uma região especialmente densa, em objetos, sejam distribuídas entre diferentes partições, para se obter um número semelhante de objetos por partição. A atribuição de uma célula a uma partição pode ser feita por *hash* ou *round-robin*. Os objetos de cada conjunto são copiados para as respectivas células que os interceptam. Na segunda etapa do algoritmo, as partições são, seqüencialmente, trazidas para a memória. Se a partição ocupar um espaço maior que o *buffer* de memória

disponível, ela é dividida. Com os objetos pertencentes à partição em memória, pares de objetos que atendem ao predicado de junção espacial são identificados e colocados em um conjunto temporário de resposta. Como os mesmos objetos podem ser copiados em mais de uma partição, pode ocorrer pares duplicados, o que obriga a ordenar o conjunto de resposta para, finalmente, obter um conjunto de resposta sem duplicatas. O número de operações de E/S (entrada e saída) pode ser estimado pela expressão

$$e_{PBSM} = (2ofr^2 + 2fr + 1)(b_A + b_B) + 2b_{RR} + b_{RS} \qquad (1)$$

Devido às limitações de espaço, o desenvolvimento desta e outras expressões pode ser encontrado em [6,7,8]. A tabela 1 indica o significado dos símbolos empregados ao longo do artigo.

Símbolo	**Significado**
n_A, n_B	Número de objetos nos conjuntos *A* ou *B*
b_A, b_B	Número de blocos dos conjuntos *A* ou *B*
b_{RR}, b_{RS}	Número de blocos do conjunto de resposta com replicação (*RR*) e sem replicação (*RS*)
b_{quad}	Número de blocos de uma quadtree
fr	Fator de replicação
a	Fator de replicação no PBSM_NRQB
b	Fator de replicação no PBSM
f	Fator de replicação no SHJ
h	Altura de uma árvore quadtree
o	Porcentagem de partições com *overflow*
M	Número de objetos que podem ser mantidos no *buffer* de memória

Tabela 1. Significado dos símbolos utilizados.

A idéia básica do algoritmo *Scalable Sweeping–Based Spatial Join* (SSBSJ) é, em uma primeira etapa, ordenar todos objetos por um dos eixos de coordenadas [4], através de um algoritmo de ordenação externa *sort-merge*. Na segunda etapa, os objetos são lidos seqüencialmente para memória e um algoritmo tipo *sweepline* é executado para testar o predicado de junção entre pares de objetos. O número de operações de E/S, pode ser expresso por

$$e_{SBSSJ} = 3(b_A + b_B) + 2(b_A + b_B)\lceil \log_M (b_A + b_B) \rceil + b_{RS} \quad (2)$$

Lo e Ravishankar [5] propuseram o algoritmo *Spatial Hash Join* (SHJ) para realizar a operação de junção espacial. Inicialmente, o algoritmo calcula o número de blocos em disco necessários para armazenar o conjunto *A*, através de um método proposto em [9], que resulta em um número de partições maior que no algoritmo PBSM. A cada partição corresponde somente uma célula. A divisão inicial do espaço em células retangulares é realizada através de uma amostragem dos objetos de ambos conjuntos. Para cada objeto do conjunto *A*, seu centro é calculado e uma célula é escolhida para a inserção pelo critério de menor distância entre o centro da célula e o centro do objeto. Se necessário, os limites da célula são expandidos para incorporar todo o objeto. O número de células se mantem constante. Concluída a alocação do conjunto *A*, procede-se a alocação dos elementos do conjunto *B*. As células já definidas, para *A*, são mantidas para *B*, e cada objeto de *B* é inserido em todas as células que interseciona. Sendo assim, apenas objetos do conjunto *B* são replicados

Na etapa de junção, cada partição é trazida para memória. Os elementos de cada célula são verificados de acordo com o predicado da junção. Se todo conjunto de objetos da partição couber em memória, não são necessárias leituras adicionais. Se, no entanto, o número de objetos for maior que a memória disponível, o algoritmo realiza a junção por laços aninhados. Como o número de partições é maior que no algoritmo PBSM, o número de partições com mais objetos que a memória disponível é menor. Assim, consideramos, de maneira otimista, que não ocorre *overflow* em nenhuma partição. O total de operações de E/S, já prevendo a gravação da lista de respostas, pode ser expresso da seguinte forma

$$e_{SHJ} = 3b_A + (1+2\,fr_B)b_B + b_{RS} \qquad (3)$$

3 PBSM-NRQB

O algoritmo aqui proposto, chamado *PBSM-NRQB* particiona o espaço de maneira adaptada à distribuição dos objetos nos dois conjuntos envolvidos na junção, evita o aparecimento de pares de objetos duplicados no conjunto de resposta e utiliza uma estrutura especial de arquivos, visando reduzir o número de operações de E/S e, consequentemente, o tempo de resposta.

O primeiro acréscimo no algoritmo consiste em definir o particionamento de acordo com a distribuição dos objetos no espaço, utilizando um histograma da distribuição espacial dos mesmos [10], o qual é gerado uma vez para cada conjunto de objetos, e armazenado em uma quadtree, chamada *HistQ*. Inicialmente, é necessário construir a quadtree, que mantem o número total de objetos em cada quadrante, nos vários níveis da árvore. Isto é realizado através de um algoritmo recursivo, que subdivide o espaço em quadrantes, gerando uma quadtree com número de níveis pré-determinado. O nodo raiz representa todo espaço abrangido pelo conjunto de objetos. Na raiz, incrementa-se o respectivo contador de objetos. A seguir, o algoritmo verifica com quais quadrantes o envelope de cada objeto interseciona e realiza uma chamada recursiva para cada quadrante em questão. No nodo correspondente ao quadrante,

incrementa o contador, verifica a interseção com os subquadrantes, até atingir o último nível. O número de níveis da quadtree é um parâmetro que pode ser ajustado pelo administrador da base de dados (DBA), que deve considerar o tamanho médio do *buffer* de memória para operações de junção espacial e o número de objetos do conjunto. A construção da quadtree pode ser realizada no momento da carga do conjunto de objetos no Sistema Gerenciador de Banco de Dados Espacial (SGBDE), para que a mesma seja utilizada sempre que tais objetos participarem de uma junção espacial.

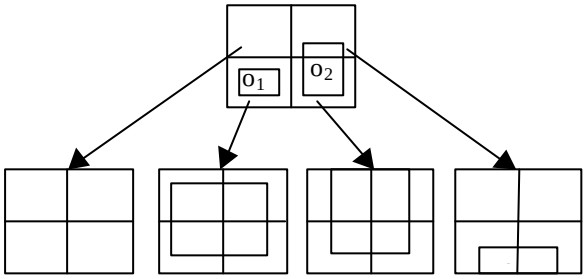


Figura 1. Exemplo de construção de uma quadtree HistQ.

A figura 1 mostra dois objetos, o_1 e o_2, em uma quadtree. O objeto o_1 interseciona apenas um quadrante no nível da raiz. Assim, apenas um chamada recursiva do algoritmo será realizada. O objeto o_2 interseciona dois quadrantes, ocasionando duas chamadas recursivas.

A quadtree HistQ é armazenada de uma maneira linear, através de um vetor de inteiros. Dado um quadrante *p*, seus subquadrantes ocupam as posições *4p-2*, *4p-1*, *4p* e *4p+1* no vetor.

O tamanho de *HistQ* é função da sua altura. Para seis níveis, há 5461 quadrantes, contendo, cada um, um número inteiro (4 bytes). Esta quadtree representa, no nível das folhas, uma grade regular que subdivide o espaço em 64x64 células. O tamanho total é pouco maior que 20Kb. Para 7 níveis, o tamanho é aproximadamente 85Kb, e, para 8 níveis, 277Kb. Para uma altura *h* qualquer, o número de blocos da quadtree é:

$$b_{quad} = \left\lceil \frac{\sum_{i=1}^{h} 4^{(i-1)}\, integersze}{pagesize} \right\rceil$$

O algoritmo de junção inicia carregando, para a memória, a quadtree HistQ de cada conjunto e percorrendo-as, sincronizadamente, da raiz para as folhas, até que a soma de objetos dos respectivos nodos de cada árvore, que representam o mesmo quadrante, seja menor ou igual a capacidade do *buffer* disponível em memória. Isto garante que não haverá *overflow* de partições. Em cada quadtree pode, ainda, ser possível somar dois ou três quadrantes adjacentes, que possuam o mesmo nodo pai, se a soma do número de objetos for menor que o *buffer*. Cada partição é espacialmente delimitada pelos quadrantes correspondentes aos nodos selecionados nas duas árvores. Ao final desta etapa, o conjunto de partições e os seus respectivos limites estão definidos. Esta informação é, também, organizada na forma de uma quadtree, onde cada folha contem o número da respectiva partição. Esta quadtree, chamada PartQ, não é balanceada.

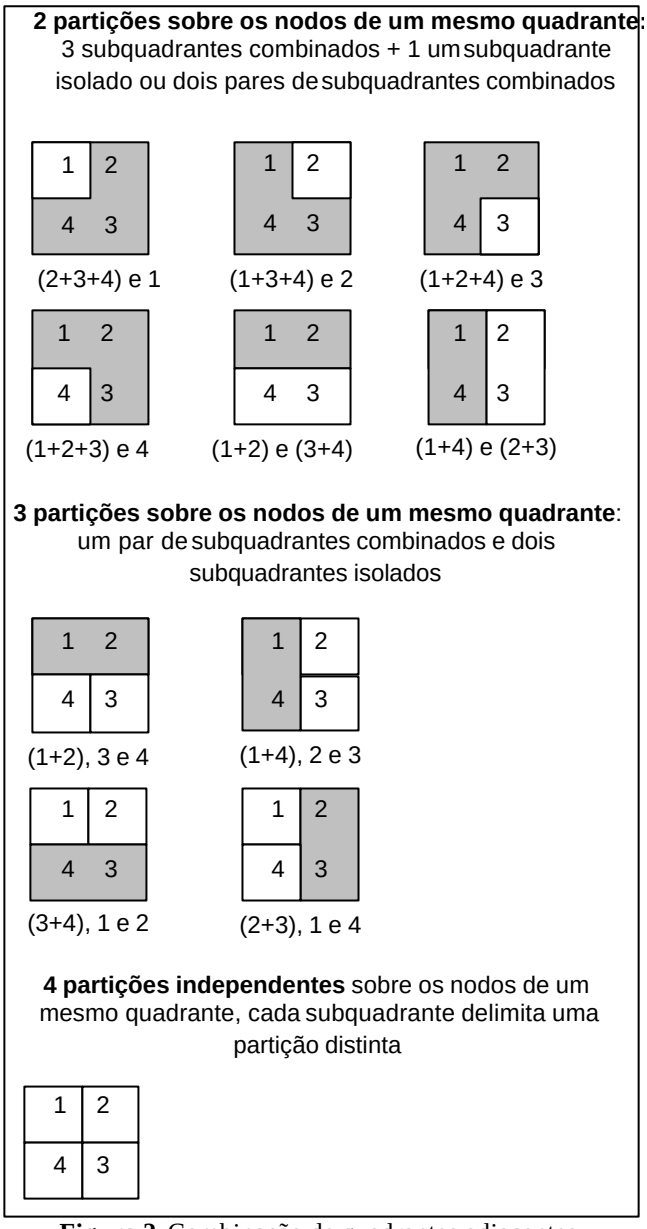


Figura 2. Combinação de quadrantes adjacentes

A figura 2 mostra os casos em que é possível combinar dois ou três quadrantes adjacentes para favorecer o uso de *buffers* maiores, ou seja, a redução do

número de partições. O agrupamento de nodos adjacentes em uma mesma partição evita a replicação de objetos transpassantes. Para os quadrantes não reunidos, talvez seja necessário descer mais um nível da árvore, ou não, dependendo do número de objetos existentes.

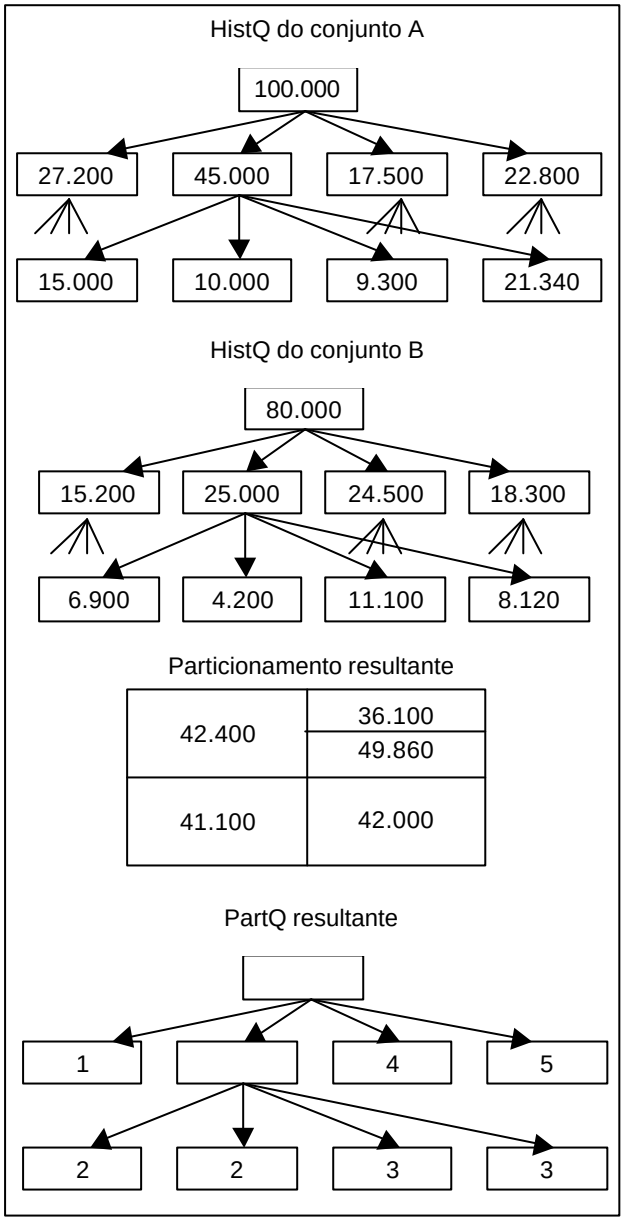


Figura 3. HistQ de dois conjuntos e particionamento do espaço resultante.

Considerando-se, por exemplo, as árvores mostradas na figura 3, e, supondo que o espaço do *buffer* de memória seja suficiente para 50.000 descritores de objetos, teremos a seguinte seqüência: no nível da raiz, a soma das quantidades de objeto ultrapassa este limite (100.000+80.000). Descendo para o nível inferior da árvore, primeiro o algoritmo constata que não é possível qualquer agrupamento de quadrantes adjacentes. Após, verifica as situações individuais. Na folha 1 temos 27.200+15.200, menor que o tamanho do *buffer*, permitindo estabelecer todo este quadrante como uma partição. Na folha 2, temos (45.000+25.000), o que leva a descer mais um nível da árvore para estabelecer as partições. No nível inferior, o algoritmo primeiro tenta reunir combinações de três nodos, mas todas resultam em valores maiores que o limite. Depois, o algoritmo verifica as combinações de dois nodos, encontrando os pares (1,2) e (3,4) como estando dentro do limite. O algoritmo retorna para o nível superior e prossegue testando a situação nos quadrantes 3 e 4. Ainda, na figura 3 pode-se observar a divisão em partições resultante e o número de objetos em cada partição, bem como a quadtree PartQ que identifica cada partição.

Nesta etapa de definição do conjunto de partições, são realizadas operações de E/S apenas para ler as respectivas quadtrees HistQ. Sua expressão de custo é:

$$b_{quadA}+b_{quadB} \qquad (4)$$

Com o objetivo de melhorar o desempenho das operações de junção espacial, mais uma estrutura de armazenamento é proposta. Nesta estrutura, chamada QuadBucket, utiliza-se uma função baseada na HistQ para gerar o endereço do *bucket* onde o descritor de um objeto espacial deve ser gravado. Os descritores, por sua vez, são armazenados em um arquivo *hash*.

O arquivo *hash* de descritores é montado junto com a quadtree HistQ. Sempre que um objeto interseciona um quadrante no nível das folhas da árvore, o número deste quadrante indica o número do *bucket* do arquivo *hash* em que o descritor deve ser incluído. Se um objeto intersecionar mais de um quadrante, haverá duplicatas de seu descritor no nível das folhas e, em conseqüência disso, no arquivo *hash* também.

Cabe observar que uma situação especial pode ocorrer na formação das partições. Quando, mesmo para nodos folha, o número de objetos somados for maior que o tamanho do *buffer* de memória disponível. Como não há informação mais detalhada em níveis inferiores, pois este é o nível das folhas das duas árvores, uma partição é definida. Para tratar esta situação, é necessário verificar o número de objetos em cada partição e, se houver partição com *overflow*, reparticioná-la. O reparticionamento é realizado lendo todos descritores de objetos e redividindo-os em dois ou mais novos arquivos de partição. O número de operações de E/S realizadas, incluindo *overflow* de partições, é dado pela expressão:

$$o(1+fr)fr.(b_A+b_B) \qquad (5)$$

Após definidas as partições, o algoritmo percorre a PartQ, identificando, para cada partição, os *buckets* abrangidos pela partição, o que é realizado com o auxílio de uma das HistQ. A partir daí, o conteúdo destes *buckets* pode ser carregado para memória. Como pode ocorrer de um descritor de objeto ser gravado em dois *buckets* diferentes agrupados em uma mesma partição, é necessário eliminar os objetos duplicados, em memória. Finalmente, pode-se aplicar o predicado de junção para pares de objetos da respectiva partição.

Para evitar pares de objetos duplicados no conjunto de resposta, a técnica chamada *Reference Point Method* [SEE91] é empregada. O par de objetos é incluído no conjunto de resposta somente se o ponto superior esquerdo do objeto pertencente ao conjunto *A* estiver dentro da partição que está sendo considerada. Em caso contrário, o par não é incluído no conjunto de resposta, pois será, novamente, identificado durante o processamento dos mesmos objetos em outra partição. Este critério utiliza-se de um ponto já existente, não necessitando de calculo adicional e é válido para qualquer predicado de junção espacial. O conjunto de respostas é o final, sem replicações de pares de objetos.

A figura 4 ilustra a técnica. A interseção entre os objetos a_1 e b_1 será identificada duas vezes, porque os descritores destes objetos são replicados na partição 1 e na partição 3. O ponto de referência escolhido, superior esquerdo do objeto a_1, pertence à partição 1. Assim, este par de objetos será incluído, no conjunto de respostas, apenas quando a partição 1 for avaliada.

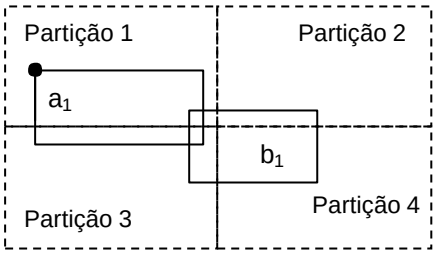


Figura 4 – Critério para inclusão de um par de objetos no conjunto resposta.

O número de operações de E/S, nesta etapa final do algoritmo, pode ser expresso por:

$$ofr^2.(b_A+b_B) + (1\text{-}o)fr.(b_A+b_B) \qquad (6)$$

O algoritmo completo do PBSM-NRQB esta descrito na figura 5. O número total de operações de E/S é calculado somando-se as expressões (4), (5) e (6), e acrescentando-se o custo de gravação do conjunto de resposta, resultando em:

$$b_{quadA}+b_{quadB}+2\,ofr^2.(b_A+b_B) + fr.(b_A+b_B)+b_{RS} \qquad (7)$$

```
Procedure PBSM-NRQB(A,B: conjunto de
objetos espaciais)

quadA, quadB, quadResul : quadtree
lim: MBR
nniveis, numpart: inteiro
c_buckets: conjunto de inteiros
p: partição
b: inteiro
Begin
QuadA = Ler HistQ do arquivo A
QuadB = Ler HistQ do arquivo B
Lim = limites do espaço
nniveis = min( número de níveis de
quadA e quadB)
numpart = 0

quadResul = Define partições

For each p in quadResul do
    If length(p > bufferSize)
       Reparticionar p
    End-if
End-for
For each p in quadResul do
      c_buckets = Identificar buckets
abrangidos pela partição
      For each b in c_buckets
          Ler os descritores inseridos
no bucket b no arquivo A
          Ler os descritores inseridos
no bucket b no arquivo B
      End-For
      Eliminar objetos duplicados
      Aplicar predicado de junção a
pares de objetos
      Gravar os pares de objetos em um
arquivo de resposta, se o par de
objetos atender a condição espacial
End-For
End-Proc
```

Figura 5 – Algoritmo PBSM-NRQB

4. Comparação analítica

Para comparar-se o desempenho do algoritmo PBSM-NRQB frente aos demais, uma análise em termos de número de operações de E/S foi desenvolvida. Em relação ao PBSM original, comparando as expressões (1) e (7), obtem-se:

$$b_{quadA}+b_{quadB} + 2\,ofr^2.(b_A+b_B) + fr.(b_A+b_B)+b_{RS} \leq$$
$$(2ofr^2+2fr+1)(b_A+b_B)+2b_{RR}+b_{RS}$$

Embora os dois algoritmos realizem replicação de descritores de objetos, o método é diferente. Assim, os valores esperados são diferentes. Para diferenciar os símbolos na expressão, o fator de replicação do algoritmo PBSM-NRQB é indicado, em (8), pela letra a, enquanto o fator de replicação do PBSM é indicado por b. Também, o número de blocos das HistQ é muito menor que o número de blocos dos conjuntos *A* e *B*, podendo ser descartados, para simplificar. Reescrevendo a inequação, tem-se:

$$(2oa^2 + a).(b_A + b_B) + b_{RS} \leq (2ob^2 + 2b + 1)(b_A + b_B) + 2b_{RR} + b_{RS} \quad (8)$$

Portanto, se a condição $a \leq 2b + 1$ se verificar, o algoritmo PBSM-NRQB realizará menos operações de E/S que o algoritmo PBSM.

Quanto ao algoritmo SSBSJ, a partir das expressões (2) e (7), pode-se definir a inequação abaixo:

$$b_{quadA} + b_{quadB} + 2ofr^2.(b_A + b_B) + fr.(b_A + b_B) + b_{RS} \leq 3(b_A + b_B) + 2(b_A + b_B)\lceil \log M(b_A + b_B) \rceil + b_{RS}$$

Substituindo o termo $\lceil \log_M (b_A + b_B) \rceil$ por d, chega-se a:

$$(2ofr^2 + fr)(b_A + b_B) \leq 3(b_A + b_B) + 2(b_A + b_B)d \quad (9)$$

Como $d = 1,2,3...$, a inequação (9) representa um conjunto de inequações, uma para cada valor de d.

$$\begin{cases} (2ofr^2 + fr)(b_A + b_B) \leq 5(b_A + b_B) & ,d = 1 \\ (2ofr^2 + fr)(b_A + b_B) \leq 7(b_A + b_B) & ,d = 2 \\ (2ofr^2 + fr)(b_A + b_B) \leq 9(b_A + b_B) & ,d = 3 \\ \quad ... \qquad\qquad ... \end{cases}$$

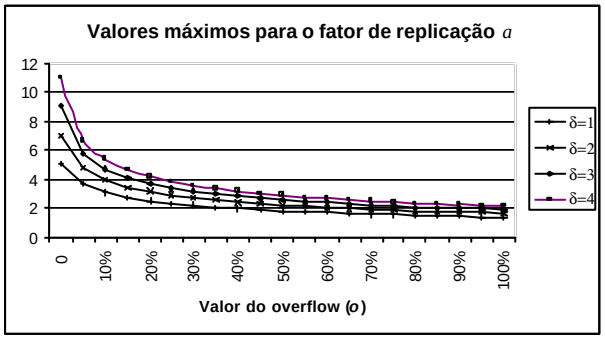


Gráfico 1- Valores máximos para o fator de replicação do algoritmo PBSM-NRQB, em comparação ao algoritmo SSBSJ.

Para cada inequação, é possível obter uma solução numérica para o máximo valor do fator de replicação no algoritmo PBSM-NRQB que garanta um número de operações de E/S menor que o algoritmo SSBSJ. Este valor é função da porcentagem de *overflow*. O gráfico 1 mostra os valores máximos do fator de replicação. Cada curva corresponde a um valor diferente de d.

A comparação com algoritmo SHJ pode ser realizada relacionando as expressões (3) e (7). Novamente, ambos possuem replicação de objetos, por critérios diferentes. Assim, para o PBSM-NRQB é mantida a letra a e para o SHJ, é utilizada a letra f. A partir daí, obtem-se a inequação (10):

$$(2oa^2 + a)(b_A + b_B) + b_{RS} \leq 3b_A + (1 + 2f)b_B + b_{RS} \quad (10)$$

Esta inequação é satisfeita se $p_1 : 2oa^2 + a \leq 3$ e $p_2 : 2oa^2 + a \leq 1 + 2f$. O predicado p_1 é satisfeito se $a \leq \frac{-1 + \sqrt{1 + 24o}}{4o}$. O predicado p_2 é satisfeito se $a \leq \frac{-1 + \sqrt{1 + 8o + 16of}}{4o}$. Se o fator de replicação do algoritmo SHJ (f) for o menor possível, ou seja, igual 1, as duas condições ficam idênticas. O gráfico 2 mostra diferentes curvas para a, trocando o valor de f, em função da porcentagem de *overflow*.

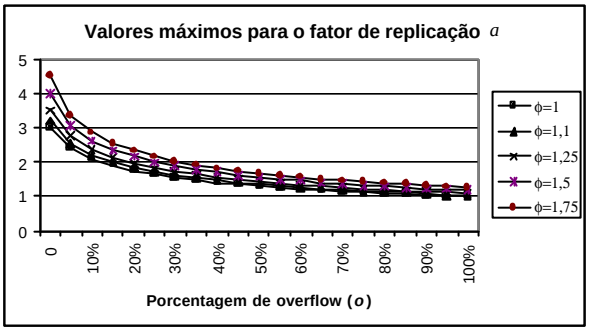


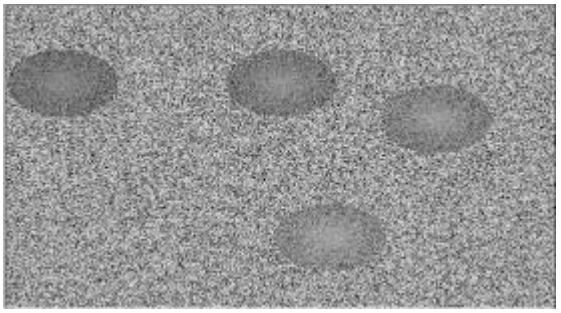
Gráfico 2- Valores máximos para o fator de replicação do algoritmo PBSM-NRQB, em comparação ao algoritmo SHJ.

Se considerarmos que o algoritmo PBSM-NRQB não realiza reparticionamento para uma determinada junção, como feito para o algoritmo SHJ, bastará que $a \leq 1 + 2f$, para que o algoritmo PBSM-NRQB seja mais eficiente que o SHJ.

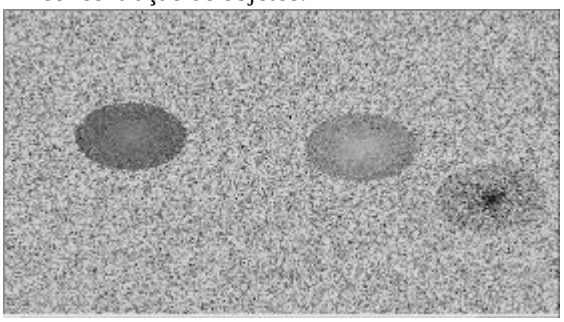
5. Testes realizados

Para comparar o desempenho do algoritmo PBSM-NRQB frente aos demais e comprovar a correção das expressões de número de operações de E/S, também foi realizado um conjunto de testes, utilizando conjuntos de objetos reais e não reais. Estes testes demonstraram que o algoritmo é competitivo em todas as situações, representando melhorias significativas em relação aos três algoritmos. A figura 6 mostra dois conjuntos não reais, o conjunto A com 500.000 objetos, e o conjunto B com 300.000 objetos, com diferentes distribuições no espaço.

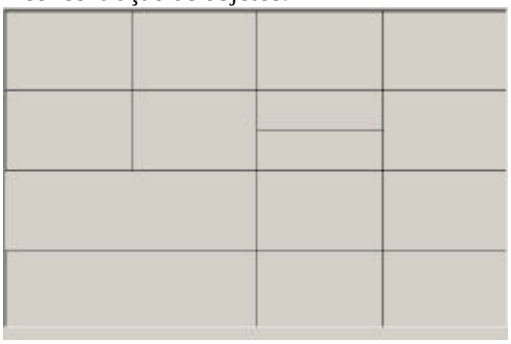
A partição do espaço obtida utilizando um *buffer* de memória com 4Mb resultou em 8 partições, e um *buffer* de memória com 2Mb, resultou em 15 partições, mostrada na figura 6.



Conjunto *A*, 500.000 objetos e 4 pontos de concentração de objetos.



Conjunto *B*, 300.000 objetos e 3 pontos de concentração de objetos.



Partição do espaço para um *buffer* de memória com 2 Mb.

Figura 6. Dois conjuntos não reais e o particionamento obtido pelo algoritmo PBSM-NRQB.

O gráfico 3 mostra o número de operações de E/S realizadas para cada algoritmo, variando o tamanho do *buffer* de memória. O algoritmo PBSM-NRQB é quase constante, variando apenas pela redução do fator de replicação com *buffers* maiores. O algoritmo SBSSJ, em todos estes casos, realizou o mesmo número de operações para ordenar externamente os conjuntos de objetos, mantendo o número total de operações de E/S. Apenas *buffers* muito pequenos ou grandes resultariam um valor diferente. O algoritmo SHJ, por aumentar o número de partições de acordo com o tamanho da memória, aumenta o fator de fragmentação e, consequentemente, o número de operações de E/S.

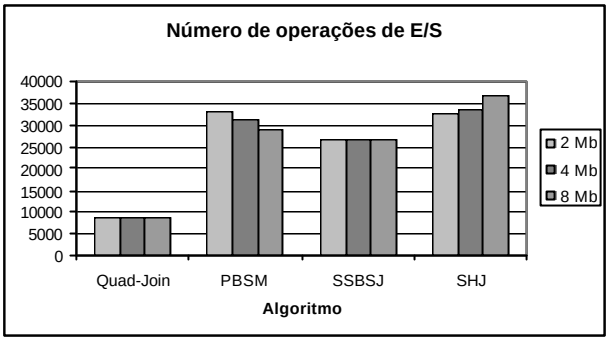


Gráfico 3. Número de operações de E/S, variando o tamanho do *buffer* de memória.

O tempo de resposta do algoritmo também foi inferior em todas as situações. O gráfico 4 mostra o tempo de resposta normalizado de cada algoritmo, para dois conjuntos com 100.000 objetos e um *buffer* de memória de 4 Mb. Este tempo de resposta inclui o tempo de processamento em CPU. O algoritmo SHJ é prejudicado porque a divisão em células não é regular, o que torna a alocação de objetos a uma célula mais difícil que no PBSM. No PBSM-NRQB, esta alocação é realizada percorrendo a PartH, sendo função do número de níveis desta quadtree. Em todos testes, a altura máxima da PartQ foi de 4 níveis, sendo que nesta situação específica foi de apenas 3 níveis. Uma etapa importante é a ordenação dos objetos para realizar o teste do predicado de junção através de *sweep-line*, que é realizada pelos quatro algoritmos.

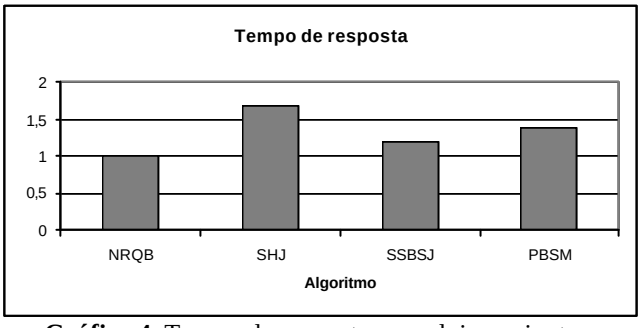


Gráfico 4. Tempo de resposta para dois conjuntos com 100.000 objetos.

Também foram realizados testes com três conjuntos reais: o primeiro é formado por 3.141 polígonos, representando os distritos dos Estados Unidos; o segundo possui 517.538 multilinhas, representando a hidrografia do país; e o terceiro com 166.688 multilinhas, retrata as ferrovias.

A tabela 2 mostra o número de operações de E/S de cada algoritmo, para um *buffer* de memória com 8 Mb.

Sets	NRQB	PBSM	SHJ	SSBSJ
Dist X Hidro	8.509	24.475	27.428	18.360
Ferr X Hidro	10.716	27.363	27.242	23.170
Ferr X Ferr	3.435	12.965	28.914	6.867
Hidro X Hidro	15.406	33.565	58.190	33.490

Tabela 2. Número de operações de E/S para conjuntos reais.

O fator de replicação do algoritmo PBSM-NRQB foi sempre menor. O algoritmo SHJ, por definir um grande número de partições, cujas células se intersecionam, possui o maior fator de fragmentação, sendo o SSBSJ o algoritmo intermediário, embora bem mais alto que o PBSM-NRQB.

6. Conclusão

O algoritmo PBSM-NRQB realiza um número menor de operações de E/S em todas as situações testadas. A adaptação do particionamento à distribuição dos objetos envolvidos na junção é eficiente. A arquivo *hash* reduziu o número de operações de E/S à metade dos demais algoritmos, uma vez que elimina a necessidade de uma leitura e gravação de todo arquivo de objetos para montar arquivos de partição. É necessário realizar apenas a carga dos descritores de objetos para memória, uma única vez.

Um questionamento que surgiu, nos testes, refere-se ao tempo de CPU dos algoritmos, em relação ao tamanho do *buffer*. Como os quatro algoritmos tendem a utilizar ao máximo o *buffer*, carregando para memória um número maior de objetos, conforme o *buffer* cresce, o tempo de CPU cresce proporcionalmente, devido ao tempo de classificação dos objetos em memória. Este fato sugere que *buffers* menores são mais eficientes. No entanto, *buffers* menores tendem a aumentar o fator de replicação, e, consequentemente, o tempo de E/S. Assim, este trabalho seguirá investigando um método para encontrar a melhor relação entre tamanho de *buffer* e fator de fragmentação, de acordo com o algoritmo e cardinalidade dos conjuntos envolvidos na junção.

Também é necessário incluir, nestas comparações, algoritmos baseados em árvores de índice espacial, como, por exemplo, R-Tree, apresentados em [11,12], e explorar as possibilidades de reduzir o tempo de processamento em CPU através de aceleração por hardware, como realizado por Sun et all [13].

Bibliografia

1. RIGAUX, P. SCHOLL, M. VOISARD, A. **Spatial Databases with Applications to GIS** . San Francisco, USA: Morgan Kaufmann Pub., 2000.
2. ORENSTEIN, J. Spatial Query Processing in a Object-Oriented Database System. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings,** p. 326-336, 1996.
3. PATEL, J.M.; DeWITT. D.J. Partition Based Spatail-Merge Join. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings,** p. 259-270., June, 1996.
4. ARGE et al. Scalable Sweeping-Based Spatial Join.In: VLDB CONFERENCE, 24, 1998. **Proceedings,** 1998.
5. LO, M-L.; RAVISHANKAR, C.V. Spatial hash-joins. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings,** p.247-258, June, 1996.
6. FORNARI, M.R., IOCHPE, C. A New Algorithm for Spatial Join Based on Space Subdivision. 11TH SYMPOSIUM OF ADVANCED DATABASE SYSTEMS. **Proceedings**, p. 69-81. June, 2003.
7. KOUDAS, N.; SEVCIK, K.C. Size Sepation Spatial Join. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings**, p. 324-335, May, 1997.
8. KOUDAS, N. **Fast Algorithms for Spatial and Multidimensional Joins.** Tese de Doutorado. Toronto, CA, 1998.
9. LO, M-L.; RAVISHANKAR, C. V. Spatial Joins Using Seeded Trees**.** In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1994. **Proceedings.** ACM SIGMOD Record, New York, v. 23, n.2, p.209-220, 1994.
10. BELUSSI, A.; BERTINO, E.; NUCITA, A. Grid Based Methods for Spatial Join Estimation. 11TH SYMPOSIUM OF ADVANCED DATABASE SYSTEMS. **Proceedings**, p. 49-60. June, 2003.
11. BRINKHOFF, T.; KRIEGEL, H-P.; SEEGER, B. Efficient Processing of Spatial Joins Using R-trees. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings,** p.237-246, 1993.
12. BRINKHOFF, T.; KRIEGEL, H-P.; SEEGER, B. Parallel Processing of Spatial Joins Using R-Trees. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE´96), 12, 1996. . **Proceedings**. p.258-265.
13. SUN, C.; AGRAWAL, D.; ABBADI, A.E. Hardware Accelaration for Spatial Selections and Joins. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. **Proceedings,** p.455-466, June, 2003.