Realização Eficiente de Junções Espaciais Utilizando *Hash-Join*

GERALDO ZIMBRÃO
JANO MOREIRA DE SOUZA

Programa de Engenharia de Sistemas e Computação
Coordenação dos Programas de Pós-graduação em Engenharia
Universidade Federal do Rio de Janeiro
Caixa Postal 68511, Rio de Janeiro, RJ, CEP: 21945-970
email: zimbrao@cos.ufrj.br, jano@cos.ufrj.br

Abstract*: This paper describes the 2DHM (Two Dimensions Hash Matrix), a new algorithm for performing spatial join operation. We apply the hash-join paradigm, that came from relational field, in two phases: the partition phase and the join phase. The difficulties peculiar to spatial hash-joins, like data inflation and dispersion of correlated data across many pages, are handled by means of a hierarchical distribution of overflow buckets, the clustering of pages and pages of different but predetermined sizes. Our method mirrors relational hash-joins in other aspect: it needs no pre-computed indices. So, it is applicable to a wide range of spatial joins. The practical results achieved are good: the number of MBRs comparisons are proportional to the size of answer data sets. This makes the 2DHM method a practical alternative for performing spatial joins.*

Keywords*: spatial join, spatial hash, polygons intersection.*

1. Introdução

Atualmente a área de bancos de dados espaciais vem experimentando um rápido desenvolvimento. Existe uma grande quantidade de sistemas que representam dados que possuem atributos espaciais. Tais sistemas, conhecidos como Sistemas de Informação Geográfica (SIGs), são em geral construídos sobre bancos de dados espaciais. No entanto, bancos de dados espaciais, por serem em geral construídos sobre banco de dados relacionais, carecem ainda de algoritmos especialmente desenvolvidos para as suas necessidades específicas. Como exemplo, podemos citar o atuais esforços que ainda são feitos para criação de um padrão SQL para consultas espaciais [SQL395,SAIF94].

Em geral, tais sistemas permitem a realização de consultas espaciais utilizando-se alguns operadores semelhantes aos da álgebra relacional. Nesse cenário, é de fundamental importância para a avaliação efetiva de consultas um algoritmo eficiente para a realização de junções espaciais. Em [BRIN94], temos a definição de um processador modular de junções espaciais. Tal modelo tem sido muito citado em trabalhos subsequentes, e por isso o trabalho aqui apresentado foi desenvolvido de forma a ser compatível com tal processador.

Nos bancos de dados relacionais, o uso de técnicas de *hash-join* é muito difundido e aceito para a realização de junções, obtendo uma larga vantagem sobre os outros métodos no quesito desempenho. No entanto, a aplicação das técnicas de *hash-join* para a execução de junções espaciais apresenta uma série de dificuldades. Assim, apresentaremos nesse trabalho um novo algoritmo de *hash-join* para a realização de junções espaciais e os resultados já obtidos com o seu uso. Esse trabalho está dividido em seções, descritas a seguir. A seção um é a presente introdução. A seção dois apresenta a definição do problema. Na seção três é feito um apanhado geral de trabalhos na área. Na seção quatro apresentamos o nosso algoritmo em detalhes. Na seção cinco os resultados obtidos são mostrados. Na seção seis temos as

conclusões, onde são mostradas as contribuições do presente trabalho. Na seção sete estão os nossos agradecimentos. Finalmente, na seção oito está relacionada a bibliografia.

2. Definindo o Problema

Atualmente, a pesquisa em bancos de dados espaciais tem tido um lugar de destaque nos esforços de pesquisa da comunidade de banco de dados. Embora um grande número de questões ainda estejam em aberto, alguns requisitos já obtiveram o consenso: por exemplo, um banco de dados espaciais deve oferecer as mesmas funcionalidades para os dados espaciais que um banco de dados relacional oferece aos dados convencionais. Em vista disso, um banco de dados espaciais deve fornecer suporte para responder consultas *ad-hoc* sobre os atributos espaciais armazenados. Por analogia com os bancos de dados relacionais, tais consultas devem ser decompostas em consultas menores, mais simples, que possam ser implementadas através de um conjunto pequeno de operadores espaciais.

Duas das operações mais frequentes nos Sistemas de Informações Geográficas atualmente são as operações de sobreposição e interseção de mapas. Um exemplo de consulta simples que se enquadra em tal categoria é a que nos é apresentada em [VEEN95]: "recupere todas as áreas rurais abaixo do nível do mar que possuam tipo de solo igual a areia e distantes três milhas de lagos poluídos". Com os planos temáticos uso da terra, solo, poluição e elevação, várias junções espaciais terão de ser realizadas para responder a tal consulta. Assim podemos afirmar que tanto nos bancos de dados relacionais como nos bancos de dados espaciais a operação de junção é de grande importância.

Nesse trabalho, objetos espaciais são caracterizados por possuírem pelo menos um atributo que descreva a sua extensão espacial através de uma ou mais linhas poligonais. É importante observar que a única restrição feita quanto à forma de um objeto espacial é que ele seja fechado, podendo no entanto possuir buracos (p.e. lagos) ou ser desconexo (p.e. ilhas).

2.1 Junções Espaciais

Novamente por analogia com a álgebra relacional, entendemos a junção espacial como um subconjunto do produto cartesiano de dois conjuntos, A e B, não necessariamente distintos, contendo respectivamente *m* e *n* elementos. Esse subconjunto é formado pelos elementos do produto cartesiano que atendam a um determinado predicado espacial. De especial interesse para aplicações práticas é a sobreposição de objetos espaciais, de forma que, nesse trabalho, estaremos interessados tão somente na junção de polígonos que se interceptam. No entanto, como é apontado em [BRIN94], os resultados obtidos para esse predicado podem ser facilmente aplicados em outros tipos de predicados.

Uma primeira abordagem para a realização de junções espaciais consiste em utilizar o algoritmo de *loops* aninhados, proveniente das junções em bancos de dados relacionais. Tal algoritmo consiste em confrontar cada elemento do conjunto A com todos os elementos do conjunto B, verificando se o predicado é satisfeito. Este algoritmo simples serve para ilustrar as operações mais custosas na realização de junções espaciais: a transferência de objetos grandes do disco para a memória e o teste de interseção de polígonos. Certamente, é possível diminuir tanto o número de vezes que um objeto será lido como o número de testes de interseção realizados, e é o que nos propomos a fazer aqui.

2.2 Dados de Teste

Nossos dados de testes serão sempre dois conjuntos de polígonos, contendo cada um de 200 até 23.000 polígonos. Utilizamos dados de municipalidades (ou equivalente) de alguns países europeus, condados de alguns estados americanos, províncias da América do Sul e a malha municipal brasileira [IBGE96]. Além disso, alguns conjuntos de dados foram criados

artificialmente conforme proposto em [BRIN94], ou seja, através de um deslocamento aleatório nas coordenadas x e y, e possivelmente uma rotação também aleatória. Tipicamente, a média de pontos por polígono de cada conjunto de dados varia de 22 a 96, conforme a Tabela 1.

Conjunto	# Polígonos	# médio de vértices
Sul dos EUA	496	22
Europa-B	249	96
América do Sul	228	54
Brasil	5081	80
Brasil-B	23007	82

Tabela 1- no de polígonos e vértices em cada conjunto de dados.

Assim, trabalharemos com dois tipos básicos de junções: o conjunto contra ele mesmo e o conjunto contra ele mesmo porém deslocado aleatoriamente de (x,y). Dos conjuntos de dados da Tabela 1, apenas Brasil-B possui polígonos sobrepostos. No entanto, todos possuem MBRs que se interceptam.

2.3 Dificuldades com Junções Espaciais

Técnicas de realização de junções utilizando o paradigma *hash-join* na área relacional são bem estudadas e possuem excelente desempenho [DEWI84], particularmente para relações que são grandes quando comparadas ao tamanho disponível para *buffers*. Contudo, devido à algumas peculiaridades da junção espacial, este método não pode ser diretamente aplicado ao domínio espacial.

As junções realizadas em bancos de dados relacionais são em sua maioria realizadas com um predicado simples: em geral, procura-se tuplas que possuam valores idênticos para um determinado atributo. Conforme apontado em [BRIN94] e [MING96], as dificuldades surgem principalmente por dois motivos.

Primeiro, devido à natureza do predicado da junção espacial: não se busca polígonos exatamente iguais, mas que possuam alguma interseção. Essa diferença é fundamental para a compreensão dos problemas relativos à junção espacial. No caso relacional, como o predicado é de igualdade, sempre que $x=y$ teremos $f(x)=f(y)$, garantindo que as tuplas que possuem o mesmo valor para o atributo escolhido serão mapeadas para o mesmo *bucket*. No entanto, no caso da junção espacial, nem sempre isso ocorre. Sejam p_1 e p_2 dois polígonos que se interceptam, porém tais que $p_1 \neq p_2$ (certamente, esse é o caso padrão): nem sempre será possível garantir que $f(p_1)=f(p_2)$, por melhor que seja a nossa escolha para f, já que $p_1 \neq p_2$. Além disso, a igualdade é uma propriedade transitiva, ao passo que a interseção espacial não. Um estudo mais aprofundado sobre tais fatos pode ser obtido em [MING96].

Segundo, devido ao fato de que os dados espaciais carecerem de uma função de ordenação total que preserve a proximidade espacial. Embora existam técnicas baseadas em curvas de preenchimento do espaço [SAME90,GAED95] que permitam a definição de funções de ordenação total sobre os conjuntos de dados espaciais, tais funções não preservam uniformemente a proximidade espacial, de modo que dois objetos adjacentes no espaço podem ser mapeados bem distantes na ordenação.

3. Trabalhos Correlatos

Recentemente temos observado um certo aumento de trabalhos relacionados com o tema tratado aqui, embora ainda não possamos dizer que haja uma abundância de material publicado nessa área. Conforme apontado, em [BRIN94] é apresentado um processador de consultas espaciais em múltiplos passos (Multi-step Spatial Query Processor - MSQP). Em outros trabalhos do mesmo grupo são feitas comparações entre as aproximações utilizadas

[BRIN93b] e estudos sobre a complexidade de polígonos [BRIN95]. Também em [VEEN95] e [ZIMB97] são apresentadas diferentes aproximações para objetos espaciais. Em [GÜNT93] é apresentado um modelo geral para a estimar o custo de junções espaciais. Já em [DEWI96], [MING95,96], [BERC96] e [BRIN93b,96] são apresentados algoritmos para a seleção dos pares de polígonos candidatos, sendo que em [MING96] temos também uma boa caracterização dos possíveis algoritmos de *hash* espacial. É importante observar que o presente trabalho está plenamente de acordo com o trabalho de [MING96], porém propondo algumas alterações nos algoritmos de forma a minimizar os problemas levantados pelo mesmo.

3.1 O Processador de Junções Espaciais

Em um trabalho freqüentemente citado, [BRIN94] apresenta o processador de consultas espaciais em múltiplos passos (Multi-step Spatial Query Processor - MSQP), projetado para a realização de junções espaciais. Em tal processador, uma junção espacial é dividida em três passos, descritos mais adiante. O processador de junções espaciais definido por [BRIN94] permite uma modularização sem precedentes na realização de junções espaciais. De fato, nosso presente trabalho se ajusta a tal modularização, de forma a poder substituir (com vantagens) um dos módulos descritos no MSQP. Alguns dos outros trabalhos citados podem também ser adaptados para conformarem com o MSQP, implementando alguns de seus passos. A seguir, detalhamos o processador MSQP:

Passo 1 do MSQP: Seleção de Candidatos - Primeiro, ao invés de uma representação exata dos polígonos, o menor retângulo envolvente (Minimum Bounding Rectangle - MBR) é utilizado como uma aproximação para computar uma junção espacial aproximada. Este passo retorna um assim chamado conjunto de candidatos pois contém todos os elementos do conjunto resposta e mais alguns elementos do produto cartesiano que não se interceptam, porém cujos respectivos MBRs se interceptam. *Nosso presente trabalho se refere principalmente a esse passo.*

Passo 2 do MSQP: Aplicação do Filtro - No segundo passo, aproximações mais acuradas que o MBR são utilizadas para refinar a resposta, eliminando alguns dos falsos candidatos. Além disso, respostas positivas também podem ser identificadas através de tais aproximações sem ter acesso à representação geométrica exata dos objetos espaciais. Em particular, em [ZIMB97] é proposta uma aproximação bastante eficiente.

Passo 3 do MSQP: Comparação Geométrica Exata - Finalmente, nesse passo todos os pares candidatos remanescentes são examinados. Este passo requer acesso à representação geométrica exata dos objetos espaciais, frequentemente sendo o passo que mais consome tempo. No entanto, o tempo gasto nesse passo pode ser reduzido através do uso de aproximações melhores no passo anterior.

3.2 *Caracterização de junções Espaciais utilizando* Hash-Join

No já citado trabalho de [MING96], podemos ver uma caracterização das escolhas a serem feitas para a aplicação de técnicas envolvendo *hash-join* às junções espaciais. Faremos aqui um resumo rápido de tais idéias, utilizando termos emprestados da área relacional.

O algoritmo de *hash-join* pode ser dividido em duas fases distintas: a fase de partição e a fase de junção. Na fase de partição, os conjuntos internos e externos são particionados e seus elementos são colocados nos respectivos *buckets*, utilizando-se para isso funções de partição (funções de *hash*). Na fase de junção, os *buckets* correspondentes dos conjuntos internos e externos são examinados, aplicando-se o predicado a cada um dos elementos e gerando-se assim o conjunto resposta.

Os algoritmos utilizados em bancos de dados relacionais têm concordado com duas restrições, uma para cada uma das fases anteriormente citadas. A primeira é a atribuição

simples, significando que cada elemento da entrada é colocado em apenas um *bucket*. A segunda é o casamento simples, significando que cada *bucket* do conjunto interno será comparado com apenas um *bucket* do conjunto externo.

Finalmente, [MING96] aponta que para se adaptar algoritmos de *hash-join* para a realização de junções espaciais, devido ao problema do predicado não ser de igualdade, é necessário que pelo menos uma das duas restrições levantadas seja relaxada. As possíveis escolhas são:

- Atribuições múltiplas: nesse caso, um elemento da entrada pode ser mapeado para mais de um *bucket*.
- Casamentos múltiplos: nesse caso, cada *bucket* do conjunto interno pode necessitar de ser comparado com mais de um *bucket* do conjunto externo.

Cada uma das escolhas acima tem suas desvantagens. Em [MING96] é feita a opção pela segunda alternativa. Nesse presente trabalho, e também em [DEWI96], optou-se pela primeira alternativa, mantendo a segunda restrição intacta.

4. Algoritmo Proposto

Nessa seção, apresentaremos a nossa abordagem, a 2DHM (*Two Dimensions Hash Matrix*), os problemas resultantes da mesma, e justificaremos a não utilização das técnicas já desenvolvidas para os dados pontuais no espaço (*bit-interleaving*) [SAME90]. Além disso, apresentaremos as soluções propostas para tais problemas.

Conforme citado anteriormente, a nossa abordagem prevê que um polígono poderá ser inserido em vários *buckets*. O nº médio de *buckets* em que cada polígono é inserido configurará o **inchaço** dos dados. Claro está que esse inchaço deve ser minimizado, pois ele significa redundância de informação e aumento do espaço utilizado em disco, o que poderá resultar em um acréscimo do nº de operações de entrada e saída.

A idéia principal é dividir o espaço de busca em células de tamanho fixo, formando uma grade regular. Por convenção, cada célula será unicamente identificada pelas coordenadas de seu canto inferior esquerdo, sendo essas coordenadas a chave da célula. Será aplicada a função de *hash* na chave de cada célula que um determinado polígono interceptar, de forma a produzir um conjunto de *buckets* onde tal polígono deve constar. A Figura 1 ilustra esse processo.

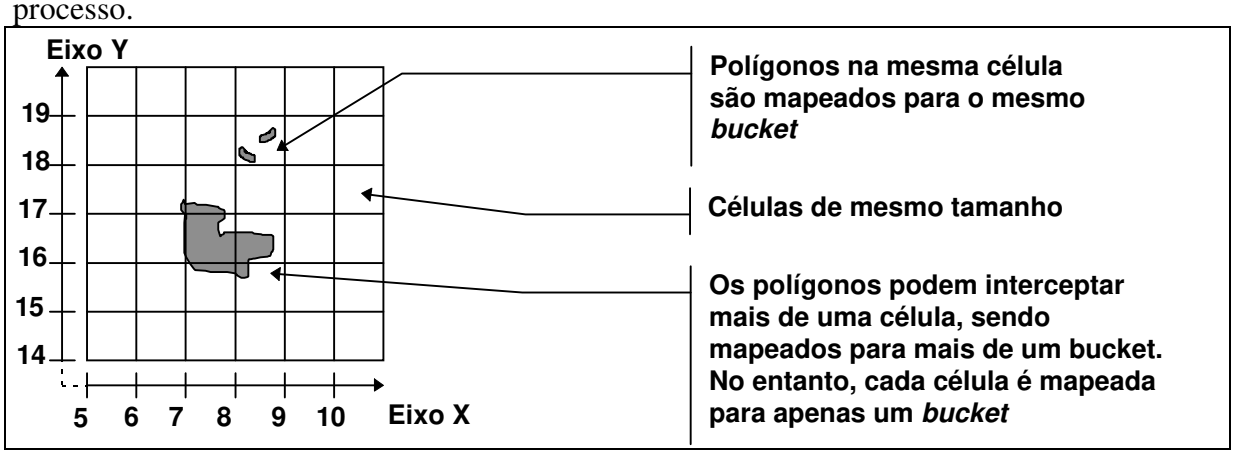

Figura 1 - Mapeamento de um polígono

Finalmente, iremos utilizar aproximações no lugar dos polígonos, mais precisamente a aproximação 4CRS, descrita em [ZIMB97], e os polígonos originais serão armazenados em um arquivo a parte. Basicamente, a aproximação 4CRS é uma figura raster contendo m×n células, cada uma delas com dois bits de informação indicando uma das possibilidades indicadas na Tabela 2.

Tipo de Célula	Descrição
00	A célula não intercepta o polígono
01	A célula contém uma interseção de 50% ou menos com o polígono
10	A célula contém uma interseção de mais de 50% com o polígono
11	A célula está totalmente ocupada pelo polígono

Tabela 2 - Valores para cada célula da aproximação 4CRS

A Figura 2 mostra um exemplo de aproximação 4CRS para um polígono. Essa aproximação possui uma série de vantagens, descritas em [ZIMB97], dentre as quais podemos citar: é compacta, tendo tamanho mínimo e máximo previsível; possui um ótimo desempenho como filtro para o passo 2 do processador MSQP; e podemos utilizá-la para determinar quais células um determinado polígono intercepta sem que seja necessário ter acesso ao polígono original.

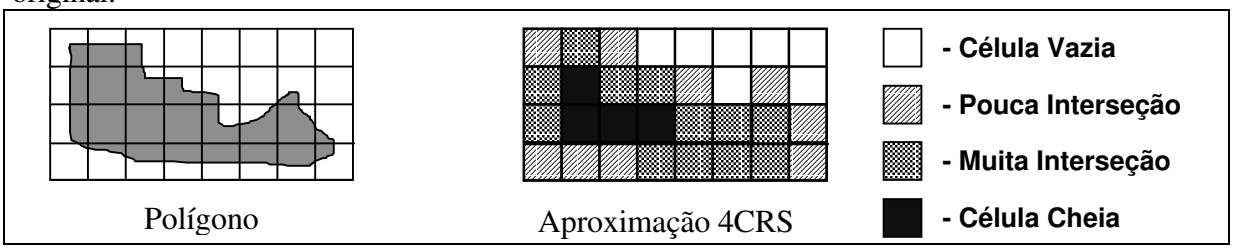

Figura 2 - Aproximação 4CRS de um polígono

Para realizarmos uma comparação entre duas aproximações devemos sobrepô-las e comparar cada célula de uma aproximação com a sua correspondente na outra aproximação. Como é fácil verificar, apenas três casos em dezesseis não permitem conclusão (Pouco×Pouco, Muito×Pouco e Pouco×Muito). Maiores detalhes são encontrados em [ZIMB97].

4.1 Bit-interleaving

A abordagem tradicional para o hash espacial quando os dados são pontos no espaço é realizar *bit-interleaving* nas coordenadas de cada um dos pontos, gerando uma chave unidimensional [SAME90]. Após a criação da chave unidimensional para cada ponto aplica-se na mesma a função de *hash*. Espera-se assim obter uma distribuição homogênea, com poucas colisões, embora haja uma perda da proximidade espacial. Essa abordagem, contudo, não pode ser utilizada diretamente em polígonos, pois embora pontos e polígonos possuam natureza espacial, o predicado mais utilizado para pontos é o de igualdade, sendo portanto muito semelhante ao predicado utilizado para dados adimensionais na abordagem relacional. Já na junção de polígonos o predicado mais comum é a interseção, que, como vimos, possui muitas particularidades.

4.2 Matriz de Hash *Bidimensional*

Para tentar melhorar tal situação, propomos uma forma alternativa de realizar o *hash* espacial. Nossa proposta consiste em não reduzir a informação bidimensional. Preservando a dimensão original podemos preservar também a proximidade espacial. Dessa forma, mapearemos as chaves não para uma tabela de *hash*, mas para uma matriz de *hash*. Por exemplo, tomando a célula de coordenadas x e y, teremos: $(x, y) \mapsto (h_1(x), h_2(y))$, onde h_1 e h_2 são funções simples de *hash*, usualmente envolvendo o resto da divisão por um número primo. Note que utilizando duas funções diferentes a matriz de *hash* não precisa ser quadrada, podendo obedecer à algum critério de proporcionalidade em relação aos conjuntos de entrada. A Figura 3 ilustra o mapeamento de diversas células para a mesma posição na matriz de *hash*.

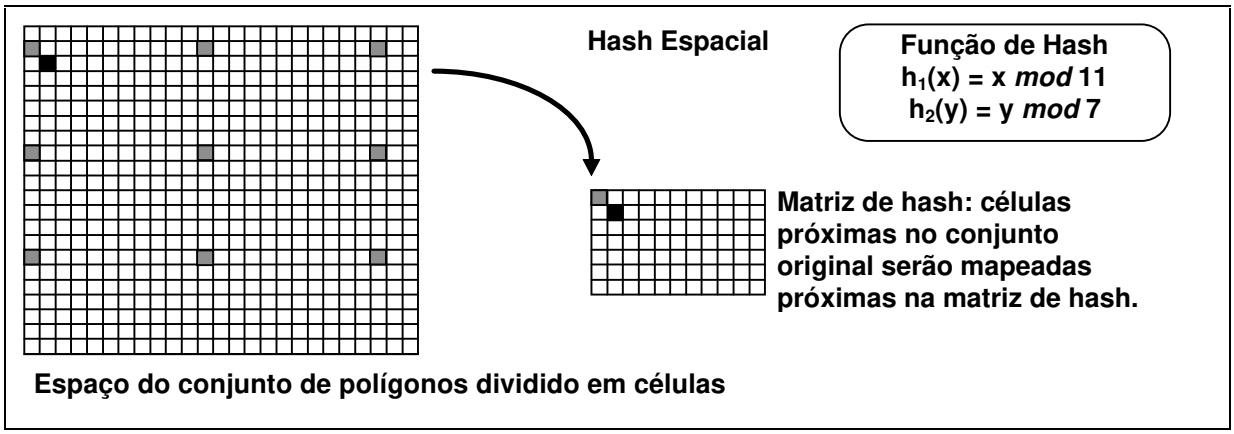

Figura 3: Cada posição da matriz de *hash* será um *bucket* contendo várias células.

4.3 Tamanho das Células

Ao utilizar tal abordagem diretamente verificamos que o inchaço nos dados apontado em [MING96,DEWI96] foi muito grande, tornando o algoritmo muito ineficiente tanto do ponto de vista do uso de espaço em disco (e consequentemente, em operações de entrada e saída) quanto do número de comparações. O problema pode ser caracterizado da seguinte forma [GAED95]: se utilizarmos células pequenas quando comparadas ao tamanho médio dos polígonos, cada célula no espaço de busca será interceptada por poucos polígonos, embora cada polígono intercepte várias células inchando os dados. Por outro lado, se utilizarmos células grandes, teremos um inchaço menor nos dados, mas em compensação cada célula conterá mais polígonos. A Figura 4 ilustra essa situação.

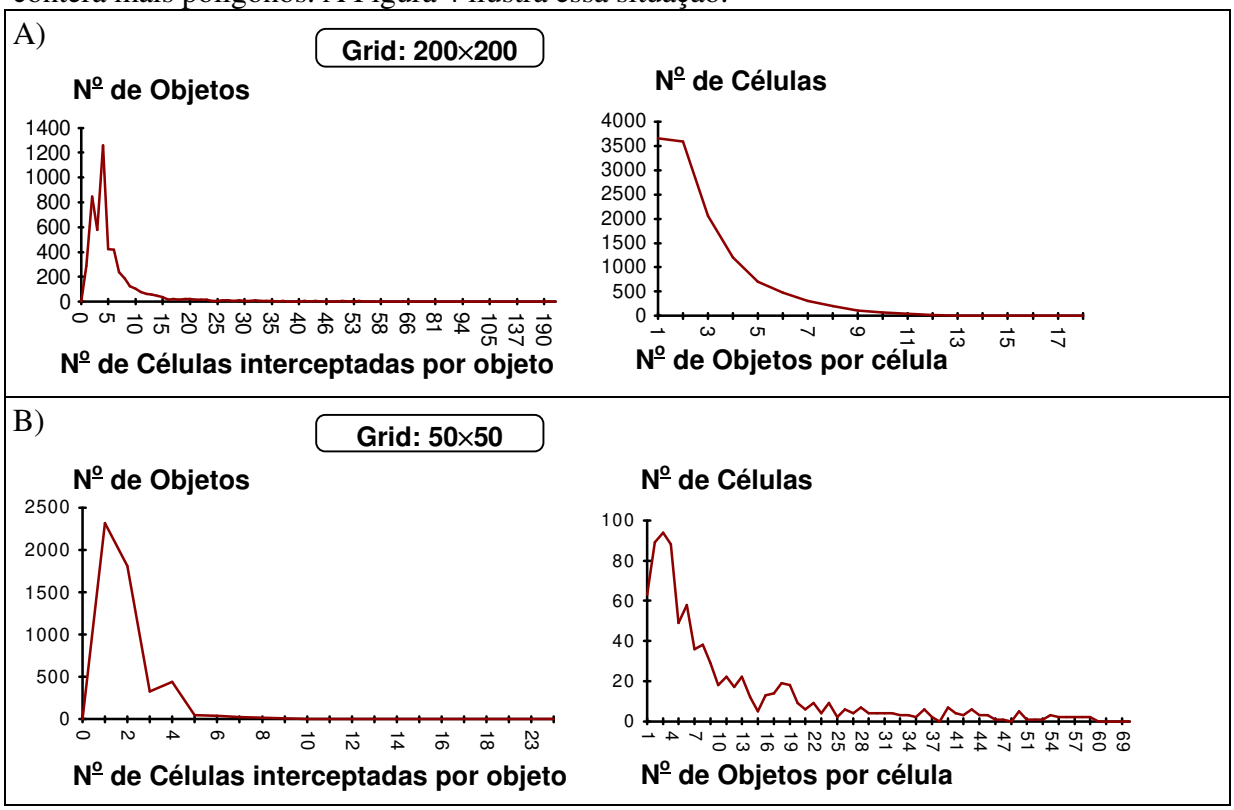

Figura 4 - Distribuição das células e objetos para o conjunto de dados Brasil.

Um meio termo deve ser encontrado, porém para tanto, em geral é necessário ter um grande conhecimento prévio sobre os dados. Conforme podemos ver na Figura 4-A, ao utilizarmos células pequenas temos um grande número de células (mais de 3500) interceptando

apenas um polígono, porém teremos polígonos interceptando mais de 190 células. Já na Figura 4-B, ao utilizarmos células maiores, teremos poucas células interceptando apenas um polígono (em torno de 60), porém apenas um objeto intercepta mais de 23 células.

Em outras palavras, se tivermos muitas células, várias delas serão mapeadas para o mesmo *bucket*, aumentando consideravelmente a ocupação dos mesmos. Além disso, como cada polígono interceptará muitas células, o resultado é um aumento enorme no número de comparações redundantes, ou seja, o mesmo par de MBRs será investigado mais de uma vez. Por outro lado, se utilizarmos células grandes, teremos um grande número de polígonos interceptando a mesma célula, e como consequência, será preciso um grande número de comparações para descobrirmos os MBRs que se interceptam. É necessário portanto uma escolha criteriosa para obtermos um tamanho de célula eficiente. A Tabela 3 ilustra esse fato para alguns de nossos conjuntos de dados, supondo uma junção do conjunto com ele mesmo. Vale observar que esse é um dos testes de pior caso, pois se houver uma área com grande concentração de polígonos, ela o será nos dois conjuntos. Como estimativa otimista, assumiremos uma distribuição uniforme dos polígonos pelas células. Embora pela Figura 4 possamos constatar que isso não corresponde à verdade, essa estimativa representa o menor nº possível de comparações para determinado *grid*, de forma que teremos um parâmetro para avaliar o quão boa é uma determinada abordagem. Se α é o nº de células ocupadas, η_i a ocupação da célula *i* e μ é a ocupação média das células (contando apenas as células ocupadas), o número estimado de comparações é $\alpha.\mu.(\mu-1)/2$, e o número real de comparações é $\Sigma(\eta_i-1)^2/2$. Convém lembrar que não é necessário comparar o polígono contra ele mesmo.

Tamanho do *Grid*		**Sul dos EUA**	**Europa-B**	**América do Sul**	**Brasil**	**Brasil-B**
200 x 200	Ocupação média	1,47...	3,60...	1,38...	2,78...	4,01
	# comp. estimada	4.247	56.785	2.355	30.817	194.155
	# comp. real	6.928	108.202	4.021	55.501	365.451
	Inchaço dos MBRs	36,19×	175,36×	54,68×	6,82×	5,60×
	Inchaço dos dados	12,73×	59,12×	18,89×	2,94×	2,53×
100 x 100	Ocupação média	2,00...	4,00...	1,72...	5,16...	6,85...
	# comp. estimada	3.129	18.780	1.460	34.679	190.382
	# comp. real	4.440	36.610	2.411	71.588	375.074
	Inchaço dos MBRs	12,62×	50,25×	17.70×	3,28×	2,83×
	Inchaço dos dados	4,87×	17,42×	6,57×	1,76×	1,61×
50 x 50	Ocupação média	3,18...	4,88...	2,51...	11,77...	14,83...
	# comp. estimada	2.881	8.055	1.181	54.253	286.246
	# comp. real	3.594	16.656	1.968	130.534	606.312
	Inchaço dos MBRs	5,31×	16.65×	6,85×	1,98×	1,80×
	Inchaço dos dados	2,44×	6.22×	2,95×	1,33×	1,27×
25 x 25	Ocupação média	6,07...	6,85...	4,36...	30,51...	39,43...
	# comp. estimada	3.509	4.710	1.260	108.525	606.079
	# comp. real	4.347	10.391	2.163	293.322	1.406.822
	Inchaço dos MBRs	2,79×	6,47×	3,29×	1,45×	1.37×
	Inchaço dos dados	1,60×	2,82×	1,76×	1,15×	1,12×
13 x 13	Ocupação média	14,00...	11,60...	8,28...	88,23...	123,25...
	# comp. estimada	5.733	4.183	1.599	269.362	1.657.404
	# comp. real	7.147	9.473	2.828	747.141	4.069.951
	Inchaço dos MBRs	1,78×	3,17×	1,93×	1,22×	1,18×
	Inchaço dos dados	1,26×	1,72×	1,31×	1,07×	1,06×
MBRs que se interceptam		1.475	3.201	722	18.935	130.140

Tabela 3 - Nº de comparações em função do tamanho das células do *grid*.

O inchaço nos dados não tem um correspondente aumento no espaço utilizado, pois a abordagem 4CRS permite que apenas parte da chave seja armazenada em uma página. Podemos dividir uma chave em duas partes: a parte que contém as informações genéricas e a parte que contém a aproximação. Será necessário replicar apenas a parte genérica da chave, sendo a parte que contém a aproximação dividida entre as várias células. Finalmente, a compactação será pouco afetada por poder ser realizada de forma independente nas partes da aproximação [ZIMB97]. Tipicamente, uma aproximação 4CRS de 750 pontos ocupa 40 bytes. Incluindo o MBR e Identificador do polígono, teremos uma chave de 60 bytes. Portanto, apenas um terço da chave será duplicada para cada polígono que interceptar mais de uma célula, o que significa dizer que um fator 1,45× na realidade terá um efeito multiplicador de 1,15× sobre os dados.

4.4 Critérios para Escolha do Tamanho da Célula

Nessa seção mostraremos os principais problemas relativos a abordagem *hash-join* para a realização de junções espaciais, bem como a solução proposta.

4.4.1 Complexidade

O ótimo desempenho atingido pelo *hash* nas abordagens relacionais pode ser explicado como se segue. Suponha que os conjuntos de entrada possuam cardinalidade *m* e *n*. Então, utilizando a técnica dos *loops* aninhados, devemos realizar n×m comparações, certamente um tanto o quanto ineficiente. A complexidade pode ser estimada como $O(n^2)$: suponha *n* o menor dos conjuntos, então $m = n + b$, para algum $b > 0$. Como são necessárias $m \times n$ multiplicações, teremos: $(n + b) \times n \Rightarrow n^2 + b \times n \Rightarrow O(n^2)$.

Porém, se particionarmos os conjuntos de entrada em *p* partições (células) de forma a termos de realizar comparações apenas entre as partições correspondentes a um e a outro conjunto, o número de comparações pode ser estimado como (supondo-se que as partições de cada conjunto contenham o mesmo número de elementos *n*/*p* e *m*/*p*, respectivamente):

$$p \cdot \frac{n}{p} \cdot \frac{(n+b)}{p} \Rightarrow \frac{n^2 + b \cdot n}{p}$$

Finalmente, se escolhermos p como sendo proporcional a n, podemos ter complexidade $O(n)$:

$$p = n/c,$$

$$\frac{n}{c} \cdot \frac{n}{n/c} \cdot \frac{n+b}{n/c} \Rightarrow \frac{n}{c} \cdot c \cdot \frac{c \cdot (n+b)}{n} \Rightarrow c \cdot (n+b)$$

Na prática, contudo, como dificilmente as partições terão o mesmo número de elementos, o desempenho será um pouco pior. Além disso, se cada *bucket* puder conter mais de uma partição, possivelmente teremos um pequeno aumento no número total de comparações. Devemos lembrar também que o número de comparações terá que depender da cardinalidade do conjunto resposta - principalmente porque uma grande quantidade de interseções implicará também em muitas colisões nas partições. Finalmente, como os dados iniciais foram inchados por um determinado fator (ver Tabela 3), não estamos mais lidando com *n*, mas sim com *n* vezes esse fator.

4.4.2 Tamanho de Célula Variável

Desses resultados segue que devemos escolher o tamanho da lateral da célula de forma a termos poucos objetos na mesma - sem contudo inflar muito os dados. Ora, então o tamanho ideal de célula dependerá de dois parâmetros: o tamanho dos polígonos e sua

distribuição pelo espaço. Como em dados reais ambos os parâmetros variam bastante dentro do mesmo conjunto de dados, o tamanho das células deveria ser variável. É importante notar porém que se houver uma grande concentração de polígonos na mesma área devemos esperar uma grande quantidade de MBRs se interceptando, o que resultará em um grande número de testes inevitavelmente. Por isso escolheremos o tamanho da célula dando maior importância ao tamanho dos polígonos do que à sua distribuição espacial.

Por outro lado, se o tamanho das células for variável, ele será definido por uma função *f(x,y)* onde *x* e *y* são as coordenadas base de cada célula. O problema com esta abordagem é que a função *f* deveria ser específica para cada conjunto de dados. No entanto, se assim procedermos não poderemos realizar a comparação entre os conjuntos interno e externo da junção de forma simples, pois não existirão partições correspondentes. Na realidade, estaremos relaxando também a restrição de "casamentos simples". Portanto, *f* deve ser escolhida levando-se em conta cada par de conjuntos interno e externo, o que aumenta consideravelmente a sua complexidade - note que, nesse caso, ela não pode ser precalculada e armazenada. Em [MING96], abordagens utilizando partições com células de tamanho variável são propostas. A função *f*, que determina a partição, é definida através de R-Trees [GUTT84, BECK90]. Já em [DEWI96], tamanhos de células fixo são utilizados - bastante semelhante ao particionamento aqui proposto.

Nossa proposta é utilizar uma função de partição que dependa exclusivamente do tamanho médio dos polígonos de determinado conjunto, sendo auto-adaptável para regiões em que o tamanho seja muito distante da média ou haja grande concentração de polígonos. Para isso, ao invés de utilizarmos um tamanho de célula variável, utilizaremos três tamanhos de célula fixos, predeterminados para o conjunto de dados a partir do tamanho médio de seus polígonos. Em ordem crescente de tamanho, classificaremos as células em tipo 0, tipo 1 e tipo 2. Além disso, os tamanhos das células devem ser múltiplos uns dos outros, e cada célula deve ter lado 2^n para algum $n > 0$, e coordenadas $(x.2^n, y.2^n)$. Assim, sendo os tamanhos das células tipo 0, tipo 1 e tipo 2 respectivamente t_0, t_1 e t_2, teremos $t_2 = 2^w.t_1$ e $t_1 = 2^v.t_0$, conforme a figura seguinte.

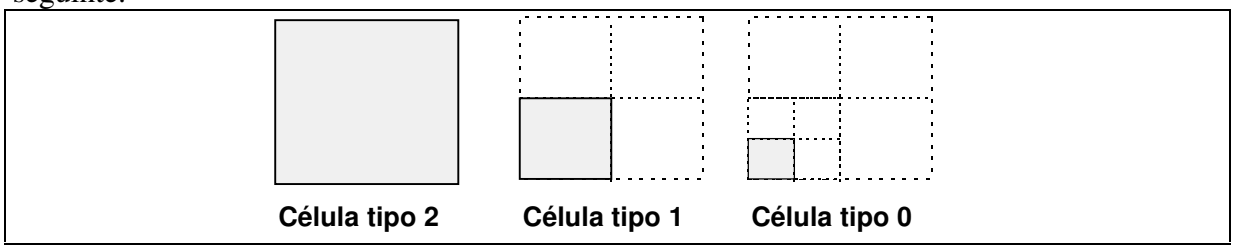

Figura 5 - Tipos de células

Apenas as células tipo 2 serão mapeadas para a matriz de *hash* baseadas na sua própria coordenada base. As células tipo 1 e tipo 0 utilizarão a mesma coordenada base da célula tipo 2 que as contém, e consequentemente serão mapeadas para a mesma partição e para o mesmo *bucket*. Dessa forma, um polígono que intercepte uma célula tipo 2, algumas células tipo 1 e várias células tipo 0 não será replicado no *bucket* correspondente: haverá apenas uma entrada referente ao mesmo. Com isso é possível utilizar uma grade de células pequenas (grade fina) sem causar uma explosão de dados desnecessários, replicados inutilmente.

Em um *bucket*, se houver mais de uma célula tipo 2, elas estarão ordenadas segundo a coordenada base e organizadas hierarquicamente segundo o tipo, conforme a Figura 6. Dessa forma, durante a comparação entre *buckets* correspondentes, apenas os MBRs contidos nas células correspondentes serão testadas.

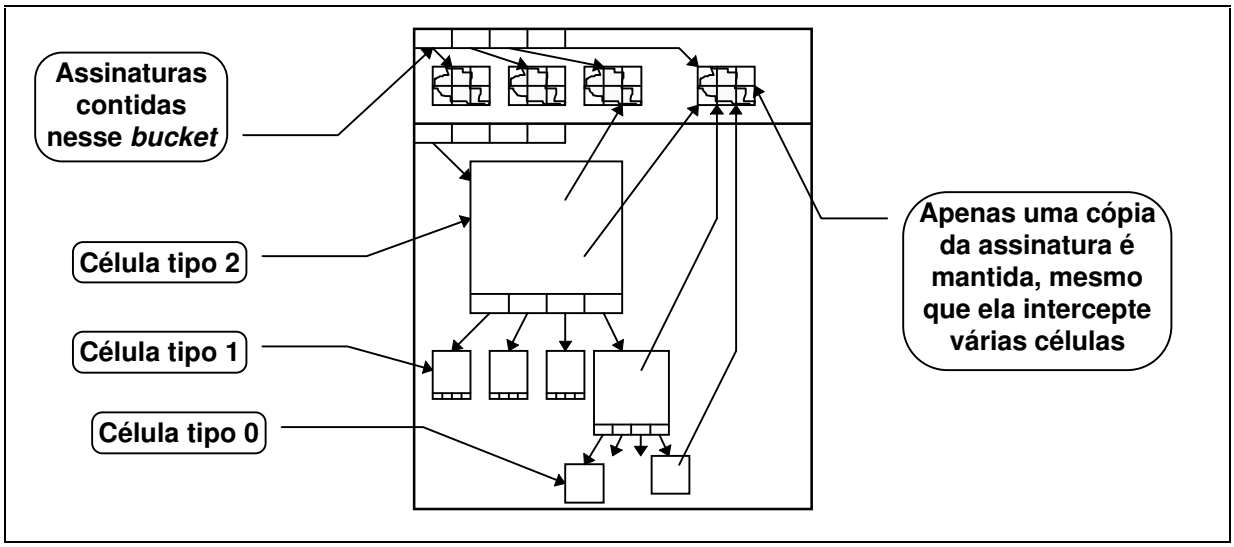

Figura 6 - Estrutura interna de um *bucket*.

O algoritmo escolherá a opção de comparação entre células baseado na distribuição de polígonos pelas células tipo 0, 1 e 2, preferindo sempre o caso ótimo. A Tabela 4 ilustra essas possibilidades, relativas à Figura 7. Convém notar que se uma célula contém 4 polígonos, seriam necessárias 4×3 comparações para se descobrir o nº de MBRs que se interceptam, já que se trata da junção de um conjunto com ele mesmo.

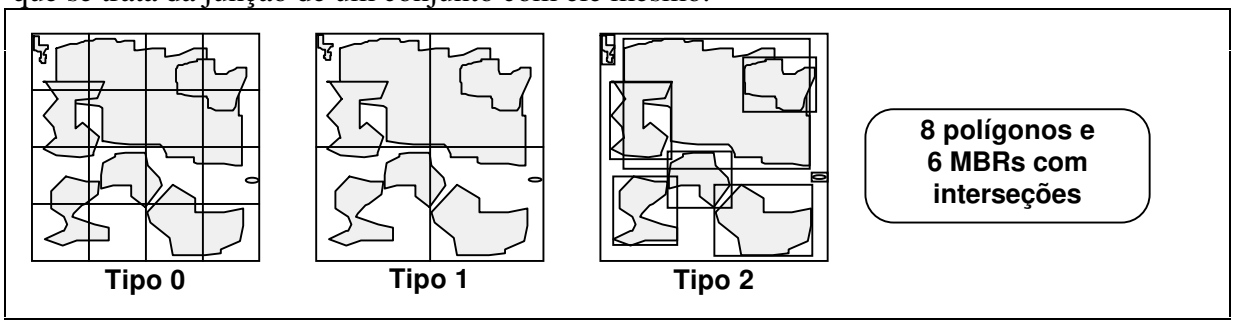

Figura 7 - Ocupação das células segundo o tipo.

Tipo 0	**Célula (x,y)**	0,0	1,0	0,1	1,1	2,0	3,0	2,1	3,1	0,2	1,2	0,3	1,3	2,2	3,2	2,3	3,3	**Total**
	#Polígonos	1	1	2	2	1	1	3	2	2	2	3	3	2	2	2	2	
	#Comparações	0	0	1	1	0	0	3	1	1	1	3	3	1	1	1	1	18
	Total Parcial	2				4				8				4				18
Tipo 1	**Célula (x,y)**	0,0				1,0				0,1				1,1				
	#Polígonos	3				4				3				2				
	#Comparações	3				6				3				1				13
Tipo 2	**Célula (x,y)**	0,0																
	#Polígonos	8																
	#Comparações	28																28
Misto	**Célula (x,y)**	0,0				0,1				1,0				1,1				
	Total Ótimo	2 (Tipo 0)				4 (Tipo 0)				3 (Tipo 1)				1 (Tipo 1)				10

Tabela 4 - Possibilidades de comparação para descobrir o nº de MBRs que se interceptam.

É possível realizar comparações mistas a nível de células tipo 0, 1 ou 2. Sabendo-se quantos polígonos há em cada tipo de célula teremos a opção de escolher a combinação que resulte no menor número possível de comparações. Assim, ao invés de realizarmos 3 comparações para a célula tipo-1 (0,0), realizamos apenas duas comparações nas células tipo-0 correspondentes: (0,0), (1,0), (0,1) e (1,1). Com isso, ao invés das 28 comparações necessárias na célula tipo-2 realizaremos apenas 10. Há casos em que o contrário ocorrerá, ou seja, será

mais vantajoso realizar a comparação utilizando a célula tipo 2 - dependerá justamente do tamanho dos polígonos e de sua distribuição. Em [DEWI96], algoritmos de *plane-sweep* são utilizados no que seria o equivalente a nossa célula tipo-2. Como tal algoritmo depende do nº de polígonos, nada impede que o mesmo seja utilizado após a definição da estratégia de comparações a ser realizada, resultando em um menor número de comparações de MBRs do que o obtido naquele trabalho. Contudo, no momento tal algoritmo não está sendo utilizado.

4.5 *Eliminação de Duplicatas*

Se a área de interseção de dois MBRs interceptar mais de uma célula, isso significa que o par de MBRs será avaliado e incluído na resposta várias vezes. Esse fato não é desejável, pois acarretaria em trabalho redundante nos passos seguintes do processador de consultas além de aumentar sem necessidade o tamanho do conjunto resposta, podendo inclusive invalidar as consultas que pedem a cardinalidade do conjunto resposta.

Em [GAED95] vemos uma solução simples para esse problema: antes de incluir um par de MBRs no conjunto resposta, uma consulta é feita à uma tabela de *hash* que conterá todos os pares já enumerados. Claramente, essa abordagem introduz um novo complicador: manter uma tabela de *hash*, com o tamanho proporcional ao conjunto resposta (inicialmente, esse tamanho é desconhecido). Além disso, certamente essa tabela auxiliar não residirá na memória principal, o que resultará em um grande aumento do número de acessos a disco.

Já em [DEWI96], tal trabalho é postergado para os passos seguintes do processador de junções espaciais - às custas, porém, de uma ordenação de todo o conjunto de saída. Cabem aqui as mesmas considerações sobre memória e disco já feitas no parágrafo anterior, com o agravante de que o melhor algoritmo de ordenação possui complexidade $O(n.\log n)$.

Nossa abordagem, contudo, facilita que se encontre e elimine da resposta os pares de MBRs duplicados: como o espaço de busca está dividido segundo um *grid* uniforme, apenas as células vizinhas àquela que está sendo processada poderão conter pares de MBRs duplicados. Aproveitaremos aqui a proximidade espacial mantida pela matriz de *hash*. Manteremos várias tabelas de *hash*, uma para cada *bucket*, contendo somente os pares de MBRs que não estão inteiramente contidos na célula (e não o conjunto resposta inteiro), e iremos eliminando aquelas tabelas cujas células vizinhas já foram todas processadas. Essa alternativa foi testada com sucesso, e em nossos conjuntos de dados as tabelas de *hash* geradas foram pequenas o suficiente para permanecerem na memória, mesmo no caso do maior conjunto de dados.

4.6 Tratamento do Overflow

Embora o tamanho dos *buckets* possa ser dimensionado de forma a não haver nunca *overflow* em um *bucket*, em geral essa abordagem não é satisfatória por desperdiçar muito espaço, aumentando sobremaneira o acesso ao disco. Portanto, certamente haverá *overflows* em alguns *buckets*. Nossa abordagem trata o *overflow* de uma forma hierárquica, semelhante a uma árvore: cada grupo predeterminado de *buckets* terá um *bucket* extra para *overflow*. Por sua vez, cada grupo de *buckets* de *overflow* terá também um *bucket* extra para *overflow*. A vantagem de tal abordagem é que, na fase de junção, como os *buckets* são percorridos em uma ordem predeterminada (e que deve ser a mesma utilizada para a criação dos *buckets* de *overflow*), quando for lido pela primeira vez o *bucket* de *overflow* do grupo devemos mantê-lo na memória. Dessa forma, é possível evitar acessos extras ao disco quando formos processar os *buckets* restantes do grupo. Novamente, podemos fazer uso da proximidade espacial mantida pela matriz de *hash* para determinar a ordem de leitura dos *bucekts*.

Em nossos testes, utilizando-se um *bucket* de *overflow* para cada grupo de 16 *buckets*, obtivemos uma hierarquia de apenas dois níveis (os outros níveis não chegaram a ser utilizados). É importante observar que o *bucket* de *overflow* apenas agrupa as células que não

couberam no *bucket* original em um bloco para a transferência do disco para a memória. As células em si continuam separadas na estrutura interna do *bucket*, não sendo pois necessário comparar cada um das células do *bucket* de *overflow*, mas apenas as células que originalmente vieram do *bucket* sendo processado.

5. Resultados Obtidos

A Tabela 5 ilustra os resultados obtidos para o conjunto de dados "Brasil", que possui 5.081 polígonos e 18.935 pares de MBRs com interseção. Já a Tabela 6 ilustra os resultados obtidos para o conjunto de dados "Brasil-B", que possui 23.007 polígonos e 130.140 pares de MBRs com interseção. Repare que a coluna "Tipo 2" reflete o desempenho do algoritmo apresentado em [DEWI96]. Finalmente, vale dizer que o nº de comparações se manteve razoavelmente estável para uma faixa grande de valores para o *grid*.

Grid	Número de Comparações			Razão sobre a resposta	Escolha das Células		
	Estimado	Tipo 2	Otimizado		Tipo 2	Tipo 1	Tipo 0
200×200	30.817	55.501	48.750	2,57×	51.60%	29.51%	18.88%
100×100	34.679	71.588	48.257	2,55×	49.11%	31.75%	19.14%
50×50	54.253	130.534	51.204	2,70×	38.48%	29.60%	31.91%
25×25	108.525	293.322	70.731	3,74×	25.74%	25.08%	49.17%

Tabela 5 - Conjunto Brasil: nº de comparações de MBRs e a razão sobre o tamanho do conjunto resposta (quanto menor, melhor) para vários *grids*.

Grid	Número de Comparações			Razão sobre a resposta	Escolha das Células		
	Estimado	Tipo 2	Otimizado		Tipo 2	Tipo 1	Tipo 0
300×300	194.155	365.451	346.113	2,66×	49.85%	27.75%	22.40%
150×150	190.382	375.074	312.133	2,40×	37.67%	36.45%	25.88%
75×75	286.246	606.312	323.367	2,70×	21.14%	33.98%	44.88%
38×38	606.079	1.406.822	378.066	2,91×	18.63%	15.48%	65.89%

Tabela 6 - Conjunto Brasil-B, idem ao anterior.

A Tabela 7 a seguir mostra os custos de entrada e saída de alguns conjuntos. É importante observar que, no conjunto Europa-B, gerado com distribuição espacial aleatória, não existe nenhum tipo de ordenação. Ou seja, dados sequenciais no disco estão completamente dispersos no espaço. Nos outros conjuntos, existe um agrupamento simples, que pode ser por região (América do Sul) como por latitude (Brasil). De uma forma geral, os dados reais possuem uma certa localidade.

Conjunto de dados	América do Sul	Europa-B	Brasil	Brasil	Brasil
Grid	25×25	25×25	200×200	200×200	100×100
tamanho do buffer (em relação ao tamanho dos dados)	9.30%	6,81%	8.22%	2.05%	1.98%
# acessos por página	1,66	4,62	1,04	1,41	2,19
# acessos a disco - 0.5kb	287	1.086	13.022	17.628	7.073 (1kb)
% acertos do buffer	61,73%	32,55%	62,41%	49.11%	57.53%

Tabela 7 - Custos de entrada e saída em blocos de 0.5 Kbyte (a última coluna mostra blocos de 1Kbyte).

Podemos analisar o número de acessos a disco em dois momentos distintos: durante a fase de partição e durante a fase de junção. Durante a fase de partição, o número de acessos a disco é enormemente afetado pela ordem em que os dados dos conjuntos estão dispostos. Se esses dados estão distribuídos aleatoriamente pelo espaço de busca, teremos em consequência um grande número de acessos a disco, pouco adiantando o tamanho do *buffer* utilizado. Porém, conforme apontado em [BRIN93a], outras abordagens também sofrerão com tal situação. Já na fase de junção, como os *buckets* são percorridos em uma ordem

predeterminada, cada *bucket* é lido apenas uma vez - inclusive os *buckets* de *overflow*, conforme visto anteriormente. Cabe observar também que, na prática, em geral teremos várias junções encadeadas, de forma que a primeira a ser realizada terá como consequência uma ordenação nos dados, se porventura os mesmos estivessem distribuídos aleatoriamente. A partir daí, os conjuntos que utilizarem tanto o conjunto resposta da primeira junção como qualquer um dos conjuntos envolvidos poderão tirar proveito dessa ordenação.

Embora o acesso ao disco seja a operação que consome mais tempo nessa fase do processamento de junções espaciais, o tempo utilizado para a comparação e identificação de pares de MBRs que se interceptam não é desprezível. No entanto, poucos artigos sobre o assunto mencionam o número de comparações entre MBRs necessárias. Uma exceção é o trabalho de [BRIN93a], que mostra diversos resultados nesse sentido, utilizando R*-trees. Apenas a título de ilustração, o melhor desempenho dos algoritmos apresentados lá necessitou realizar um número de comparações de MBRs que ficou em torno de dez vezes o tamanho do conjunto resposta, havendo casos em que esse fator chegou a 25. Nossa abordagem conseguiu taxas que ficaram em torno de duas a três vezes o tamanho do conjunto resposta.

Finalmente, a fase de partição, que é a fase que realiza mais acessos a disco de forma não sequencial, pode ser armazenada e mantida como um índice para determinados conjuntos de dados [DEWIT96]. Conforme observamos, o aumento da massa de dados é menor do que o inchaço dos MBRs, resultando em uma estrutura de dados viável de ser armazenada na base de dados. Assim, tendo índices armazenados na forma aqui descrita para os conjuntos de dados, o processamento de junções espaciais poderá ser realizado diretamente a partir da fase de junção - que faz acessos tipicamente sequenciais.

6. Conclusões e Trabalhos Futuros

Este trabalho apresentou uma forma de se realizar junções espaciais segundo o paradigma *hash-join*, a 2DHM, e que está em conformidade com os modelos correntes de processamento de junções espaciais. Ao contrário de quase todas as outras abordagens, a abordagem 2DHM conseguiu manter a restrição de ter partições que não se sobrepõem, às custas de muito pouco inchaço nos dados. Além disso, conseguimos resolver satisfatoriamente o problema de se eliminar os pares de MBRs duplicados.

Comparado aos outros métodos, a nossa abordagem se revelou superior quanto ao nº necessário de comparações de MBRs - o que significa que, para dois casos específicos, podemos esperar um desempenho melhor para a nossa abordagem: pequenos conjuntos de dados, que possam residir na memória, e conjuntos de dados cujos índices sejam mantidos armazenados na base. No entanto, os resultados obtidos por [DEWI96] indicam outras situações em que podemos esperar um desempenho melhor para o 2DHM.

É importante notar que a abordagem 2DHM possui características que facilitam a paralelização do algoritmo. Um trabalho nesse sentido está em sua fase inicial, de forma que não dispomos ainda de resultados práticos.

Finalmente, os resultados obtidos nesse trabalho juntamente com os resultados obtidos por [DEWI96] sugerem que um amplo estudo comparativo entre os vários métodos, utilizando R*-trees, *hash-join* e partições de tamanho variável deva ser realizado, para que se possa identificar claramente quando um ou outro algoritmo se mostra mais eficiente a nível tanto de utilização de memória/disco quanto de tempo gasto.

7. Agradecimentos

Gostaríamos de agradecer a CAPES, pelo financiamento parcial deste trabalho, bem como ao Prof. Cláudio Esperanza por sua leitura, revisão e contribuições.

8. Bibliografia

[BECK90] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger: "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In Proceedings of the ACM SIGMOD Intl. Conf on Management of Data, Atlantic City, NJ, 1990.

[BERC96] S. Berchtold, D. A. Keim, H. P. Kriegel: "The X-Tree: An Index Structure for High-Dimensional Data", In Proceedings of 22th International Conference on Very Large Data Bases, Bombay, India, 1996.

[BRIN93a] T. Brinkhoff, H. P. Kriegel, and B. Seeger: "Efficient processing of Spatial Joins Using R-Trees". In Proceedings of the 1993 ACM-SIGMOD Conference, Washington, DC, USA,May 1993.

[BRIN93b] T. Brinkhoff, H. P. Kriegel, and R. Schneider: "Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Saptial Databse Systems". In Proceedings of 9th International Conference on Data Engineering, Vienna, Austria, 1993.

[BRIN94] T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger: "Multi-step Processing of Spatial Joins". In Proceedings of the 1994 ACM-SIGMOD Conference, Minneapolis, USA, May 1994.

[BRIN95] T. Brinkhoff, H. P. Kriegel, R. Schneider, and A. Braun: "Measuring the Complexity of Polygonal Objects". In Proceedings of ACM International Workshop on Advances in Geographic Informatin Systems, Baltimore, MD, USA, December 1995.

[BRIN96] T. Brinkhoff, H. P. Kriegel And B. Seeger: "Parallel Processing of Spatial Joins Using R-trees", In Proceedings of 12th International Conference on Data Engineering, New Orleans, LA, USA, 1996.

[DEWI84] DeWitt, R. et ali: "Implementation Techhniques for main memory database systems". In 1984 SIGMOD, June.

[DEWI96] D. DeWitt and J. M. Patel: "Partition Based Spatial-Merge Join". In Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996.

[GAED95] Gaede, V. "Optimal Redundancy in Spatial Database Systems". SSD95.

[GÜNT93] O. Günther: "Efficient Computation of Spatial Joins". In Proceedings of 9th International Conference on Data Engineering, Vienna, Austria, 1993.

[GUTT84] A. Guttman: "R-Trees: A Dynamic Index Structure for Spatial Searching". In Proceedings of the ACM SIGMOD Intl. Conf on Management of Data, Boston, MA, 1984.

[IBGE96] Fundação Instituto Brasileiro de Geografia e Estatística: "Malha Municipal Digital do Brasil - 1994", Rio de Janeiro, 1996.

[MING95] M. L. Lo and C. V. Ravishankar: "Generating Seede Trees From Data Sets". In Proceedings of 4th International Symposium on Large Spatial Databses, Portland, ME, USA, August, 1995.

[MING96] M. L. Lo and C. V. Ravishankar: "Spatial Hash-Joins". In Proceedings of the 1996 ACM-SIGMOD Conference, Montreal, Canada, June 1996.

[SAIF94] MELP Spatial Archive and Interchange Format (SAIF): Formal Definition Release 3.1 April 1994 Reference Series Volume 1 - Surveys and Resource Mapping Branch - Ministry of Environment, Lands and Parks (MELP) - Province of British Columbia - Canada.

[SAME90] H. Samet: "The Design and Analysis of Spatial Data Structures", Addison-Wesley Publishhing Company, 1990.

[SQL395] X3H2-95-084/DBL:YOW-004, (ISO-ANSI Working Draft) Database Language SQL (SQL3),Jim Melton (ed.), March, 1995

[VEEN95] H. M. Veenhof, Peter M. G. Apers, Maurice A. W. Houtsma: "Optimisation of Spatial Joins Using Filters". In Advances in Databases, 13th British National Conference on Databases, BNCOD 13, Manchester, United Kingdom, July 1995.

[ZIMB97] Zimbrão, G. e Souza, J. M. "Using Raster Approximations For Processing of Spatial Joins". COPPE/UFRJ - *Relatório Técnico ES-442/97.*