Universidade do Minho
Escola de Engenharia

Gil Fernando Ferreira da Costa Pontes

R-Trees, sua influência no desempenho dos sistemas de processamento analítico

Novembro de 2008

Universidade do Minho
Escola de Engenharia

Gil Fernando Ferreira da Costa Pontes

R-Trees, sua influência no desempenho dos sistemas de processamento analítico

Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor Orlando Manuel de Oliveira Belo

Novembro de 2008

Dedico esta dissertação de mestrado ao mais recente membro da minha família, a recém-nascida e querida afilhada Eva Maria.

Agradecimentos

Agradeço a todos os que me ajudaram durante o exercício deste projecto, independentemente do tipo de apoio proporcionado.

Agradecimentos especiais à minha família, nomeadamente ao meu pai, à minha mãe, ao meu irmão e minha cunhada por todos os conselhos, apoio emocional, incentivos e demais intervenções que em muito me auxiliaram ao longo destes meses. À minha namorada e amigos por toda a ajuda nos momentos menos bons e clima de euforia nos momentos mais positivos. E, por último, mas não menos importante, agradeço todas as orientações e transmissão de sabedoria do Professor Doutor Orlando Belo, essencial para que este trabalho chegasse a “bom porto”.

Resumo

R-Trees, sua influência no desempenho dos sistemas de processamento analítico

Actualmente, os sistemas de processamento analítico, vulgarmente reconhecidos por sistemas OLAP (Online Analytical Processing), constituem uma peça muito importante no xadrez dos processos de tomada de decisão. A sua aplicação tem vindo a aumentar nos últimos anos. As enormes facilidades de exploração de dados que estes sistemas colocam à nossa disposição fazem com que a sua adopção pelas empresas, e sua consequente utilização, seja perfeitamente natural, em especial em organizações que têm mantido data warehouses ao longo da sua existência. Por outro lado, a garantia que estes sistemas dão relativamente a excelentes tempos de resposta em queries do tipo ad hoc é, por outro lado, um outro factor a ter em conta quando os seleccionamos para assegurarem a primeira linha de interacção de um sistema de suporte à decisão com os seus utilizadores. Todavia, sabemos que, apesar destas excelentes características, o desempenho de qualquer sistema analítico tende a diminuir com o aumento do número de dimensões, bem como com o aumento natural do volume de dados a manipular. Uma possível solução para atenuar esta perda de desempenho pode ser desenvolvida em torno da criação de estruturas de indexação eficientes, especificamente desenvolvidas para ambientes

multidimensionais de dados. Nesse sentido, neste trabalho de dissertação, pretende-se estudar e analisar pormenorizadamente as estruturas de indexação multidimensional e a sua influência em ambientes de processamento analítico, incidindo com particular ênfase no domínio das R-Trees e suas variantes.

Abstract

R-Trees, its influence on the performance of online analytical processing systems

Nowadays, analytical processing systems, usually recognized as OLAP (Online Analytical Processing) systems, plays an important role on the chess of the decision support systems. Its application has been increasing in the last years. In one hand, the huge and easiest ways of exploring data, available by these systems, stimulate its adoption by the enterprises, and consequently its use, became perfectly natural, specially by organizations that kept data warehouses during its existence. On the other hand, these systems guarantee excellent response times to ad hoc queries type, which is a factor to take in account when we select them to ensure the first line of interaction between a decision support system and its users. However we know, that despite all its excellent characteristics, the performance of any analytical system it tends toward a decrease with the increase of the number of dimensions, as well as the natural increase of the natural volume of data to manipulate. One possible solution to relief this performance lost, may be develop in lathe of innovating efficient indexing structures, namely develop for multidimensional environments of data. In this direction, this thesis work, wants to accomplish the study and the analysis, in detail, of multidimensional data structures and its influence on analytical

processing environments, focusing, in particular, on the domain of R-trees and its variants.

Índice

Índice de Figuras

Índice de Algoritmos

Capítulo 1

1 Introdução

1.1 Contextualização

O desenvolvimento acelerado nas plataformas tecnológicas tem sido uma realidade constante ao longo das últimas décadas. A necessidade de se criarem soluções informáticas passíveis de gerir quantidades de dados cada vez maiores tem sido uma exigência imposta por todos os sectores de negócio. Como tal, o tratamento dos dados conheceu contornos distintos. O poder da informação tornou-se cada vez maior. Através do registo de todas as actividades de negócio, as empresas/instituições conhecem tendências, viabilizam estudos de mercado e traçam rumos e estratégias para o futuro. As soluções de *Business Intelligence*, vulgarmente conhecidas por sistemas de suporte à decisão, são responsáveis por proporcionar esta visão geral à admnistração das entidades empresariais.

A informatização da sociedade levou ao surgimento de inúmeras aplicações que, de forma muito natural, potencializaram a informação intrínseca presente nos dados. Embora a informação obtida pelos “algarismos e texto” represente o *core* das tecnologias de informação actuais, a noção de espaço está cada vez mais

presente. A obtenção de informação relevante em aplicações médicas, geográficas, multimédia, espaciais, entre outros, é uma realidade cada vez mais visível. Sendo assim, os dados passaram a conter uma característica muito relevante – dimensionalidade. Se, na maioria destas aplicações, falamos, principalmente, em bidimensionalidade ou tridimensionalidade, no caso dos sistemas de suporte à decisão, mais precisamente, os sistemas de processamento analítico, este conceito e ideologia é levado a outros níveis.

As estruturas de dados acompanharam este processo evolutivo. Tal como referido em [MNPT03, MNPT05], a mítica B-Tree, assim como as suas variantes, foram, e continuam a ser, um marco de referência para a informação unidimensional, no entanto não estão, particularmente, adaptadas para o tratamento de informação espacial. A exigência imposta por dados mais complexos, nomeadamente, o tratamento de regiões de dados levou ao aparecimento de estruturas de indexação com características muito distinctas, mas com objectivos idênticos, ou seja, providenciar acesso rápido aos dados e facilitar a execução das operações base – inserção, remoção e procura. Neste sentido, surgiram as estruturas multidimensionais de dados. Estes índices organizam os dados segundo a sua localização espacio-temporal.

A R-Tree[Gut84] surgiu como uma das mais promissoras e importantes estruturas multidimensionais. A sua semelhança orgânica com a B-Tree foi notória e contribuiu para o seu conhecimento. Não obstante, e mais importante ainda, foi a sua aptidão para lidar com dados em ambientes multidimensionais. Organizada segundo uma hierarquia de regiões multidimensionais, a R-Tree adequa a sua orgânica à localização espacio-temporal dos dados, recorrendo a técnicas de particionamento do espaço, *splitting*, de forma a reduzir o tempo de execução das suas operações e efectuar um melhor aproveitamento do espaço.

As características base da R-Tree tornaram-na extraordinária, tendo em conta a aceitação por parte das comunidades científica. Como tal, um vasto lote de estruturas baseadas na R-Tree foram surgindo ao longo dos tempos, de modo

a optimizar determinadas características. Nesta Tese serão abordadas algumas dessas variantes, nomeadamente a R^+-Tree, vocacionada para a optimização de *point queries*, a R^*-Tree, uma estrutura que através de alterações na sua política de particionamento espacial obteve desempenhos bem superiores à R-Tree, a Hilbert R-Tree, uma estrutura baseada em *space filling curves* que permite um melhor controlo da localização dos dados e das suas regiões multidimensionais, a X-Tree, uma estrutura vocacionada para ambientes de alta dimensionalidade e, por último, a Ra*-Tree, uma estrutura adaptada da R*-Tree com dados agregados nos seus nodos intermédios para uma melhor adaptação ao tipo de *queries* mais usuais em sistemas de processamento analítico.

O impacto gerado por esta estrutura de dados nas mais variadas áreas foi deveras substancial. Na actualidade é uma referência para os principais SGBD (Sistemas de Gestao de Bases de Dados), principalmente para a área relacionada com as bases de dados espaciais.

1.2 Motivação

Os sistemas de processamento analítico foram adoptados por muitos como parte integrante de uma arquitectura de um sistema de suporte à decisão em adjacência aos sistemas de data warehousing. Estes sistemas permitem capitalizar os dados contidos nos data warehouses em informação com formatos mais próximos e relevantes para os utilizadores através de um vasto rol de operações muito sofisticadas, nomeadamente operações de agregação, de filtragem e muitas outras mais resultantes da combinação essencialmente, entre os dois tipos de operações referidos. Conceptualmente, estes sistemas empregam modelos multidimensionais de representação da informação, denominados por *DataCubes*, em que cada um dos seus eixos k-dimensional

representa um factor de análise (dimensão) e cada célula contém as medidas envolvidas.

Nos últimos anos, como consequência de processos de investigação muito apurados, estes sistemas têm sido alvos constantes em processos de procura de melhores modelos para a satisfação de *queries ad hoc* e na optimização do seu desempenho. Como resultado, as estruturas de indexação multidimensional têm sofrido grandes evoluções, sendo apontadas actualmente como possíveis soluções para o aperfeiçoamento destes sistemas, em particular como elementos preponderantes para a diminuição dos tempos do sistema em processos de obtenção de dados.

O funcionamento dos índices tradicionais das bases de dados relacionais segue um padrão distinto relativamente às estruturas multidimensionais. Se abordarmos os mais emblemáticos, B-Tree e Bitmaps, constatamos diferenças profundas nas suas interacções. A B-Tree é uma estrutura vocacionada para ambientes unidimensionais, cuja construção se baseia em chaves de procura, que podem corresponder a um atributo ou à concatenação de vários atributos de uma tabela[GMUW01]. Os bitmaps são estruturas semelhantes a arrays multidimensionais que alocam uma determinada chave binária a cada um dos distintos valores de um atributo dimensional, efectuando operações de controlo para aceder rapidamente à informação pretendida. Tal como referido em [Sar97], estas estruturas apresentam limitações, no caso dos bitmaps o *overhead* computacional gerado por estes cálculos pode ser extenso, para além disto, estas estruturas não aproveitam devidamente o espaço, a não ser que estejam devidamente comprimidas, o que por si só, eleva a exigência computacional associada à sua reorganização. No caso da B-Tree, o facto de associarmos uma estrutura a cada atributo dimensional obriga à manutenção individual das existentes, assim como para efectuarmos uma operação sobre vários atributos obriga a intersecção dos seus resultados. O tempo na obtenção dos dados e o *overhead* computacional são obviamente afectados negativamente. A R-Tree

apresenta vantagens significativas neste campo, pois efectua um tratamento igual dos atributos dimensionais, organizando-os segundo uma hierarquia baseada em regiões multidimensionais dos dados. Desta forma, uma *range query* é obtida através da travessia desta estrutura, evitando as operações complexas das estruturas anteriormente referidas. No entanto, apresenta limitações relativamente à sua performance em ambientes de média e alta dimensionalidade, em muito devido à complexidade associada à organização das suas regiões no espaço multidimensional.

Partindo da definição das R-Tree, podemos então classificá-las como estruturas multidimensionais hierárquicas, concebidas inicialmente para aplicações CAD (*Computer Aided Design*). Estas estruturas tiveram um sucesso considerável quando foram aplicadas em aplicações multimédia e geográficas. A sua similaridade com as B-Tree permitiu a sua incorporação nos sistemas de gestão de bases de dados, tendo sido essenciais na optimização de bases de dados espaciais. Apesar de terem alcançado um patamar de intervenção nos sistemas de bases de dados bastante importante, as R-Tree não são perfeitas e, como tal, sofreram inúmeras reestruturações no sentido de se colmatar algumas das suas conhecidas falhas de desempenho. Os factores de optimização abrangidos nesses processos de melhoria têm sido o tempo de processamento em operações de carregamento, a actualização e a procura de elementos de dados, a memória utilizada pelos sistemas analíticos durante a realização das suas operações base, os acessos ao disco e a adequação aos diversos cenários existentes em ambientes multidimensionais - dimensionalidade, cardinalidade ou a densidade dos dados.

Tendo em conta as características base deste tipo de estruturas de dados, a associação com os sistemas de processamento analítico não é descabida. Aliás, tem sido um desafio para toda a comunidade científica optimizar e adaptar estes índices para a sua incorporação no modelo espacial/muldimensional dos sistemas OLAP.

1.3 Objectivos

Desde o seu aparecimento que as R-Tree têm uma importância considerável do ponto de vista comercial e científico. Estas estruturas são muitas vezes catalogadas como um "ícone" das estruturas de indexação multidimensional, uma vez que foram, em várias alturas, determinantes para o aparecimento de novas estruturas de indexação e sua consequente evolução. Nesta tese pretendeu-se realizar um estudo detalhado sobre a evolução das R-Tree, visando, principalmente, a sua adequação às funcionalidades e propriedades dos sistemas de processamento analítico e, em particular à:

- capacidade para lidar com dados multidimensionais nas suas mais importantes operações (carregamento, procura e actualização);
- presença de operações de agregação de dados de modo a reduzir acessos ao disco e tempo de processamento na obtenção de dados;
- optimização do espaço em disco, nomeadamente através do aproveitamento da capacidade dos blocos e consequente diminuição de acessos aleatórios;
- aptidão para as típicas e usuais *range/window* queries.

É, portanto, imperial demonstrar em pormenor o funcionamento e orgânica da R-Tree, assim como abordar, em pormenor, algumas das suas mais emblemáticas variantes. Desta forma, pretendeu-se mostrar alguns dos fios condutores que comunidade científica tomou no sentido de optimizar e alargar o uso desta estrutura nas mais diversas áreas. Os sistemas de processamento analítico são abordados como a área preferencial neste estudo. Logo, uma

análise sobre a incidência, impacto e abordagens tomadas pelo universo R-Tree nesta área serão alvos de especial destaque.

1.4 Estrutura da Tese

Esta tese é composta por duas grandes componentes, o universo R-Tree e a sua incidência nos sistemas de processamento analítico. Após esta breve introdução, o capítulo 2 é, inteiramente, dedicado à R-Tree e suas demais variantes. Inicialmente, será demonstrada a orgânica, funcionamento, propriedades e outras ilações pertinentes sobre a estrutura base desta tese. O objectivo é proporcionar ao leitor um entendimento perfeito sobre todas as componentes da R-Tree. De seguida, serão abordadas 4 variantes desta estrutura, R^+-Tree, R*-Tree, Hilbert R-Tree e X-Tree. Estas estruturas são algumas das mais importantes variantes e demonstram algumas das optimizações e avanços obtidos face à estrutura base. São analisadas em pormenor, com especial destaque para a execução das operações base – inserção, remoção e procura. Por último, é efectuado um ligeiro *overview* sobre o que ficou por demonstrar relativamente a este vasto universo, sendo, no entanto, referenciado algumas das áreas de intervenção de algumas das inúmeras variantes desta estrutura.

O capítulo 3 é vocacionado para a ligação entre estes 2 mundos, isto é, à conjugação do universo R-Tree com as características dos sistemas de processamento analítico. Inicialmente, é efectuada uma pequena contextualização referenciando as ideias base que levaram a esta abordagem, nomeadamente o tratamento de dados multidimensionais e tipo de operações de procura usuais nos sistemas de processamento analítico. Posteriormente, será analisada ao detalhe a Ra*-Tree, a variante adaptada aos conceitos e ideologias deste tipo de sistemas. Por último, será demonstrado o impacto e o

comportamento desta estrutura mediante o tipo de informação existente nestas aplicações, nomeadamente a sua adaptação ao volume, cardinalidade, dimensionalidade e disposição dos dados. A tese é terminada no capítulo 4, após a exposição das conclusões e das linhas de orientação para futuro enriquecimento deste trabalho.

Capítulo 2

O Universo das R-Tree

A R-Tree [Gut84] marcou e iniciou uma nova era nas estruturas multidimensionais de dados. Ao longo das últimas décadas, o aparecimento de aplicações, mais robustas e exigentes, relacionadas com o tratamento de informação no espaço (CAD, geográficas, multimédia, etc.) motivou a criação de estruturas de indexação adaptadas a estes novos conceitos e desafios. O manuseamento dos dados nas diversas operações base – inserção, remoção e procura – adquiriu novos contornos, assim como uma nova e importante variável – dimensionalidade.

Neste sentido, Antonin Guttman, em 1984, criou uma estrutura que se revelou um "ícone", não só pelas performances demonstradas, mas também pela simplicidade da sua orgânica e funcionamento, em muito similar à consagrada B-Tree. Apesar de ter alcançado uma elevada notoriedade no seio das estruturas multidimensionais de dados, a R-Tree não é perfeita e, como tal, sofreu inúmeras reestruturações no sentido de se colmatar algumas das suas conhecidas falhas de desempenho. Desta forma, o surgimento de novas estruturas, variantes da R-Tree, foi um acontecimento muito natural.

Ao longo do presente capítulo será efectuada uma descrição pormenorizada da R-Tree, a apresentação de algumas das suas mais importantes variantes e, por último, uma análise crítica sobre as vantagens e desvantagens demonstradas por cada uma. Pretende-se, desta forma, evidenciar o impacto gerado por esta estrutura na comunidade científica ao longo dos tempos, expondo os seus desenvolvimentos e a sua incidência em diversas áreas.

2.1 As R-Trees

As R-Tree são estruturas multidimensionais hierárquicas. Estas estruturas tiveram um sucesso considerável quando foram aplicadas em aplicações multimédia e geográficas. A sua similaridade com as B-Tree permitiu a sua incorporação nos sistemas de gestão de bases de dados, tendo sido essenciais na optimização de bases de dados espaciais.

Este índice organiza os dados (pontos, segmentos ou regiões k-dimensionais) de acordo com a sua disposição espacial, recorrendo a técnicas de particionamento do espaço multidimensional por forma a criar uma hierarquia de regiões multidimensionais que permita um melhor desempenho durante a execução das operações base – inserção, remoção e procura.

2.1.1 Estrutura e Propriedades

A R-Tree possui características similares à B-Tree, sendo mesmo referenciada como uma extensão da B-Tree para ambientes multidimensionais. Ambas são balanceadas e possuem os registos no último nível (folhas) que são, de facto, apontadores para a localização dos dados na BD. Como a R-Tree é constituída por regiões multidimensionais organizadas hierarquicamente, possui informação diferenciada comparativamente ao nível 0 (folhas).

Sendo assim, a R-Tree possui dois tipos de nodos:

- ***Nodos-Folha*** - Correspondem ao nível 0 da árvore e contêm registos do tipo
  (*I*, *apt_o*), em que o *apt_o* é o apontador para a posição do registo na BD e *I* representa um intervalo na forma (I_0, I_1...I_{k-1}), sendo que *k* é o número de dimensões associadas.

- ***Nodos-Superiores*** – Correspondem aos nodos cujo nível é superior a 0 e possuem entradas do tipo (*I*, *apt*), em que *apt* é um apontador para o nodo-filho e *I* é a representação do rectângulo k-dimensional, conhecido por *MBR (Minimum Bounding Rectangle)*, que envolve todos os objectos indexados pelo nodo-filho.

A R-Tree possui duas variáveis essenciais para o seu desempenho, *M* e *m*, em que *M* representa o número máximo de entradas que cada nodo poderá possuir, uma vez que depende do tamanho de página do disco. É calculado dividindo-se o tamanho de página do disco pela estimativa do espaço ocupado pelas entradas do nodo. Este procedimento minimiza a memória utilizada e o número de acessos à mesma. O valor de *m* representa o valor mínimo de entradas num nodo. A relação entre estas variáveis é a seguinte:

$$m \leq \frac{M}{2}$$

A definição de *m* depende do tipo de operacões mais comuns na BD, ou seja, se for uma BD onde ocorram poucos *updates* e bastantes operações de procura de dados, um valor alto de *m* será necessário, o que implica que os nodos se tornem mais densamente preenchidos. Desta forma, a altura da árvore mantém-se baixa, logo o tempo de resposta na procura será reduzido. Se, por outro lado, a operação de *update* for comum na BD, é recomendado um valor

baixo de *m* para manter os nodos pouco preenchidos. Como consequência, a operação de inserção é facilitada, pois a ocorrência de alterações estruturais, significativas, na R-Tree é em menor número, o que implica um menor tempo na sua execução.

Independentemente do critério adoptado na definição das capacidades dos nodos, a R-Tree possui o seguinte conjunto de propriedades:

- Todos os nodos-folha, exceptuando a raíz, contêm entre *m* e *M* registos.
- Todos os nodos-superiores, exceptuando a raíz, contêm entre *m* e *M* nodos-filho.
- Para cada entrada (*I*, *apt_o*), *I* representa o menor rectângulo k-dimensional capaz de envolver o objecto representado.
- Para cada entrada (*I*, *apt*), *I* representa o menor rectângulo k-dimensional capaz de envolver o nodo-filho.
- A raíz possui, pelo menos, 2 nodos-filho, exceptuando a situação em que se encontre no nível 0, sendo, simultaneamente, um nodo-folha.
- A árvore é totalmente balanceada, encontrando-se os nodos-folha no mesmo nível.

Na Figura 1 podemos observar um exemplo de uma representação espacial de uma R-Tree cujos dados são regiões rectangulares, enquanto que na Figura 2 é demonstrada a sua respectiva representação hierárquica.

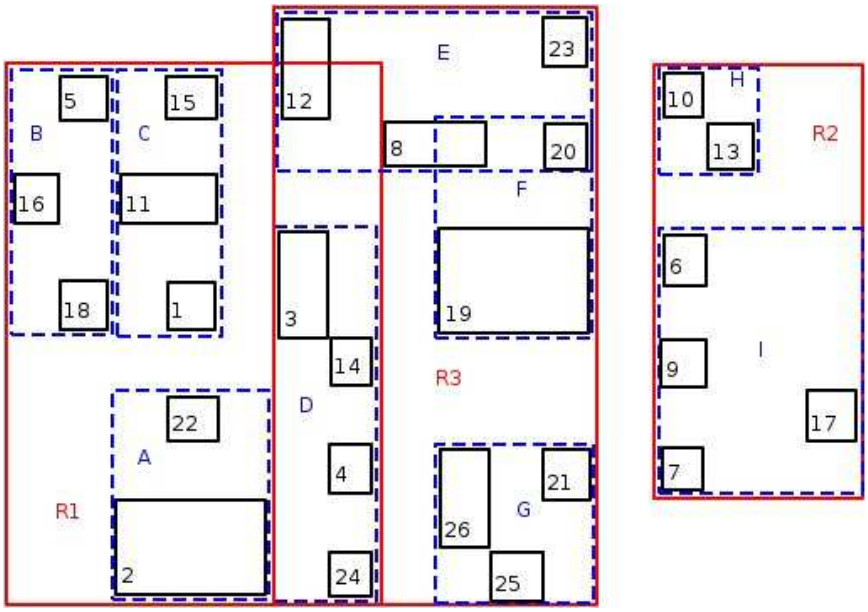

Figura 1 – Representação Bidimensional de uma R-Tree

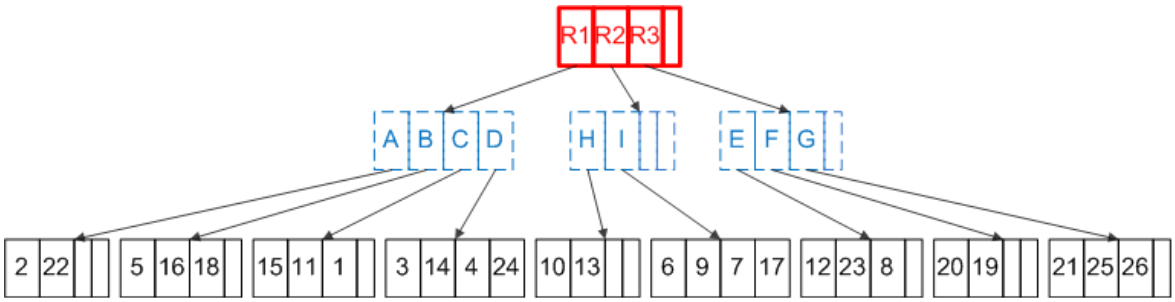

Figura 2 – Representação Hierárquica da R-Tree

A altura máxima de uma R-Tree é definida pela seguinte equação:

$$H_{max} = \lceil \log_m N \rceil - 1$$

Este valor é atingido tendo em conta a capacidade mínima dos nodos da R-Tree (*m*). Se tivermos *N* dados para alocar numa R-Tree, pelo menos, *m* entradas são indexadas por cada nodo. Desta forma, o número máximo de nodos da R-Tree é obtido através da seguinte equação:

$$Nodos_{max} = \left\lceil \frac{N}{m} \right\rceil + \left\lceil \frac{N}{m^2} \right\rceil + \ldots + 1$$

Estas fórmulas são representativas do cenário relativo ao pior cenário na utilização de uma R-Tree, isto é, demonstram os valores obtidos numa situação onde existe o menor aproveitamento do espaço de armazenamento de dados, o que directamente implica um menor aproveitamento de disco e consequentemente maiores tempos na travessia da estrutura.

2.1.2 Algoritmos

Em [Gut84], Guttman definiu um conjunto de algoritmos que manipulavam a estrutura da R-Tree de forma a permitir a execução das operações base – inserção, remoção e procura. Os algoritmos foram designados por ChooseLeaf, FindLeaf, AdjustTree, CondenseTree e SplitNode. Neste trabalho não serão demonstrados nem analisados estes algoritmos em pormenor. A abordagem adoptada consiste em agrupar as funcionalidades de cada um de forma a construir um único algoritmo para cada uma das operações base.

Antes de avançar para a descrição das operações na R-Tree, convém realçar a existência de 2 situações que podem ocorrer durante a execução das

mesmas que implicam uma desgaste computacional maior, em muito devido às alterações que provocam na estrutura. Tal como referido em [KDFB99], os cenários de *Overflow* e *Underflow* nos nodos da R-Tree obrigam a uma reorganização da estrutura. *Overflow* está associado a operações de inserção e acontece quando uma nova entrada é alocada num nodo totalmente preenchido. Esta situação obriga à criação de um novo nodo, sendo as *M* + 1 entradas distribuídas, posteriormente, pelos 2 nodos. A adição de 1 nodo na estrutura pode desencandear outras situações de *Overflow* em níveis mais elevados da estrutura. Este fenómeno acontece com maior frequência em R-Trees com os nodos mais densamente preenchidos. *Underflow* está associada a cenários de remoção e acontece quando uma entrada é removida de um nodo que apenas possui *m* entradas. Este cenário obriga à eliminação do nodo, sendo as *m* - 1 entradas reinseridas na estruturas. Esta situação é mais usual em R-Trees com nodos pouco preenchidos. Na Figura 3 podemos observar uma situação de *Overflow* e *Underflow* para um nodo com *M*=4 e *m*=2.

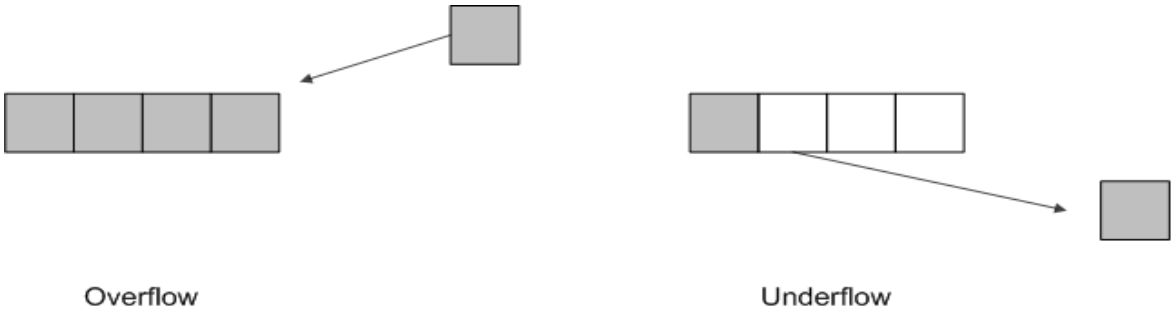

Figura 3 – Overflow e Underflow na R-Tree

2.1.2.1 Inserção

A R-Tree tem um crescimento Bottom-Up, ou seja, cresce de baixo para cima. A única forma desta estrutura aumentar a sua altura é através da ocorrência de operações de *splitting*. Estas operações ocorrem na presença de situações de *overflow*, originando a criação de um novo nodo, e consequente entrada no nodo-pai caso tenhamos uma R-Tree com 2 ou mais níveis.

O processo de inserção de um elemento na R-Tree ocorre através da travessia da estrutura desde a sua raíz até ao nodo-folha mais apropriado para alocar o novo membro. A escolha do nodo a seguir em cada nível é efectuada através do critério de menor alargamento de área necessário para cobrir o novo elemento. Caso, segundo este critério, haja um empate entre nodos, a escolha recai sobre o que tiver menor área.

O Algoritmo 1 demonstra as operações necessárias para a execução de uma inserção numa R-Tree. Para melhor compreensão do algoritmo, suponhamos que *R* é a raíz da R-Tree, *L* um nodo-folha e *X* o elemento a inserir.

```
Função Inserção (Elemento X, Nodo R)
  • Averiguar qual a entrada de R que necessita de menor
    alargamento de área para alocar X. Se houver necessidade de
    desempatar, usar como critérios a menor área e o menor número
    de entradas, por esta ordem de importância respectivamente.
    Passar para o nível seguinte da estrutura e efectuar o mesmo
    procedimento até chegar ao nodo-folha adequado, L.
  • If L não estiver totalmente preenchido
      ▪ Alocar X a uma das entradas disponiveis em L
  • Else
      ▪ Criar 2 nodos L e L'
```

Algoritmo 1 – Inserção na R-Tree com *Quadratic Cost Algorithm* como técnica de *Splitting*

- Agrupar X e todas as restantes entradas, anteriormente, pertencentes a L. Escolher as 2 entradas, E1 e E2, que entre si formem o rectângulo k-dimensional que necessite de maior alargamento de área. Resultado = área[E1+E2] – área[E1] – área[E2]
- Cada uma das entradas escolhidas,E1 e E2, forma um grupo, G1 e G2, respectivamente. Testar cada uma das restantes entradas e alocar a que mostrar maior preferência por um dos 2 grupos. Para tal, escolhe-se a entrada que denote maior diferença de área entre os rectângulos originados pelo grupo que a tenta alocar. Resultado = área[G1+entrada] – área[G2+entrada]. Alocar a entrada escolhida ao grupo que necessite de menor alargamento de área para cobrir a entrada. Em caso de empate, os critérios de desempate são a menor área e o menor número de elementos, por esta ordem de importância.
- Efectuar este procedimento até todas as entradas serem alocadas a cada um dos grupos.
- Garantir que ambos os grupos tenham pelo menos *m* elementos cada um.
- Associar os grupos aos nodos L e L’
- Efectuar uma travessia da estrutura desde a raíz até L para actualizar os MBRs dos nodos atingidos
- Repetir este procedimento caso seja gerado *Splitting* nos níveis superiores. Se tal acontecer em R, uma nova raíz tem de ser criada, tendo como suas entradas R e R’

Algoritmo 1 (continuação) – Inserção na R-Tree com *Quadratic Cost Algorithm* como técnica de *Splitting*

As operações que maiores alterações provocam na estrutura da R-Tree são as operações de *Splitting*. Estas ocorrem quando há uma tentativa de inserção de um elemento num nodo que já contém *M* entradas. De uma forma resumida e independentemente do algoritmo escolhido, esta operação baseia-se em dividir as *M*+1 entradas em 2 grupos e alocá-las em 2 nodos. O objectivo é minimizar o tamanho dos *MBRs* para assim diminuir a probabilidade de haver ocorrência de sobreposição entre estes.

Guttman apresentou 3 algoritmos de *Splitting*[Gut84]:

- ***Exhaustive Splitnode Algorithm*** – Consiste em efectuar todas as possíveis 2^{M+1} combinações de agrupamentos entre as *M*+1 entradas. Este algoritmo para BD com elevado volume de dados é bastante ineficiente devido ao tempo gasto na sua execução.
- ***Quadratic Cost Algorithm*** – Este algoritmo utiliza os 2 elementos que entre si formem um rectângulo k-dimensional que necessite de um maior alargamento de área. Consequentemente, aloca os restantes elementos a um dos iniciais mediante a preferência demonstrada. O príncipio base é alocar a entrada com maior preferência por um dos grupos, e assim sucessivamente até se alocarem todos os elementos. Embore não efectue os melhores *splits*, a relação entre a qualidade do *split* efectuado e o tempo dispendido na operação tornam este algoritmo no mais eficaz dos propostos por Guttman. No Algoritmo 1 é demonstrado o processo de inserção com recurso a este algoritmo.
- ***Linear Cost Algorithm –*** Este algoritmo escolhe para cada dimensão as entradas mais distantes entre si e aloca-as em grupos diferentes. Este procedimento é efectuado para todas as dimensões e as restantes entradas são distribuídas aleatoriamente pelos grupos. É o mais rápido dos 3 algoritmos apresentados, mas apresenta pior qualidade nos *splits* efectuados.

2.1.2.2 Remoção

As remoções de dados numa R-Tree podem originar situações computacionalmente mais exigentes, devido às modificações estruturais que podem implicar na estrutura. Normalmente, estes cenários são relativos a situações de *underflow*. Neste sentido, os elementos do nodo afectado são, posteriormente, reinseridos na R-Tree e o nodo eleminado. Para além disto, toda a componente da estrutura afectada é actualizada. O facto de reinserirmos os elementos na estrutura torna-a mais eficaz, pois cada elemento é alocado no nodo-folha mais indicado para si. No entanto, a possibilidade de haver inserções em processos de remoção, torna-se toda esta operação mais demorada e exigente computacionalmente, pois, como vimos anteriormente, o processo de inserção, também, pode levar a modificações estruturais significativas na R-Tree. No caso mais simples, uma remoção implica uma actualização dos nodos afectados por esta operação, nomeadamente os *MBRs* das suas entradas.

```
Função Remoção(Elemento X, Nodo R)
  • If R é um nodo-folha
    ▪ Remover X
  • Else
    ▪ Percorrer a estrutura desde a raíz R até ao nodo-folha
      que contenha o elemento a eliminar X
    ▪ Remover X
    ▪ If nodo-folha ficar com m ou mais entradas
        o Actualizar os MBRs dos nodos afectados
```

Algoritmo 2 – Remoção de um elemento na R-Tree

- Else
  - Remover o nodo-folha e guardar os restantes elementos contidos nesse nodo
  - Actualizar o nodo-pai do nodo-folha, removendo a entrada referente a este último
  - If nodo-pai ficar com m ou mais entradas
    - ❖ Actualizar os nodos afectados até à raíz.
  - Else
    - ❖ Repetir o procedimento efectuado para o nodo-filho
- Reinserir todos os elementos guardados devido à eliminação de nodos em *underflow*. Os elementos devem ser inseridos no mesmo nível em que se encontravam anteriormente.
- If raíz apenas tiver um nodo-filho
  - Remover raíz
  - Definir o nodo-filho da anterior raíz como a nova raíz.

Algoritmo 2 (continuação) – Remoção de um elemento na R-Tree

No Algoritmo 2 podemos observar os procedimentos necessários para efectuarmos esta operação.

2.1.2.3 Procura

Uma das limitações da R-Tree consiste no facto de permitir sobreposições de regiões k-dimensionais (*MBRs*), independentemente do nível em que situam. Desta forma, há uma degradação no tempo de pesquisa, pois todos os nodos que intersectam, total ou parcialmente, a região de procura têm de ser percorridos. No entanto, a sua adequação á localização dos dados e o total balanceamento da

sua orgânica atenua este defeito. Devido a estes factores, a altura da estrutura não aumenta drásticamente, o que implica que um menor número de níveis seja percorrido. A operação de procura é efectuada na forma Top-Down, sendo percorrida desde a raíz até aos nodos-folha que se encontrem nos limites estabelecidos.

Nesta secção iremos analisar a procura sobre um determinado volume de dados, *range query*. A procura sobre uma entrada, *point query*, é efectuada de forma similar. No Algoritmo 3 pode-se observar o funcionamento base desta operação. Considere-se *R* a raíz da R-Tree e *Q* a região de procura.

```
Função Procura(Nodo R, Região Q)

• If R não é um nodo-folha
    ▪ Confrontar todas as entradas de R com Q
        o If entrada intersecta Q
            ❖ Invocar Procura(entrada.apt, Q)
• Else
    ▪ Confrontar todas as entradas de R com Q
        o If entrada intersecta Q
            ❖ Devolver entrada
```

Algoritmo 3 – *Range Query* numa R-Tree

Na Figura 4 temos presente uma ilustração exemplo de uma range query numa R-Tree bidimensional. Na Figura 5 podemos observar os nodos percorridos durante a execução desta operação. Facilmente, se pode observar o impacto que a sobreposição de *MBRs* gera nos processos de procura.

Figura 4 – Representação Bidimensional de uma Região de Procura

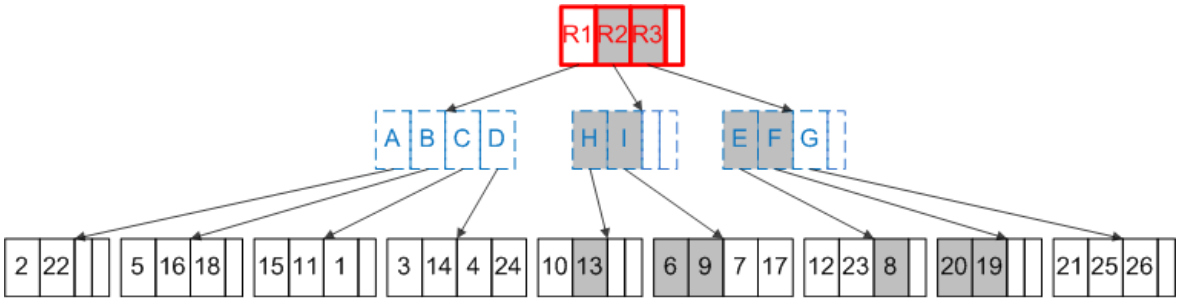

Figura 5 – Nodos Visitados na *Range Query* da Figura Anterior

Em relação ao impacto da dimensionalidade nos desempenhos da R-Tree, convém salientar que a mesma apresenta uma degradação clara, principalmente, para ambientes de média/alta dimensionalidade. Tal facto deve-se a uma gestão mais complexa do particionamento do espaço, o que envolve maiores custos de

processamento e tempo, além disto, o factor relativo à sobreposição de *MBRs* torna-se mais evidente, o que implica perdas de desempenho nas diversas operações.

Feita esta análise sobre a R-Tree, ao longo deste capítulo serão apresentadas mais algumas estruturas - variantes da R-Tree – que serão alvo de análise cirúrgica, isto é, centrada nas diferenças e nas vantagens/desvantagens demonstradas comparativamente a esta estrutura.

2.2 R^+-Tree

A R^+-Tree[SRF87] foi uma das primeiras variantes da R-Tree. O objectivo era claro, colmatar algumas das falhas visiveis na sua predecessora. Tal como referido em [MNPT05], a R-Tree possuia 2 desvantagens que se tornaram o foco de intervenção da R^+-Tree:

- As operações de procura de um dado elemento, *point query*, implicavam, em diversas situações, a travessia de múltiplos caminhos desde a raíz até ao nodo-folha que continha o elemento alvo da procura
- *MBRs* de elevado tamanho levavam, por vezes, à deterioração da performance obtida na execução de *range queries*. Tal facto deve-se ao aumento do factor de sobreposição na estrutura.

A R^+-Tree, de forma a combater estas debelidades da R-Tree, sustentou, como sua grande inovação, a inexistência de sobreposição de *MBRs* no mesmo nível da estrutura, o que significa que a travessia de múltiplos caminhos durante a execução das operações de procura relativas às *point queries*. A garantia de um único caminho percorrido foi, sem dúvida, um avanço significativo no desempenho obtido para a referida operação. No entanto, e como “não há bela sem senão”, a

solução encontrada para este efeito originou uma estrutura mais robusta e complexa, quer na sua orgânica como no seu funcionamento. Por forma a garantir a ausência de sobreposição de *MBRs* dentro do mesmo nível, a R^+-Tree adoptou o conceito de replicação dos dados. Este facto só acontece a entradas que, após o processo de particionamento do espaço no nível superior, intersectem 2 ou mais *MBRs* pertencentes ao nível referido. Sendo assim, as entradas que estejam nestas condições são alocadas a 2 ou mais nodos.

A redundância de dados gerada pela R^+-Tree se, por um lado, é benéfica para as *point queries*, por outro, afecta, negativamente, a performance nas *range queries*, visto que a estrutura, comparativamente á R-Tree e se tivermos em conta o mesmo volume de dados, é maior e mais robusta. Para esta operação a travessia de múltiplos caminhos é bastante provável, pois está sempre dependente do tamanho da região de procura de dados.

Na Figura 6 podemos observar uma representação espacial de uma R^+-Tree. Se olharmos com alguma atenção verificamos que o elemento **3** intersecta 2 *MBRs* do nível superior. Desta forma, como presente na Figura 7, onde podemos observar a disposição hierárquica desta R^+-Tree, este elemento encontra-se alocado a 2 entradas do nodo-superior, nomeadamente as representadas pelas regiões A e B.

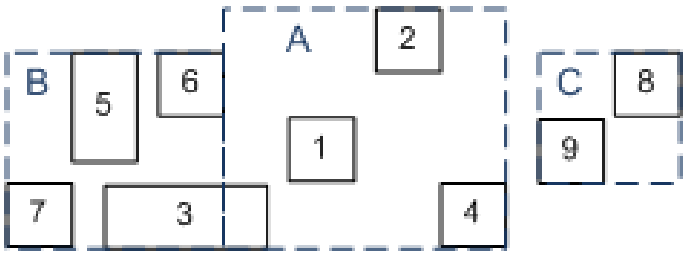

Figura 6 – Representação Bidimensional de uma R^+-Tree

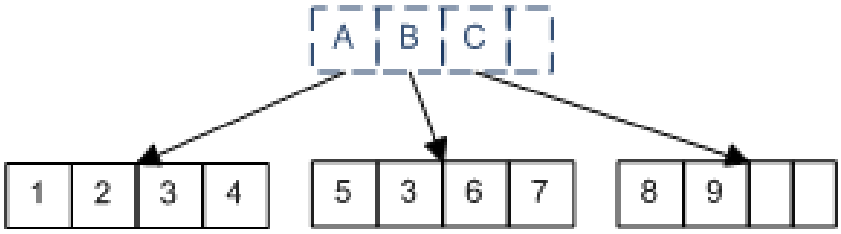

Figura 7 – Representação Hierárquica da R^+-Tree da Figura Anterior

O algoritmo de procura desta estrutura é similar ao apresentado pela R-Tree, independentemente de se efectuarem *point* ou *range queries*. No entanto, e tal como referido em [MNPT05], há necessidade de efectuar um controlo em relação à devolução de resultados. No caso das *point queries*, apenas um ramo da árvore atravessado, não havendo necessidade de iterar noutros *MBRs* que estejam no mesmo nível e também intersectem o elemento alvo. Para o caso das *range queries*, pelo facto da estrutura possuir duplicados nos nodos-folha, há necessidade de se eliminar dados repetidos de forma a não comprometer a qualidade da informação pedida.

Em relação à operação de inserção, existem de facto algumas diferenças comparativamente à R-Tree. Tal facto, não é estranho devido às divergentes caracteristicas estruturais entre os 2 índices. As mudanças prendem-se, principalmente, nos último níveis da árvore, pois aí se determina se um elemento é alocado a um ou mais nodos-folha. No caso de um elemento intersectar mais que um *MBR* que envolva entradas do nível 0, é alocado em nodos-folha que estejam contidos por essas regiões. Novamente, o elemento **3**, presente na Figura 1, é um exemplo representativo desta situação.

O Algoritmo 4 representa a inserção de um elemento na estrutura de uma R^+-Tree.

```
Função inserção(Nodo R, Elemento X)
  • If R não é um nodo-folha
    ▪ Para todas as entradas de R, testar se intersectam X
        o If entrada intersecta X
            ❖ Invocar inserção(entrada.apt, X)
  • Else
    ▪ If existe espaço no nodo-folha para conter X
        o Adicionar X ao nodo-folha
    ▪ Else
        o Efectuar Splitting
```

Algoritmo 4 – Inserção numa R^+-Tree

A operação de *Splitting* numa R^+-Tree contém uma diferença significativa comparativamente à R-Tree. As modificações geradas não só podem ocorrer em cadeia para os níveis superiores, como também se pode propagar para níveis inferiores. Tal facto decorre de uma das propriedades desta estrutura, em que uma entrada, existente nos nodos-superiores, terá de estar sempre contida por um *MBR* de um nível superior. Caso este último seja alvo de *Splitting* e deixe de conter totalmente a respectiva entrada, esta terá de ser alvo de *Splitting* , de forma a que se criem novos nodos que se encontrem totalmente contidos nos *MBRs* então existentes. Desta forma, pode-se afirmar que a complexidade na reorganização desta estrutura comparativamente à sua predecessora é mais elevada.

Por último, o processo remoção de um elemento também sofre algumas alterações. Tal facto deve-se à possível presença de duplicados desse elemento na estrutura. Desta forma, a execução desta operação pode ser mais demorada o que deteriora a sua performance. Além disto, este processo pode desencandear

inúmeras situações de *underflow* que obriguem a uma reorganização mais profunda e drástica da própria estrutura, afectando não só tempo de execução, como também a memória utilizada. Por este motivo, tal como referido em [GG98], esta estrutura não garante um espaço mínimo de utilização.

A R^+-Tree, embora iniba a existência de sobreposição de *MBRs* nos nodos-superiores, não obtém melhorias significativas quando confrontada com ambiente de média/alta dimensionalidade. A difícil manutenção da estrutura e a presença de duplicados tornam-se mais evidentes à medida que a dimensionalidade aumenta. Tal como a R-Tree, esta estrutura apresenta melhores performances em ambientes de média/baixa dimensionalidade.

2.3 R^*-Tree

A R*-Tree[BKSS90] é mais uma das importantes variantes da R-Tree, e tal facto deve-se, não só, às semelhanças estruturais entre ambas, mas acima de tudo aos ganhos obtidos por esta estrutura. Ainda hoje é vista como uma das mais robustas, fiáveis e emblemáticas variantes, servindo de base para investigações futuras como iremos ver mais à frente.

A estrutura da R*-Tree segue os mesmos príncipios e propriedades da sua predecessora. No entanto, os critérios de particionamento do espaço, assim como a alocação das entradas nos diversos nodos são diferentes. Em relação à optimização de *MBRs* – directamente relacionados com a optimização do espaço multidimensional - a R*-Tree tem em conta um número de factores superior ao da R-Tree. Esta última apenas tem em conta a minização da área de cada uma destas regiões como critério de absorção de novos elementos. Ao invés a R*-Tree segue os seguintes critérios:

- **A área de cada *MBR* deve ser mínima –** isto significa que área que não envolve os *MBRs* do nível imediatamente inferior – *dead space*

– deve ser a menor possível. O objectivo é proporcionar uma redução no número de travessias da estrutura durante a execução das diversas operações base. Para além, disto pretende-se facilitar a escolha desses próprias caminhos.

- **A sobreposição entre *MBRs* dos nodos-superiores deve ser mínima –** O objectivo é, também, minimizar o número de travessias da estrutura.
- **Os perímetros dos *MBRs* devem ser mínimos –** O objectivo é tornar estas regiões com um formato mais quadrangular. Desta forma, os *MBRs* tornam-se mais fáceis de agrupar a manipular. Segundo [4], agrupar *MBRs* com formatos relativamente semelhantes implica diminuição da área dos *MBRs* níveis superiores.
- **O espaço utilizado deve ser maximizado –** isto significa que a estrutura deve aproveitar ao máximo a capacidade das páginas do disco. Quanto maior for este aproveitamento, menor é a altura da árvore e menor número de acessos ao disco serão necessários durante a execução das operações base.

Todas estas condicionantes da R*-Tree tornam-na numa estrutura bastante robusta, que garante uma boa organização dos seus constituintes. Os desempenhos obtidos na execução das operações base são bem superiores aos da sua predecessora. Tendo em conta que não existem diferenças nos algoritmos de remoção e procura, os resultados provêm directamente de uma organização dos dados mais favorável. Na Figura 8 podemos observar uma representação simbólica de uma R*-Tree. Na Figura 9 temos presente a sua estrutura hierárquica.

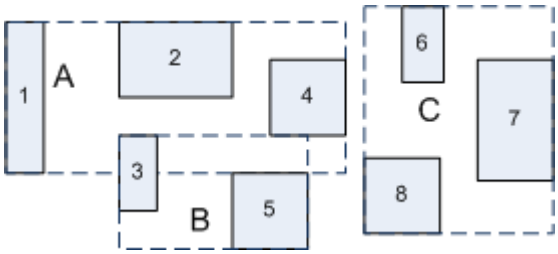

Figura 8 – Representação Espacial de uma R*-Tree

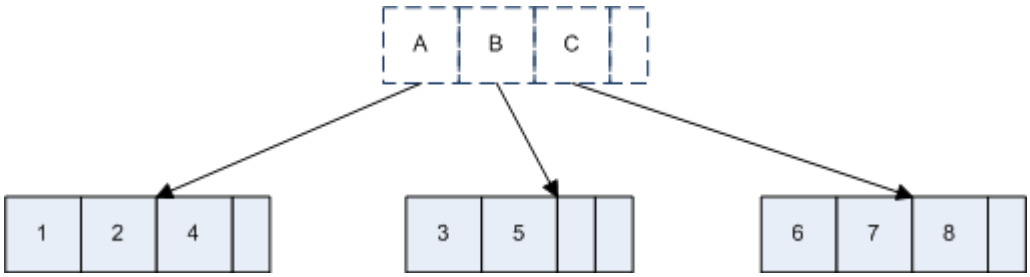

Figura 9 – Representação Hierárquica da R*-Tree

A operação de inserção, devido às condições acima referida, segue príncipios diferentes da sua predecessora. O Algoritmo 5 demonstra o processo de inserção numa R*-Tree

Função inserção(Nodo R, Elemento X)

- If R é um nodo-folha
  - If R possui espaço para conter X
    - Adicionar X a R
  - Else
    - Invocar *Tratamento de Overflow*
    - If *Tratamento de Overflow* provocou *Splitting*
      - ❖ Aplicar *Tratamento de Overflow* aos nodos-superiores afectados
    - If *Tratamento de Overflow* causar *Splitting* na raíz
      - ❖ Criar nova Raíz
- Else
  - If R está no nível 1
    - Escolher a entrada cujo *MBR* necessite de menor aumento de sobreposição relativamente aos *MBRs* das restantes entradas. Em caso de empate entre 2 ou mais entradas, a escolha recai sobre a que tiver menor alargamento de área, em último caso, a que tiver menor área na sua totalidade.
    - Invocar função inserção(entrada.apt, X)
  - If R está num nível superior a 1
    - Escolher a entrada que necessite de menor alargamento de área para comportar o novo elemento. Em caso de empate, escolher a que tiver menor área.
    - Invocar função inserção(entrada.apt, X)
- Ajustar todos os *MBRs* afectados pelo travessia da árvore durante o processo de inserção

Algoritmo 5 – Inserção de um elemento na R*-Tree

Como se pode observar pelo Algoritmo 5, o processo de inserção na R*-Tree é deveras mais complexo que o da sua predecessora. As condicionantes são em maior número. Um dos factos que se pode constatar é a existência de uma função auxiliar relativa aos cenários de *Overflow*. Tal é devido a inexistência de uma obrigatoriedade de se efectuar *Splitting* sempre que esta situação ocorre. Se esta função for invocada pela 1ªvez num determinado nível, com excepção da raíz, então procede-se a uma reinserção de algumas das *M+1* entradas na estrutura. Segundo [BKSS90], o valor que melhores resultados garante é 30% do número de elementos. Caso esta condição não se verifique, então procede-se à operação de *Splitting*. De salientar que a operação de reinserção tem em vista optimizar a própria estrutura assim como facilitar a sua manutenção. No entanto, as entradas reinseridas podem causar situações de *Overflow* nos nodos que as acolham, daí que o valor de 30% garanta algum equilíbrio nesta matéria.

As operações de *Splitting* na R*-Tree seguem, também, contornos distintos relativamente à R-Tree. Sempre que esta operação é invocada as *M+1* entradas são alocadas em 2 grupos, sendo a sua distribuição relativa às seguintes noções: no 1ºgrupo ficam *(m-1) + k* entradas, enquanto no outro grupo ficam as restantes entradas. *k* é um valor compreendido entre *1* e *M-2m+2*. O cálculo destas distribuições é importante para se obter os perímetros e áreas mínimos dos *MBRs*, permitindo assim obter o melhor particionamento espacial possível. Desta forma, é possível identificar o eixo sobre o qual será efectuado o particionamento espacial, assim como o número de entradas que ficam alocadas em cada um dos novos nodos.

O Algoritmo 6 demonstra a operação de *Splitting* na R*-Tree.

Função Split(Nodo N)

- Alocar as *M+1* entradas de N nas diversas distribuições segundo o valor do lado inferior no eixo/dimensão escolhida. Efectuar o mesmo procedimento segundo o valor do seu lado superior.
- Para todas as distribuições criadas, calcular a soma dos perímetros dos 2 grupos gerados em cada uma.
- Efectuar o mesmo procedimento para todos os eixos/dimensões
- Escolher o eixo que apresente o menor valor da soma dos perímetros.
- Escolher a distribuição que apresente menor valor de sobreposição entre os grupos formados. Em caso de empate, escolher a distribuição que tenha um menor valor total de área ocupada pelo seus grupos.
- Após escolhido o eixo e a distribuição, alocar as entradas nos novos nodos

Algoritmo 6 – *Splitting* na R*-Tree

Nos testes realizados em [BKSS90], ficou provado que a R*-Tree possui melhores performances quando a definição de *m* equivale a 40% de *M*. Para além disto, a R*-Tree é perfeitamente adaptável a qualquer distribuição de dados, apresentando melhores performances que a sua predecessora. Apesar de obter avanços significativos relativamente à R-Tree, a R*-Tree é uma estrutura mais complexa do ponto de vista funcional, além disto a sua proximidade às propriedades da R-Tree, nomeadamente, a ocorrência de sobreposição de regiões implica uma degradação de desempenho para ambientes de média/alta dimensionalidade.

2.4 Hilbert R-Tree

A Hilbert R-Tree[KF94] é uma variante baseada numa abordagem designada por *space filling curves*. Esta técnica consiste numa ligação sobre todos os dados de uma estrutura multidimensional sem nunca os repetir uma única. O objectivo foi usar esta técnica de forma a impor uma ordem específica nas regiões multidimensionais que compõem o índice. Desta forma, tal como referido em [MNPT05], é possível obter um controlo maior sobre a estrutura. No caso de haver uma situação de *overflow* de um nodo há informação sobre os nodos mais próximos, podendo, desta forma, haver maior facilidade na reinserção de algumas de entradas. *Splitting* só ocorre em último caso, isto é, quando há ocorrência de *overflow* de um nodo e os nodos mais próximos já estão totalmente preenchidos. A *space filling curve* usada nesta estrutura foi a *Hilbert Curve*, sendo conhecida por obter bons agrupamentos de dados. Em [FR89, FK94] são demonstrados os seus processos de cálculo.

A Hilbert R-Tree possui diferenças estruturais relativamente à R-Tree. Tal é devido à adequação aos *Hilbert Values*. Os nodos-superiores passam a conter uma informação adicional, o LHV (*Largest Hilbert Value*), passando a possuir registos do tipo (*LHV*, *I* , *apt*) em que *I* representa o *MBR* associado, *apt* é o apontador para o nodo-filho e *LHV* idenfica o maior *Hilbert Value* idenficado entre os dados contidos por este nodo. Cada *Hilbert Value* é relativo ao centro de cada elemento.

Na Figura 10, temos um exemplo bidimensional de uma Hilbert R-Tree retirado de [KF94], enquanto que na Figura 11, podemos observar a sua representação hierárquica, bem como a forma como esta é armazenada em disco.

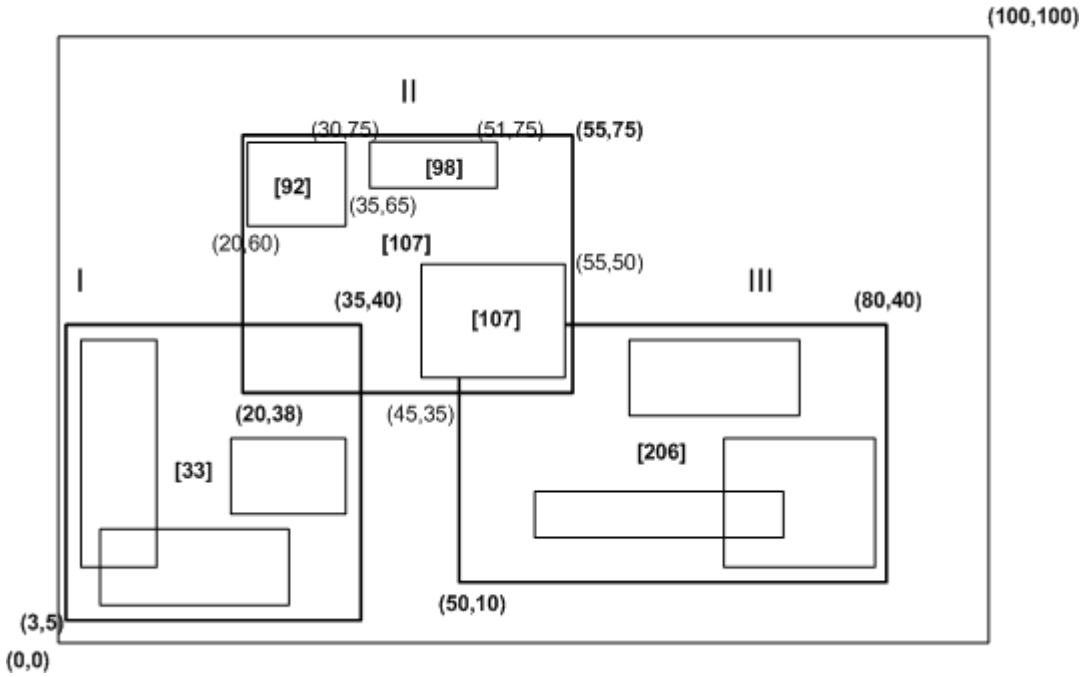

Figura 10 – Representação Bidimensional de uma Hilbert R-Tree [KF94]

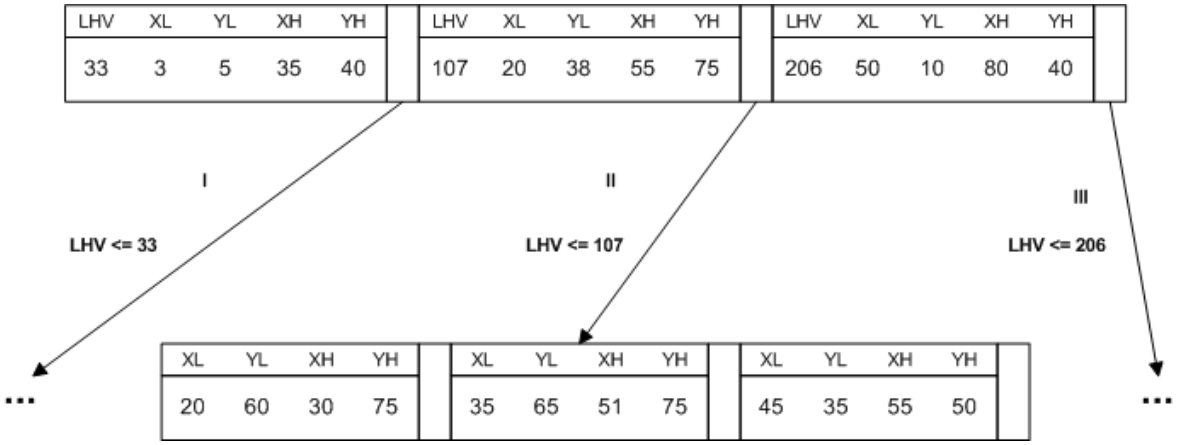

Figura 11 – Disposição Hierárquica e Informação Registada em Disco da Hilbert R-Tree[KF94]

Em relação as operações base, a Hilbert R-Tree apresenta algumas diferenças como é natural. Apenas a operação de procura apresenta os mesmos moldes da R-Tree. A inserção é baseada nos *Hilbert Values*. Quando o entrada é inserida, é calculado o seu Hilbert Value, de seguida é conduzida através dos níveis da árvore, sendo o factor de determina o seu trajecto o menor *LHV* que seja superior que o seu *Hilbert Value*. O processo de inserção está descrito no Algoritmo 7.

```
Função inserção(Nodo R, Elemento X)
  • If R é um nodo-folha
    ▪ If R tem espaço para conter X
        o Adicionar X a R de acordo com o seu Hilbert Value
    ▪ Else
        o Tratar situação de Overflow
        o Efectuar as mudanças nos níveis superiores
  • Else
    ▪ Escolher a entrada de R que possua o menor LHV que seja superior ao Hilbert Value de X
    ▪ Invocar inserção(entrada.apt, X)
```

Algoritmo 7 – Inserção de um elemento na Hilbert R-Tree

Os cenários de *Overflow* são tratados como descritos acima de forma resumida. Sempre que um elemento é direccionado para um nodo totalmente preenchido, este só entra em *Splitting* caso um dos seus nodos mais próximos esteja totalmente preenchido, caso contrário o novo elemento juntamente com as entradas do nodo preenchido são distribuídas entre este último e os seus nodos mais próximos mediante a definição do número de nodos mais próximos. Este situação está directamente ligada à política de *Splitting*. Esta operação pode ser efectuada sempre que existe *Overflow* em 1 nodo, originando 2 novos nodos,

sendo assim designada de política 1-2, ou pode ser definida com outros valores, por exemplo, n-(n+1), o que implica que só quando n nodos estão totalmente preenchidos se efectua *Splitting*, gerando n+1 novos nodos.

Relativamente às operações de remoção, a influência desta política dos nodos mais próximos está presente. O Algoritmo 8 demonstra o processo iterativo desta operação

```
Função remoção(Nodo R, Elemento X)
  • If R é um nodo-folha
      ▪ If X presente nas entradas de R
          o Remover X
          o If R encontra-se numa situação de Underflow
              ❖ Distribuir entradas de R por R e os
                seus nodos mais próximos
          o Ajustar LHV e MBRs dos níveis superiores
  • Else
      ▪ Efectuar uma procura de X tendo em conta a
        sobreposição deste com os MBRs das entradas de R.
        Descer na estrutura caso se confirme esta condição
```

Algoritmo 8 – Remoção de um elemento na Hilbert R-Tree

A Hilbert R-Tree, segundo [KF94], obteve melhores desempenhos nas diversas operações base comparativamente às estruturas até aqui demonstradas. A política de *Splitting* adoptada e que mostrou obter melhores desempenhos foi a 2-3. Apesar da *Hilbert Curve* poder ser extendida para alta dimensionalidade, os desempenhos da Hilbert R-Tree tendem a degradar-se progressivamente, seguindo a mesma linha que as estruturas anteriores.

2.5 X-Tree

A X-Tree (eXtended Node Tree) [BKK01] é uma das mais importantes variantes da R-Tree. Tal facto deve-se à sua adequação para ambientes de média/alta dimensionalidade. Este índice acenta na mesma ideologia da R^+-Tree relativamente à sobreposição de *MBRs* no mesmo nível. No entanto, tanto funcional como estruturalmente são bem distintas. A organização da X-Tree não só evita sobreposição no mesmo nível da estrutura como também evita as operações de *Splitting*. Ao invés, torna a sua estrutura menos rígida comparativamente à R-Tree e demais variantes. Como o seu próprio nome indica, esta estrutura possui nodos extendidos designados supernodos. Este tipo de nodos apenas existem nos níveis superiores da estrutura, e são essenciais para garantirem a ausência de sobreposição dentro de um determinado nível. A Figura 12 é uma ilustração baseada na presente em [BKK01] e revela-nos a estrutura hierárquica de uma X-Tree.

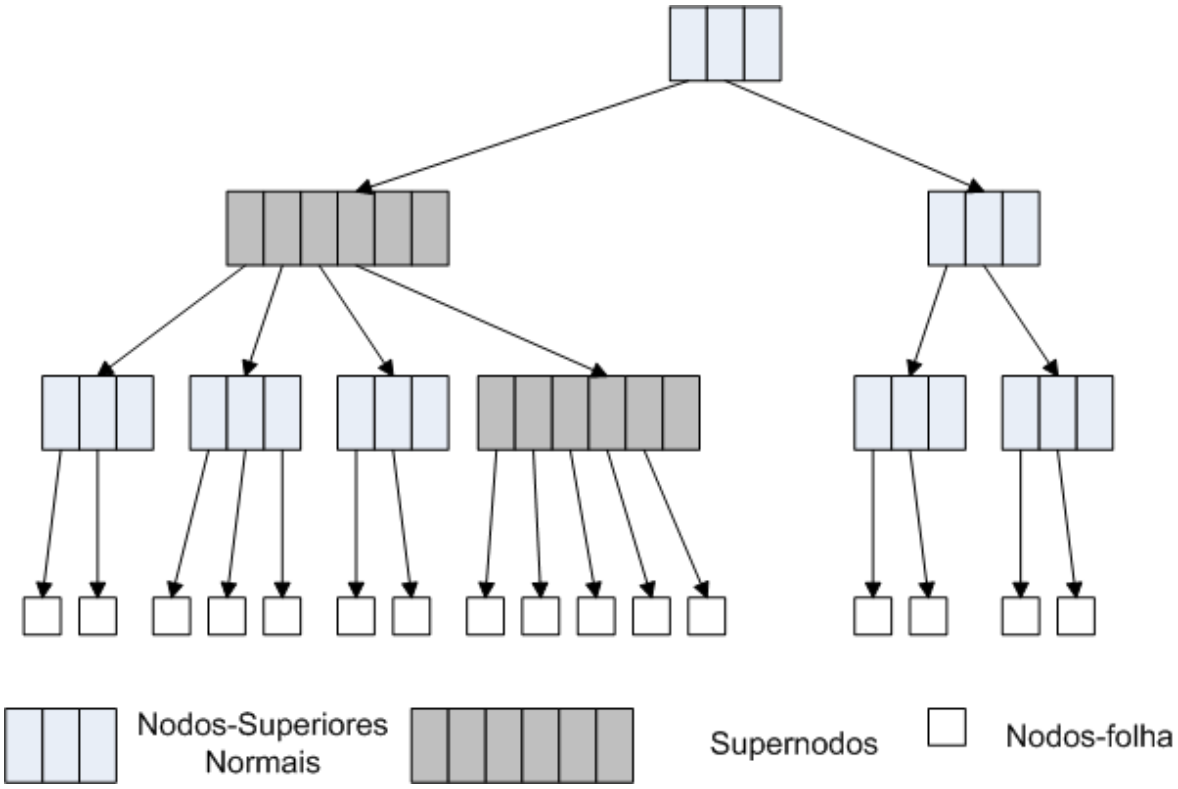

Figura 12 – Representação da Estrutura de uma X-Tree [BKK01]

Tal como referido em [BKK01], a X-Tree é uma estrutura hierárquica, mas também similar a um *array*. Tal facto deve-se às condições dos dados, nomeadamente a sua distribuição, assim como a dimensionalidade presente. Quando estamos perante ambientes de elevada dimensionalidade a probabilidade de haver sobreposição de *MBRs* é bastante mais elevada, contráriamente aos cenários de baixa dimensionalidade. Para este último caso, a necessidade de se criar supernodos é bastante mais baixa o que permite à estrutura obter um maior número de níveis, aumentando a sua altura, ou seja, adoptando uma organização hierárquica mais visível. Na Figura 13 é apresentado uma ilustração simbólica,

apresentada em [BKK01], sobre o desenvolvimento da estrutura à medida que a dimensionalidade aumenta.

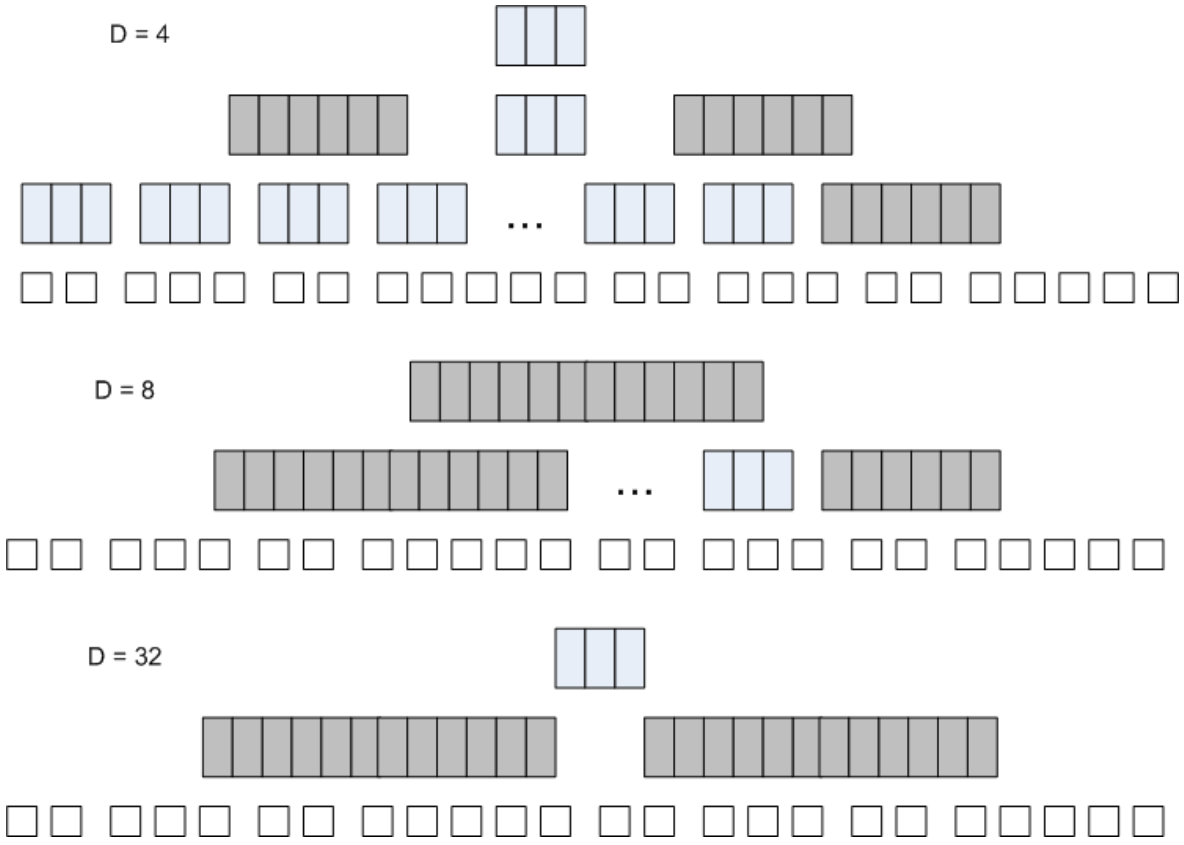

Figura 13 – Representação da Evolução da X-Tree com o Aumento da Dimensionalidade

Os supernodos não têm uma capacidade máxima estabelecida, sendo extendidos à medida que seja necessário.

As operações base na X-Tree são efectuadas de forma semelhante à R*-Tree. Apenas os processos relativos à inserção de um novo elemento, assim

como a gestão do particionamento do espaço possuem contornos distintos, o que é natural devido às características ímpares desta estrutura.

Antes de analisar o processo de inserção é necessário demonstrar as operações de *Splitting* e a forma como os supernodos estão directamente ligados a esta. Para além da introdução de supernodos na sua estrutura como elemento inovador neste género de índices, a X-Tree apresenta algumas diferenças na estrutura dos seus nodos-superiores. Estes contêm um campo indicador dos eixos dimensionais sobre os quais as suas entradas sofreram operações de *Splitting*, designado por *Split History*. Este campo é essencial sempre que existe uma sobreposição de *MBRs*, pois em ambos está presente a informação que permite averiguar se podem ser particionados em novas regiões sem provocar sobreposição ou, pelo menos, se esse factor for inferior à constante MAX_OVERLAP. Este valor é obtido através da seguinte equação:

$$MaxO = \frac{T_{TR} + T_{CPU}}{T_{IO} + T_{TR} + T_{CPU}}$$

As variáveis em causa representam o tempo de acesso a uma página do disco (T_{IO}), o tempo necessário para transferir um bloco do disco para a memória principal (T_{TR}) e o tempo de processamento de um bloco (T_{CPU}).

Na Figura 14 [BKK01] podemos observar uma árvore de *Split*, que nos indica a evolução das operações de *Splitting* num determinado nodo, nomeadamente os *MBRs* envolvidos e o eixo dimensional que motivou a operação. Por análise da respectiva figura, podemos observar que o *MBR* A já foi particionado pela dimensão 2, logo no seu campo *Split History*, essa informação está lá registada. De forma mais drástica, podemos observar que o *MBR* B já foi alvo de 3 operações de *Splitting* em 3 diferentes eixos dimensionais, nomeadamente 2, 5 e 1.

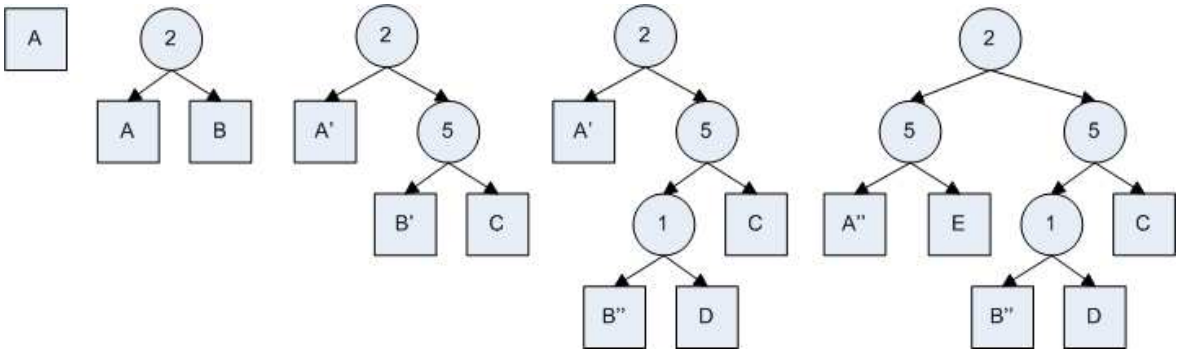

Figura 14 – Representação de uma *Split Tree* simbólica numa X-Tree

A análise desta informação será vital para a execução de uma operação de Splitting ou para a extensão de um nodo-superior ou supernodo. O Algoritmo 9 indica-nos se é ou não possível efectuar *Splitting* num nodo da X-Tree

```
Bool averiguar_splitting(MBR inicial)
  • Tentar efectuar uma técnica de splitting baseada na
    optimização do espaço, "à lá R*-Tree" (por exemplo)
  • Testar a sobreposição entre 2 novos MBRs gerados
  • If factor de sobreposição superior a MAX_OVERLAP
      ▪ Tentar nova técnica de Splitting, efectuar
        overlap-minimal split
      ▪ If número de elementos em cada grupo for inferior
        à capacidade mínima dos nodos, m
          o Splitting não é efectuado, returnar Falso
  • Returnar Verdadeiro
```

Algoritmo 9 – Testar Ocorrência de *Splitting*

O algoritmo relativo ao *overlap-minimal split* baseia-se na tentativa de encontrar uma distribuição, em um qualquer eixo dimensional, que agrupe os dados em 2 *MBRs* cujo o factor de sobreposição seja reduzido. Neste sentido, a informação presente em todas as entradas do *MBR* alvo, ou seja, o conteúdo do campo *Split History* é essencial para indicar possíveis soluções óptimas. Em [BKK01], é afirmado que se todas as entradas de um *MBR* já tiverem sido alvo de operações de *Splitting* segundo o mesmo eixo dimensional, então há garantia de haver uma distribuição que permita a não ocorrência de sobreposição de regiões após a referida operação. No entanto, não há garantia que os 2 grupos recém-formados contenham, no mínimo, *m* entradas cada um. Se se verificar que um dos grupos não contém o número mínimo de elementos contidos, então não há condições para se efectuar *Splitting*, visto que há violação da integridade da estrutura. Desta forma, quando há uma situação de *Overflow* nos níveis superiores e não há condições para se efectuar Splitting, então é necessário proceder-se a uma extensão do nodo em questão. Esta técnica é usada sempre em último recurso, pois há interesse em manter a hierarquia da X-Tree bem definida para optimizar os tempos de execução das diversas operações. O Algoritmo 10 indica-nos o processo de inserção de um novo elemento na X-Tree.

Função inserção(Nodo R, Elemento X)

- Percorrer a estrutura desde a raíz até ao nodo-folha onde o elemento X deve ser alocado
- If nodo-folha tem capacidade para alocar X
  - Adicionar X
- Else
  - Efectuar *Splitting*
  - If *Splitting* causar *Overflow* no nodo-pai
    - Averiguar condições para se efectuar *Splitting* no nodo-superior

Algoritmo 10 – Inserção de um elemento na X-Tree

- If *Splitting* permitido
  - ❖ Criar novo nodo-superior e distribuir as *M+1* entradas pelos 2 nodos-superiores
- Else
  - ❖ Criar Supernodo, ou seja, extender o nodo-pai. Caso o nodo-pai já seja um supernodo, também será alvo de extensão de capacidade

- Efectuar ajustamentos de *MBRs* a todos os nodos afectados

Algoritmo 10 (continuação) – Inserção de um elemento na X-Tree

A X-Tree revela-se como uma das estruturas multidimensionais hierárquicas mais promissoras. Embora as suas características contrastem mais com a R-Tree, comparativamente a outras variantes, a sua aptidão para ambientes de alta dimensionalidade permite-lhe obter uma notoriedade ímpar. Em relação á R*-Tree, a X-Tree segundo [BKK01], apresenta resultados relativamente semelhantes para ambientes de média/baixa dimensionalidade, mas apresenta desempenhos claramente mais favoráveis para os ambientes de maior dimensionalidade.

2.6 Outras Variantes

O Universo R-Tree é bastante vasto. Em [MNPT03] e [MNPT05] são analisadas inúmeras estruturas pertencentes a esta família. Nesta tese, optou-se por analisar algumas das mais importantes. No entanto, outras mais poderiam ser escolhidas, nomeadamente a recente Priority R-Tree [ABHY08], que garante um custo máximo no acesso ao disco nas operações de procura, a TV-Tree[LJF94], estrutura vocacionada para alta dimensionalidade, onde há uma classificação

sobre a importância dos atributos dimensionais para a definição da sua orgânica, a Sphere-Tree que baseia a sua estrutura numa hierarquia de esferas multidimensionais (*Minimum bounding Spheres*) em vez de *MBRs*, entre outras. Para além destas variantes, outras mais foram criadas com vista a optimizar a indexação de dados previamente definidos. Estas variantes são definidas como estáticas, visto que as suas estruturas indexam um volume de dados tratados *à priori*. Desta forma, estes índices não se modificam futuramente, daí a sua designação. Como exemplo deste tipo de índices temos a Packed R-Tree, Hilbert Packed R-Tree, entre outras.

A importância da R-Tree extende-se por diversas áreas, nomeadamente geografia, multimédia, Bases de Dados Espaciais, *Data Warehousing/OLAP*, *Mining*, etc. No capítulo seguinte, é analisada a importância desta estrutura no seio dos sistemas de processamento analítico (OLAP)

Capítulo 3

As R-Trees em Sistemas de Processamento Analítico

Os sistemas de suporte à decisão são, cada vez mais, aplicações de inegável utilidade para a definição de rumos e estratégias das empresas. A implantação destes sistemas é cada vez maior, o que constitui uma tendência ilucidativa da sensibilidade que o mundo empresarial/institucional demonstra pelas capacidades e vantagens inerentes a estas aplicações.

Os sistemas de processamento analítico, geralmente reconhecidos por sistemas OLAP (Online Analytical Processing), são parte integrante e essencial dos sistemas de suporte à decisão. Arquitecturalmente adjacentes ao DW – repositório central de informação relevante de negócio -, possuem a vital função de transformar os dados contidos neste, em informação passível de análise por parte dos utilizadores. Desta forma, o papel desempenhado por estes sistemas assume especial relevo ao garantir e providenciar uma mais clara visão de toda a actividade de negócio.

O objectivo central dos sistemas OLAP reside na conciliação entre 2 factores: o processamento da informação requerida e o tempo gasto na sua disponibilização. Se, por um lado, estes sistemas se encontram dotados de inúmeras operações – filtragens, agregações e combinações entre estas – que garantem ao utilizador obter toda a informação necessária para os seus processos de tomada de decisão, por outro, e não menos importante, o tempo necessário para a disponibilização dessa mesma informação é o factor chave para credibilização destas aplicações. Dentro deste contexto, e tendo em conta o elevado volume de informação característico destas aplicações, surgiram inúmeras técnicas para optimizar os tempos na obtenção dos mais variados pedidos dos utilizadores – *queries ad-hoc.* O uso de estruturas de indexação multidimensional de dados é, apenas, uma das possíveis abordagens tomadas neste sentido.

A R-Tree, como já referido nos capítulos anteriores, surge como um dos mais carismáticos e importantes índices multidimensionais. A sua orgânica e modo de funcionamento conferem-lhe atributos excepcionais para a organização de dados multidimensionais. Características como o balanceamento e a adequação à distribuição espacial dos dados permitem-lhe obter uma excelente aptidão para acções de procura, principalmente no caso das *range/window queries.*

A importância desta estrutura fomentou o interesse da comunidade científica. O aparecimento de diversas variantes que procuraram suprir alguns defeitos e desvantagens foi, desta forma, um acontecimento natural. No entanto, e apesar de ter alcançado a notoriedade em várias áreas, incluindo as bases de dados, a sua proeminência nos sistemas de processamento analítico é, de certa forma, ténue. A degradação do seu desempenho na alta dimensionalidade é um factor limitativo que poderá ser visto como uma das causas. Não obstante, as vantagens demonstradas relativamente às outras técnicas de optimização, usualmente presentes nestas aplicações, motivaram o surgimento de diversas

aproximações entre o universo R-Tree e os sistemas OLAP.[HSL01, JL98, KR98, PKZT01, RKR97, TPZ02]

A Ra*-Tree[J02, JL98], de entre as várias abordagens, foi a que obteve maior destaque. Isto deve-se, essencialmente, à adaptação das propriedades elementares das R-Tree aos dados e operações presentes em ambientes de processamento analítico. Tendo sido conceptualizada através de uma das mais emblemáticas e robustas variantes da R-Tree, a R*-Tree, teve na incorporação de técnicas de agregação de medidas a modificação que a tornou mais eficiente e adaptada para operações de procura de dados agregados - bastante requisitadas em sistemas OLAP.

Sendo assim, no presente capítulo é efectuada uma apresentação detalhada desta estrutura, visando, principalmente, a sua orgânica e comportamento nas diversas operações base, com especial ênfase, na sua adequação aos processos de procura nas *range queries*. Por último, e não menos importante, é demonstrado o impacto na organização e desempenho da Ra*-Tree, causado pelas variações de volume de dados, tamanho da região de procura, cardinalidade, dimensionalidade e distribuição dos dados - factores de bastante relevância para os sistemas de processamento analítico

3.1 Estrutura e Propriedades

A Ra*-Tree é uma estrutura de indexação de regiões multidimensionais de dados. À semelhança da R-Tree e R*-Tree, este índice mantém inalterável determinadas características estruturais. A organização dos nodos e os requisitos associados às suas capacidades de armazenamento são a herança mantida pela Ra*-Tree. Desta forma, não existem alterações de relevo relativamente à tipologia dos nodos, continuando a haver uma distinção entre 2 tipos de nodos, nodos-superiores e nodos-folha. Os nodos-folha são responsáveis pela indexação e

organização dos dados, enquanto os nodos-superiores indexam os nodos-folha no espaço multidimensional de acordo com as políticas de optimização do espaco da R*-Tree. A capacidade destes nodos, em sintonia com as normas da R*-Tree, é variável segundo uma gama de valores, M_s e m_s, representativos das capacidades máxima e mínima dos nodos-superiores respectivamente, e M_d e m_d, representativos das capacidades máxima e mínima dos nodos-folha respectivamente.

Os parâmetros M_s e M_d são atribuídos consoante o tamanho de página do disco, enquanto m_s e m_d variam entre 2 e metade da capacidade máxima. Sendo assim, estão sujeitos às seguintes condições:

$$2 \leq ms \leq \frac{Ms}{2}$$

$$2 \leq md \leq \frac{Md}{2}$$

Atendendo a estes factores, a Ra*-Tree é uma estrutura que pode ser catalogada como uma extensão da R*-Tree. Este facto é relevante para o conjunto de noções que se apresentam de seguida.

Tal como a sua predecessora, a Ra*-Tree, contém as seguintes características:

- A raíz possui pelo menos 2 filhos, exceptuando se estiver presente no nível 0, sendo assim um nodo-folha.
- Todos os nodos-folha, exceptuando a raíz, contém entre m_d e M_d entradas.
- Todos os nodos-superiores, exceptuando a raíz, contém entre m_s e M_s entradas.

- A estrutura é totalmente balanceada, implicando que os nodos-folha se encontrem no mesmo nível.

As modificações estruturais efectuadas à R*-Tree, e que motivaram o surgimento da Ra*-Tree, são relativas à composição dos seus nodos superiores. A adição de valores obtidos através da aplicação de funções de agregação foi a técnica usada para obter uma melhor adaptação ao tipo de dados e operações características dos sistemas OLAP, com especial destaque nas *range queries*.

Em [JL98] a composição dos nodos é definida da seguinte forma:

> ***nodo-superior* –** possui entradas do tipo (*apt,* ag, *I*), em que *apt* é o apontador para o nodo filho(nodo-superior ou nodo-folha), *ag* é o valor obtido após a aplicação de uma função de agregação[1] aos dados contidos no nodo filho, e, por último, *I* é a representação da região multidimensional(*MBR*) que envolve todos os registos do nodo filho.
>
> ***Nodo-folha* –** possui entradas do tipo (*apt_o*, *I*), em que *apt_o* é o apontador para a posição do registo na BD e *I* é a representação da região multidimensional que envolve cada registo, indicando a sua localização no espaço. Atendendo ao tipo de informação armazenada por estes nodos, em [JL98] são designados por *data nodes.*

Na figura 15 pode-se observar uma representação simbólica da estrutura da Ra*-Tree. Os dados estão contidos nos nodos-folha e representados por [a,b,c,d,e,f,g], enquanto os nodos-superiores contêm a representação das regiões multidimensionais que envolvem os nodos filho, através de R1 e R2, e os valores agregados, obtidos pela aplicação das funções de agregação $f_{(a,b,c)}$ e $f_{(d,e,f,g)}$. Os

valores representados para a capacidade dos nodos são os seguintes: $M_s = M_d = 5$ e $m_s = m_d = 2$.

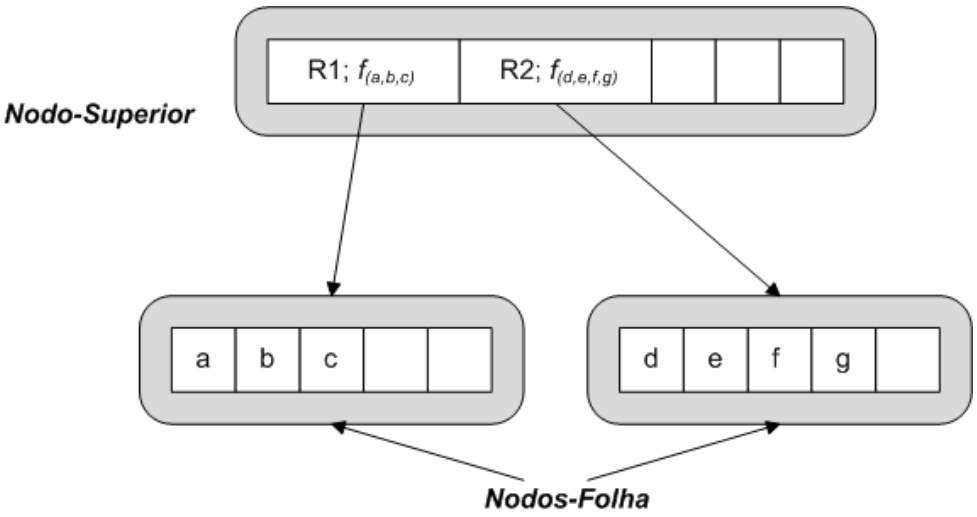

Figura 15 – Representação da orgânica de uma Ra*-Tree

A presença de similaridades entre a Ra*-Tree e a R*-Tree não se reduz à orgânica e propriedades base. A essência das operações convencionais da R*-Tree (inserção, remoção e procura), as técnicas de particionamento espacial e a evolução da estrutura foram adoptadas pela Ra*-Tree. Sendo assim, apenas os processos relativos à manipulação de dados, mais concretamente, o tratamento das agregações, realizados nas diversas operações serão descritos e aprofundados mais adiante.

[1] As funções distributivas(ex: sum, count) e algébricas(ex: average) [GCB+97] são as funções de agregação representadas nesta tese por serem, actualmente, as únicas suportadas por esta estrutura[J02, JL98].

3.2 Algoritmos

Os sistemas OLAP são aplicações que, tradicionalmente, interagem com um enorme volume de dados. Uma característica bem patente nestas aplicações é a sua periodicidade no carregamento de informação. Este factor é deveras importante, pois implica que haja uma actualização das suas estruturas de indexação. Estas, independentemente da classe que pertençam e das características que possuam, baseiam a sua funcionalidade em 3 tipos de operações bem distinctas, designadamente, inserção, remoção e procura. A Ra*-Tree, dentro deste contexto, não é excepção.

Como referido anteriormente, a Ra*-Tree herdou os príncipios e modo funcionamento da R*-Tree. Assim, as operações de inserção, remoção e procura, efectuadas por esta estrutura, seguem os critérios assumidas pela sua predecessora. Desta forma, tal como em [J02], apenas os processos relativos ao tratamento das funções de agregação nas diversas operações, serão descritos em pormenor.

3.2.1 Inserção

Para obter uma aproximação à tipologia de dados dos sistemas de processamento analítico, a Ra*-Tree dotou as suas operações de inserção com mecanismos que permitem agregações de dados nos seus nodos-superiores. Esta nova funcionalidade confere uma maior robustez a esta estrutura, tornando-a computacionalmente mais exigente. Todavia, será de enorme relevância para os processos de procura.

A inserção de um novo elemento na Ra*-Tree é tanto mais simples, quanto menos alterações provoca na organização da estrutura. O cenário mais complexo verifica-se quando uma inserção desencadeia operações de *splitting.* Esta

situação provoca modificações estruturais profundas, originando uma nova organização dos nodos nos níveis afectados. Desta forma, os mecanismos de agregação de dados tornam-se, computacionalmente, mais exigentes, pois terão de ser adaptados às alterações efectuadas durante a execução do processo de inserção. Em sentido contrário, a isenção de operações de *splitting* torna mais simples, não só, operação de inserção, como também a agregação dos novos dados.

Partindo da representação da Ra*-Tree na Figura 15 como estrutura inicial, representou-se, na Figura 16, um exemplo de uma inserção sem ocorrência de *splitting*, enquanto na Figura 17, aproveitando a nova configuração da figura anterior, podemos observar o caso oposto. Para cada um destes exemplos, é analisado o comportamento das funções de agregação.

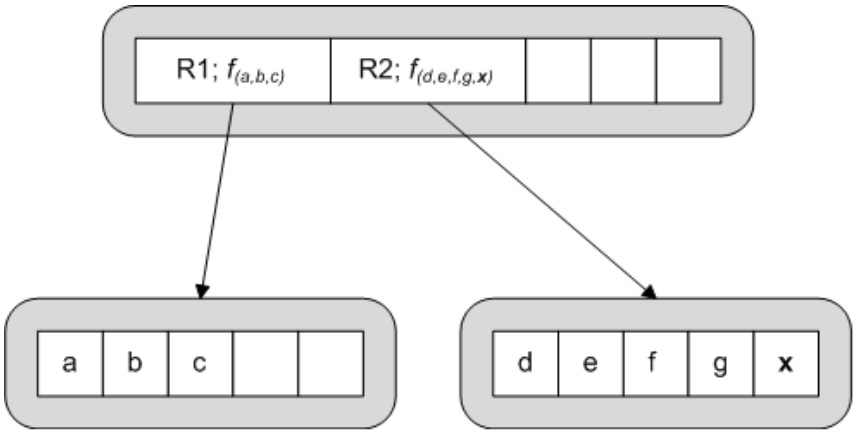

Figura 16 – Inserção sem ocorrência de *splitting*

Numa operação de inserção, o elemento a inserir percorre a Ra*-Tree desde a sua raíz até ao nodo-folha que o irá alojar, actualizando os valores agregados nos níveis superiores da árvore. Considere-se o seguinte exemplo:

Suponhamos que o elemento **x**, de acordo com a sua localização espacial, se encontra contido ou mais próximo da região multidimensional representada por R2. Neste caso, **x** é encaminhado para nodo-filho da entrada respectiva a R2, e o seu valor entra como parâmetro na função de agregação. Caso o respectivo nodo-filho fosse um nodo-superior, então haveria repetição do processo acima descrito, com **x** a entrar como parâmetro numa função de agregação referente a uma das entradas desse nodo. Como tal facto não se verifica, então **x** é alocado numa das vagas do nodo-folha.

Este processo, tal como descrito em [J02], ocorre para todas as funções distributivas(ex:sum, count, max, min)[GCB+97]. As funções algébricas(ex:average)[GCB+97], pelo facto de poderem ser obtidas através funções distributivas, também poderão ser calculadas.

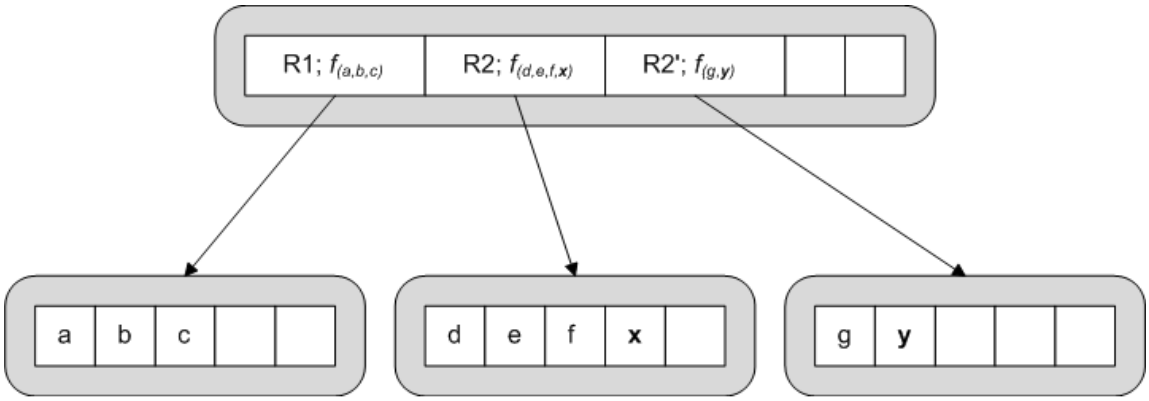

Figura 17 – Inserção com ocorrência de *splitting*

As operações de inserção mais complexas, as que motivam operações de *splitting*, seguem o procedimento acima descrito. Todavia, devido às alterações estruturais que ocorrem, originam, com naturalidade, um tempo de processamento adicional aos mecanismos de agregação de dados.

Suponhamos que, após a inserção de **x**, inserimos um elemento **y** que, devido à sua localização espacial, ficaria alocado no mesmo nodo-folha. Face às limitações na capacidade dos nodos, neste caso M_d, tal situação é insustentável, originando um *overflow* no nodo-folha. Desta forma, torna-se necessário efectuar uma operação de *splitting* para respeitar a integridade da estrutura. Estas operações, como descrito no capítulo anterior, implicam o surgimento de um novo nodo e a inclusão de uma nova entrada no nodo pai. O cenário mais drástico surge quando uma operação de *splitting*, ocorrida num determinado nível, origina novas operações de splitting em níveis superiores.

Para o exemplo em causa, supõe-se que a entrada respectiva a R2 contém um nodo filho que indexa d, e, f e **x**, enquanto que a nova entrada, R2', aponta para o novo nodo que contém g e **y**. Sendo assim, são originados 2 novos dados agregados resultantes das funções $f_{(d,e,f,x)}$ e $f_{(g,y)}$. Desta forma, é necessário haver um tempo de processamento adicional devido ao cálculo destes novos valores. Quando há ausência de operações de splitting, os valores já se encontram devidamente calculados, sendo apenas actualizados com a inclusão do novo elemento. Em suma, podemos afirmar que, quanto menores forem as modificações geradas na estrutura da Ra*-Tree durante uma operação de inserção, menor é o custo computacional associado aos cálculo das agregações, independentemente das funções usadas.

3.2.2 Remoção

A remoção de dados nos sistemas de *Data Warehousing* não é uma operação usual, antes pelo contrário. Atendendo ao objectivo central dos sistemas de suporte à decisão, facilmente se compreende que este tipo de operações sejam bastantes raras. As situações que as originam são muito excepcionais e, normalmente, associadas a remoções em bloco, *chunks* de dados, motivadas por limitações de espaço em disco, o que se torna cada vez mais raro, ou por decisões tomadas pelas empresas/instituições, onde há o descarte de informação, geralmente antiga, que deixou de ser relevante para os processos de tomada de decisão. Independentemente da relevância desta operação, iremos analisar o seu comportamento na Ra*-Tree e em que medida os mecanismos de agregação são afectados.

Tal como acontece nas inserções, também nas remoções há diferenças no custo computacional associado aos mecanismos de agregação. Tal facto deve-se à ocorrência de modificações estruturais de maior ou menor dimensão. O cenário mais complexo verifica-se na presença de um *underflow* num nodo. Esta situação ocorre quando um nodo contém menos entradas que a sua capacidade mínima, motivando, assim, a sua junção com outro nodo do mesmo nível e pertencente ao mesmo nodo pai. Desta forma, é eliminada uma entrada no nodo pai, o que poderá, eventualmente, provocar nova situação de *underflow*, aumentando assim a complexidade na reestruturação da árvore.

Para além deste cenário, também o comportamento dos processos de agregação é diferente consoante o tipo de função em causa.

Na Figura 18, podemos observar uma remoção sem ocorrência de situações de *underflow*, enquanto que na Figura 19 se verifica o cenário oposto.

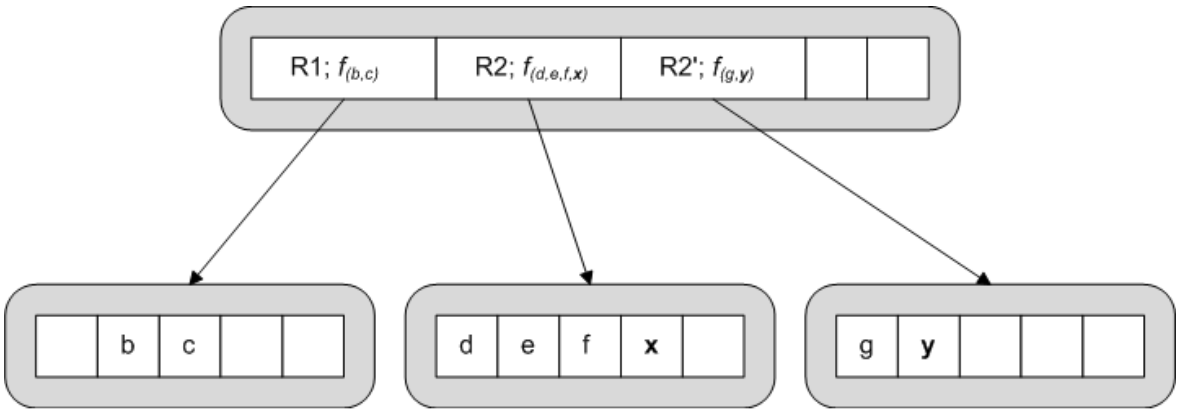

Figura 18 – Remoção sem ocorrência de *underflow*

A situação computacionalmente menos exigente numa operação de remoção ocorre quando não existem alterações estruturais nos níveis superiores da Ra*-Tree. Neste caso, o elemento é simplesmente removido do nodo-folha que o contém, não originando qualquer modificação no número de entradas do seu nodo pai. Relativamente aos processos de agregação, verifica-se um comportamento diferente consoante o tipo de função agregadora. No caso das funções distributivas, [J02] faz uma divisão em 2 tipos, aditivas e não-aditivas. As funções distributivas aditivas(ex:sum, count) têm um mecanismo de processamento e actualização semelhante ao ocorrido nas inserções. Assim, à medida que se vão percorrendo os vários níveis da estrutura, as entradas que vão constituindo o caminho até ao elemento a remover, vão actualizando os seus valores agregados. As funções não-aditivas(ex:min, max) requerem um tempo e custo computacional maior, pois “necessitam de actualizar os valores afectados desde o nodo-folha até à raíz da estrutura”. O processamento das funções algébricas está sujeito às mesmas condições referidas na sub-secção anterior.

Na Figura 18 podemos observar que o elemento **a** foi removido, e a função agregadora que o tinha como parâmetro foi devidamente actualizada.

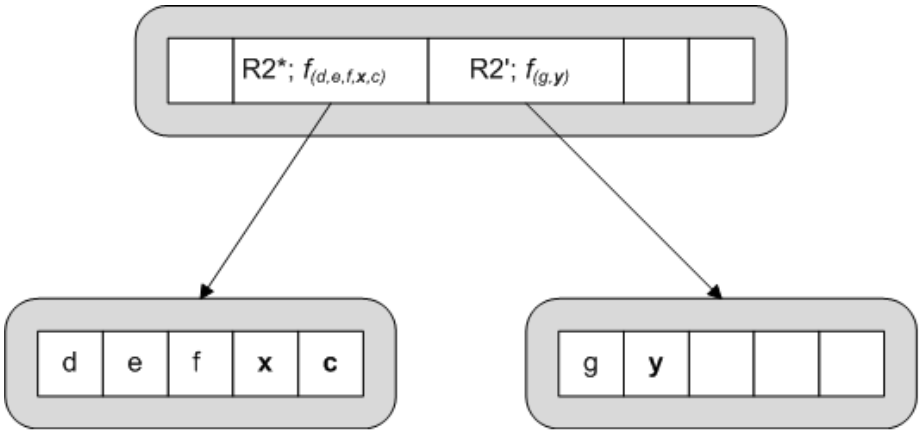

Figura 19 – Remoção com ocorrência de underflow

A situação evidenciada na Figura 19 permite-nos inferir a presença de um *underflow* no nodo-folha que continha o elemento **c**. O elemento b, ao ser eliminado da Ra*-Tree, implicou a existência de um nodo-folha com um número de elementos inferior à sua capacidade, m_d. Desta forma, motivou uma junção dos restantes elementos - **c** - com os registos de outro nodo-folha que, segundo os critérios da R*-Tree, necessitasse de menor incremento de volume multidimensional para o abranger. Para o exemplo ilustrado, supomos que o nodo-folha que contém d, e, f e x era o mais indicado. Estas ocorrências, como referido anteriormente, implicam reorganização na estrutura dos níveis superiores afectados. Na presente figura, podemos observar que a remoção de um nodo-folha motivou a retirada de uma entrada no nodo pai, a alteração de outro nodo-

folha e a modificação dos limites multidimensionais que envolvem esse mesmo nodo, R2*.

Este tipo de alterações estruturais geram custos computacionais superiores nos processos de agregação devido há necessidade de se calcular novamente os valores agregados nos nodos afectados. Como podemos observar, a função de agregação, presente na entrada representada por R2*, contém novos parâmentros, originados, neste caso, pela inclusão do elemento **c**. Por último, convém salientar que o comportamento das diversas funções agregadoras é similar, relativamente, aos cenários isentos de *underflow*.

3.2.3 Procura

O desempenho obtido na resolução de operações de procura é um dos factores mais importantes na avaliação de uma estrutura de indexação. Dentro deste contexto, e tendo em conta o tipo de dados presente nos ambientes de processamento analítico, a Ra*-Tree procura minimizar alguns defeitos evidentes no processamento destas operações na R*-Tree. A presença de sobreposição de regiões multidimensionais pode implicar a travessia de vários ramos da estrutura para obter a totalidade dos dados requisitados. Eventualmente, esta situação pode resultar numa degradação dos tempos de resposta, afectando deste modo, a eficiência desta estrutura. A inclusão de funções de agregação nos nodos-superiores foi a técnica usada na Ra*-Tree que permitiu uma melhor adaptação a uma das operações de procura mais usuais em sistemas OLAP, as *range queries* sobre dados agregados. Esta alteração visa minimizar a profundidade e o número de incursões sobre os nodos da estrutura, motivando desta forma, melhores tempos de resposta na obtenção deste tipo de informação.

A incorporação desta técnica encontra-se directamente ligada às relações topológicas [PSTE95,MNPT03] entre regiões multidimensionais. O processo de

procura da maioria das estruturas pertencentes ao universo R-Tree segue um modelo baseado na travessia da árvore desde a raíz até aos nodos-folha. O factor decisor é a intersecção entre a região multidimensional que representa a *range query* e os *MBRs* que envolvem os nodos dos vários níveis da estrutura. Sendo assim, sempre que o *MBR* de uma entrada de um nodo-superior intersecte os limites da *range query* o processo de procura avança para o nodo filho dessa entrada, e assim sucessivamente até se atingir o nível dos nodos-folha, devolvendo então os registos que respeitem esta condição. No entanto, não existe uma diferenciação entre as várias relações topológicas que podem caracterizar uma intersecção, isto é, uma sobreposição, parcial ou total, de um *MBR* nos limites da *range query* é tratada de igual forma. No caso da Ra*-Tree, esta distinção é de extrema importância. A presença de um *MBR* totalmente sobreposto nos limites da *range query* motiva um comportamento diferente no processo de procura. Se uma entrada de um nodo-superior possui um *MBR* contido nos domínios da *range query,* implica que toda a informação indexada pelo seu nodo filho também esteja contida. Portanto, não há necessidade de se aceder aos níveis inferiores, visto que o valor da função de agregação presente na respectiva entrada já é representativo de toda essa informação e pode ser utilizado. Para os casos em que não haja sobreposição total dos *MBRs* nos domínios da *range query*, o processo de procura é semelhante ao que acontece na R-Tree e outras demais variantes.

Nas seguintes figuras podemos observar um pequeno exemplo ilustrativo do processo de procura na Ra*-Tree. Posteriormente, é demonstrado e analisado o algoritmo de cálculo das funções de agregação.

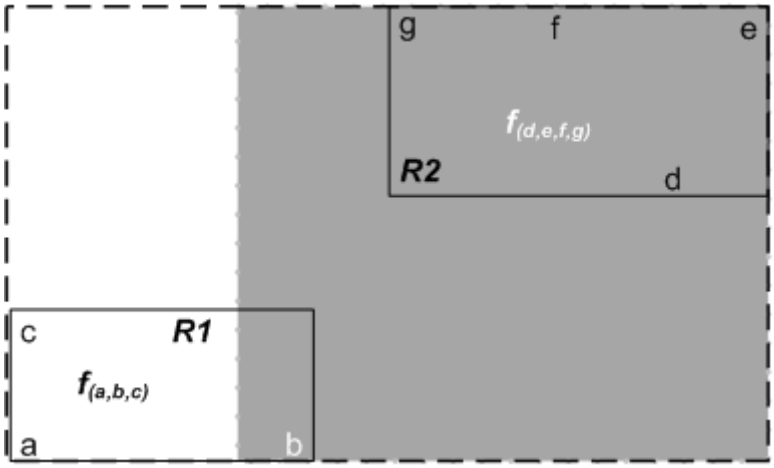

Figura 20 – Representação bidimensional dos valores agregados numa *Range Query*

Suponhamos que a representação em árvore da Figura 15 corresponde à distribuição e organização espacial visível na Figura 20. A parte sombreada a cinzento representa os limites de uma *range query* e a função agregadora representada por *f* é a função *sum*.

Como podemos observar na respectiva figura, a entrada no nodo-superior referente a R2 encontra-se totalmente contida nos domínios da procura, o que implica que todos os elementos do seu nodo filho, neste caso um nodo-folha, também o estejam. Sendo assim, se queremos obter a soma dos elementos contidos nos domínios da *range query*, não precisamos de aceder ao valor individual dos elementos contidos no nodo-folha, basta usar o valor previamente calculado na entrada de R2 e definido por $f_{(d,e,f,g)}$. No caso da entrada referente a R1, nota-se que a região multidimensional se encontra parcialmente sobreposta, o que implica que todos os elementos do nodo filho terão de ser confrontados com domínios da *range query*. Na Figura 20, observa-se que o elemento b é o único que está contido, logo o seu valor individual entra como parâmetro para a obtenção da informação requisitada. Portanto, como se pode observar, os valores

obtidos pela operação de procura estão salientados a branco, sendo que o resultado final é obtido através da seguinte expressão *sum(b , $sum_{(d,e,f,g)}$)*.

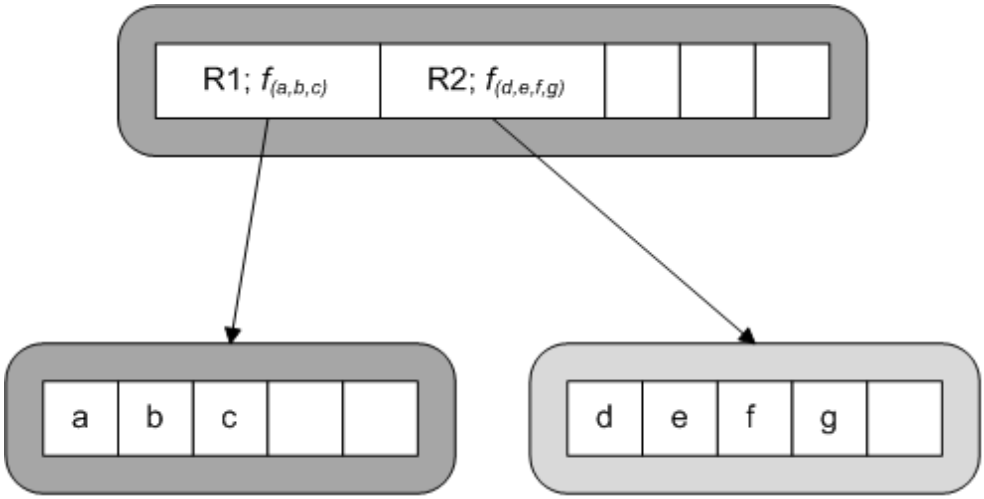

Figura 21 – Nodos acedidos pela *Range Query* da figura anterior

Na Figura 21 estão salientados a cinzento escuro os nodos acedidos pela *range query* da figura anterior. A informação mais relevante que daqui podemos retirar é o facto de não termos acedido a um dos nodos-folha, o que nos permite concluir que, para este tipo de operações de procura e comparativamente com as estruturas predecessoras, a Ra*-Tree revela-se bastante eficaz, diminuindo o número de acessos ao disco e, implicitamente, o tempo na obtenção dos dados. Outra ilação que podemos obter prende-se com a especificação da *range query*, isto é, dependendo do espaço multidimensional abrangido pela mesma, podemos obter desempenhos de maior ou menor destaque.

A análise dos desempenhos desta estrutura e a influência das diversas variáveis serão vistos com maior ênfase nas secções seguintes.

```
Função Calculo_Agregação ( Página_Disco pd, Região_Procura r)

  • Inicializar o Resultado
  • If pd é um nodo-folha
      ▪ Para todas as entradas de pd, verificar se estão
        contidas em r
          ❖ If entrada ⊆ r
                Resultado = f_agregadora(Resultado,
              valor_entrada)

  • Else
      ▪ Para todas as entradas de pd, verificar se estão
        contidas em r
          ❖ If MBR_entrada ⊆ r
                Resultado = f_agregadora(Resultado,
            valor_agregado_entrada)
          ❖ Else
            o  If MBR_entrada ∩ r ≠ 0
                Resultado = f_agregadora(Resultado,
                Calculo_Agregação( apontador_nodo-
                filho_entrada,q)
  • Returnar o Resultado
```

Algoritmo 11 – Função de Cálculo de Valores Agregados apresentada em [JL98]

O Algoritmo 11 acima descrito é uma adaptação do algoritmo apresentado em [JL98] e revela o processo de cálculo de uma função de agregação durante o processo de procura numa *range query*. Como a operação de procura é efectuada na forma *Top-Down*, isto é, desde a raíz até aos nodos-folha, a função de cálculo recebe como parâmetros a raíz da Ra*-Tree e a região multidimensional representativa da *range query*. Em primeira instância, verifica-se o tipo de nodo da raíz, visto que o cálculo das agregações tem um comportamento diferenciado entre os nodos-superiores e os nodos-folha. Caso a raíz seja um nodo-folha, então testa-se a inclusão de cada um dos registos nos limites da range query. Se

um registo estiver contido, então o seu valor é usado como parâmetro para o cálculo da função agregadora. Se a raíz for um nodo-superior, então o *MBR* de cada entrada é confrontado com os domínios da *range query*. Se um *MBR* estiver contido, então o valor agregado presente nessa entrada entra como parâmetro para o cálculo da função agregadora. Caso um *MBR* apenas intersecte parcialmente a região multidimensional da *range query*, a função é invocada recursivamente, recebendo o nodo-filho dessa entrada como parâmetro.

3.3 Organização da Ra*-Tree em OLAP

A grande prioridade num sistema de suporte à decisão consiste em fornecer aos seus utilizadores uma visão geral e representativa de toda a informação necessária para o cumprimento das suas funções. Dentro deste contexto, surgem os sistemas de processamento analítico como componente essencial para a obtenção desse objectivo. As operações de agregação, filtragem e a combinação destas, sobre um ou mais eixos de análise, são funcionalidades proporcionadas por estas aplicações, que facilitam os processos de procura de dados por parte dos utilizadores.

A Ra*-Tree surge como uma possível técnica para optimizar o desempenho dos processos de procura de dados. A sua orgânica, adaptada para ambientes multidimensionais, tornam-na capaz de responder a algumas necessidades dos sistemas OLAP. Desta forma, será demonstrado o processo de construção e modo de funcionamento deste índice perante diversas situações que, usualmente, fazem parte do dia-a-dia dos agentes de decisão.

Considere-se a seguinte representação simbólica de um modelo em estrela [KRT+98], presente na Figura 22.

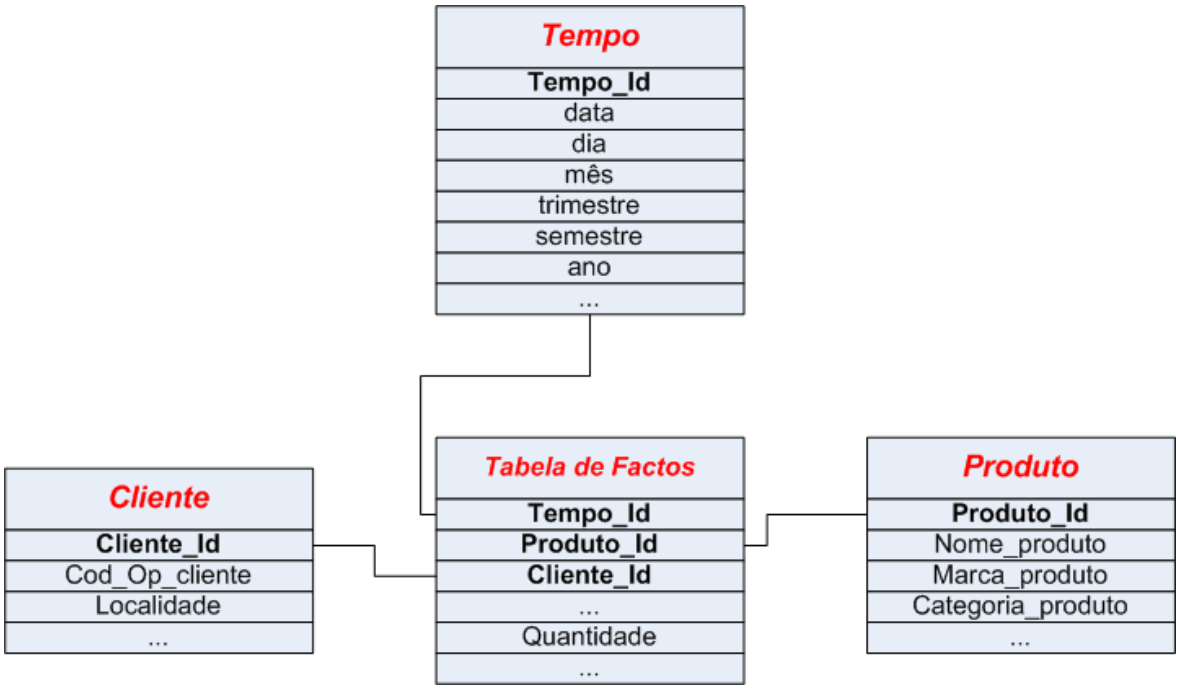

Figura 22 – Modelo em Estrela de um DW

O modelo em Estrela é um das representações mais usuais e consensuais para a definição de uma arquitectura de um DW. É constituído por uma tabela central – Tabela de Factos – que salvaguarda os registos da actividade de negócio e as medidas envolvidas, e tabelas desnormalizadas, adjacentes a esta, que contêm os dados relevantes de cada eixo de análise – Dimensões. Esta disposição das entidades lógicas tem em conta as necessidades dos utilizadores – menor tempo na obtenção de dados –, pois existe uma minimização das operações de join comparativamente aos modelos operacionais. A ligação entre as Dimensões e a Tabela de Factos é efectuada através de chaves únicas, em relações de 1:n. Na Figura 22, essa ligação é efectuada através dos atributos Tempo_Id, Produto_Id e Cliente_Id.

A necessidade de usar estruturas de indexação para acelerar processos de procura é vital em vários cenários e mediante diversas condições. Atributos bastante requisitados, tabelas com enorme volume de dados e atributos de elevada cardinalidade são apenas alguns exemplos que requerem a criação de índices. Desta forma, o gerenciamento e manutenção destes sistemas torna-se cada vez mais ardúo à medida que o volume de informação aumenta. A necessidade de se efectuar estudos e análises sobre o tipo de dados, o tipo de operações mais frequentes e o factor de crescimento da informação é fundamental para se potencializar o desempenho da aplicação mediante os requisitos dos utilizadores. Dentro desta filosofia, podemos incluir os índices multidimensionais como uma possível alternativa às técnicas de optimização em sistemas de processamento analítico. Estas estruturas possuem um comportamento diferenciado relativamente aos índices mais típicos nestas aplicações – B-Tree e Bitmaps. Um atributo dimensional representa um eixo do espaço multidimensional, sendo que cada um dos seus distintos registos representa uma posição específica dentro do respectivo eixo. Partindo deste conceito base, os dados são organizados dentro de uma estrutura baseada em regiões multidimensionais, cujo objectivo é proporcionar operações de procura rápidas, independentemente do número e das combinações entre atributos dimensionais. No caso da Ra*-Tree, não só esta capacidade está inerente, como também a materialização de dados agregados nos nodos-superiores lhe confere uma maior aptidão para responder de forma mais eficaz às operações de procura mais usuais nestes sistemas.

Um dos casos críticos num sistema de *Data Warehousing* prende-se com a optimização dos processos de procura sobre os dados da Tabela de Factos. Atendendo ao seu enorme volume de informação, a necessidade do uso de estruturas auxiliares que favoreçam o acesso aos dados é indispensável. Embora a tecnologia dos componentes físicos que suportam uma aplicação de Business Intelligence esteja em constante evolução, proporcionando uma cada vez maior

capacidade de armazenamento de dados e maior velocidade no acesso aos mesmos, a verdade é que, salvo determinadas situações, uma leitura sequencial dos dados contidos numa Tabela de Factos torna-se insustentável. Dentro deste contexto, e tomando como referência o modelo em Estrela acima descrito, encontra-se presente na Figura 23 uma pequena amostra de uma Tabela de Factos que será usada para exemplificar o funcionamento e construção de uma Ra*-Tree.

Tabela de Factos						
	...	Tempo_id	Produto_id	Cliente_id	Quantidade	
1	...	1	1	1	8	A
2	...	1	1	2	6	B
3	...	1	1	3	7	C
4	...	1	2	1	3	D
5	...	1	1	6	12	E
6	...	1	2	7	20	F
7	...	1	2	6	32	G
8	...	1	5	1	5	H
9	...	1	6	2	10	I
10	...	1	6	1	11	J
11	...	1	4	3	29	K
12	...	1	7	3	12	L
13	...	1	8	5	1	M
14	...	1	3	5	3	N
15	...	1	4	6	5	O
16	...	1	1	7	16	P
17	...	1	5	4	20	Q
18	...	1	7	5	9	R
19	...	1	8	7	10	S
20	...	1	7	9	20	T
21	...	1	1	8	5	U
22	...	1	6	10	8	V

Figura 23 – Tabela de Factos

Na Figura 23 podemos observar 3 atributos dimensionais, nomeadamente Tempo_Id, Produto_Id e Cliente_Id, e um atributo medida, representado pela Quantidade. Neste exemplo, supomos um caso simples e genérico, adequado a área de retalho, onde analisamos a quantidade de vendas de um determinado

produto. Para todos os efeitos, este exemplo é suficiente e ilustrativo para se demonstrar o comportamento de um índice desta natureza. Por motivos de mais fácil compreensão e demonstração da orgânica da Ra*-Tree, optou-se por atribuir o mesmo valor de Tempo_Id em todos os registos da Tabela de Factos, como se pode verificar na figura anterior. Sendo assim, na Figura 24, obtemos uma representação bidimensional dos dados, ou seja, uma projecção de uma vista do cubo – cubóide -, ao invés de uma representação tridimensional, que seria o mais esperado. Não obstante, a extensão das regiões espaciais que compõem a Ra*-Tree para ambientes com dimensionalidade igual ou superior a 3 é intuitiva. Desta forma, não temos rectângulos a involver os dados, mas sim paralelípedos/cubos, no caso tridimensional, e hyper-rectângulos em ambientes com dimensionalidade superior.

Como se pôde verificar na Figura 23, ao lado da Tabela de Factos está uma coluna composta pelas letras do alfabeto. Cada elemento dessa coluna identifica a posição espacial de um registo e a ordem pela qual foi inserido no índice. A estrutura da Ra*-Tree gerada com esses dados encontra-se representada na figura seguinte.

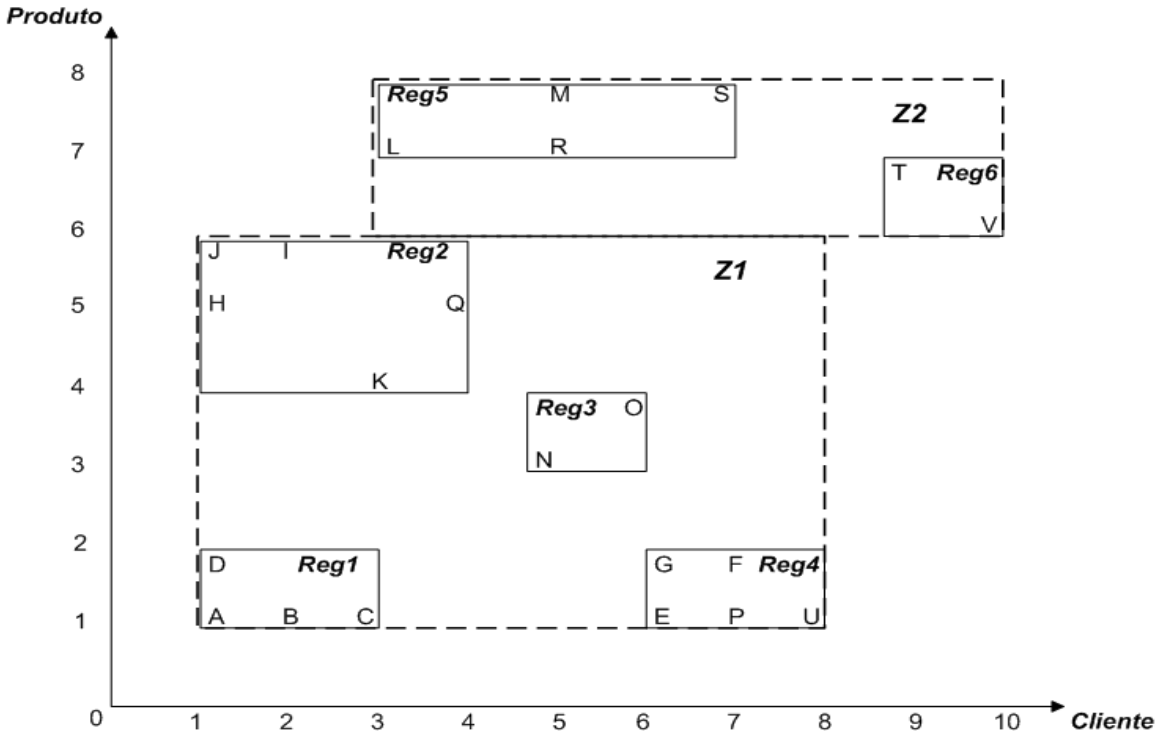

Figura 24 – Disposição Espacial dos Dados da Tabela de Factos numa Ra*-Tree

As regiões multidimensionais que envolvem os dados e compõem a estrutura da Ra*-Tree estão bem definidas nos diferentes níveis hierárquicos deste índice. Num nível intermédio, as regiões representadas por Reg1,Reg2,...Reg6, envolvem os dados presentes em cada nodo-folha, enquanto no nível mais elevado, a raíz, as regiões Z1 e Z2 envolvem as regiões anteriormente referidas. Esta organização espacial assume um papel fundamental em relação aos processos agregadores desta estrutura, visto que cada região identifica o conjunto de dados a agregar.

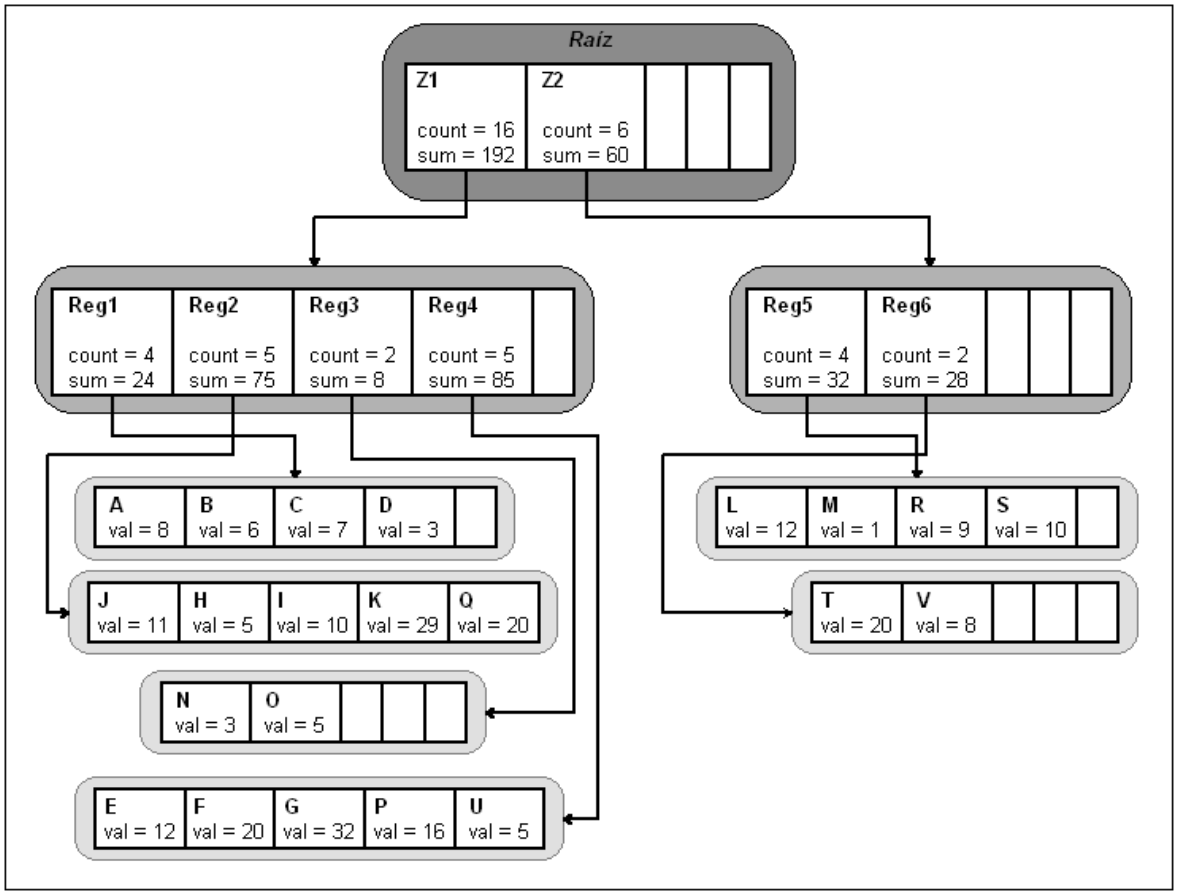

Figura 25 – Representação em Árvore da Ra*-Tree com os Dados da Tabela de Factos

Na Figura 25, observa-se com clareza, não só a hierarquia evidenciada na figura 24 como também o funcionamento dos mecanismos de agregação. Cada nível da estrutura agrega os dados do nível inferior, mantendo a integridade da informação indexada. Se repararmos, por exemplo, na região Reg5, presente no nível 1, observamos que agrega, através das funções sum e count, os valores presentes nos registos L,M,R e S. Por sua vez, a região Z2, presente no nível 2, efectua o mesmo processo às regiões que envolve, ou seja, Reg5 e Reg6. Sendo assim, se verificarmos os valores agregados presentes em Z2 – count = 6; sum =

60 – podemos concluir que estes valores seriam exactamente os mesmos caso as respectivas funções de agregação fossem aplicadas directamente ao conjunto de registos indexados – L,M,R,S,T e V. Em suma, a presença de informação agregada nos nodos superiores apresenta-se como um processo de filtragem de iterações e poupança de acessos ao disco.

3.4 Operações em OLAP

Os sistemas de processamento analítico garantem aos seus utilizadores um conjunto de operações que lhes permite visualizar e obter a informação que necessitam para os seus processos de tomada de decisão. Estas operações são, acima de tudo, processos de navegação sobre os dados do *DataCube*[GCB+97] que permitem aos utilizadores efectuar várias combinações entre estes. Neste sentido, destacam-se as operações de *slice and dice* , *roll up*, *drill down* e *pivoting*.

A actuação de cada uma destas operações está directamente ligada aos processos de agregação e selecção da informação. Embora com contornos distintos, juntas funcionam como um todo, permitindo uma eficaz exploração dos dados. A caracterização destas operações é efectuada da seguinte forma:

- *Slice and Dice* – esta operação é composta por 2 operações distintas, *Slicing* e *Dicing. Slicing* ocorre através da selecção de um dos registos de um determinado atributo dimensional, proporcionando, desta forma, uma diminuição na dimensionalidade do subcubo obtido. Esta operação é efectuada quando se pretende obter toda a informação sobre um determinado registo de um atributo dimensional.

Ex1: *Select Cliente_Id, Produto_Id, sum(Quantidade) from Tabela_Factos where Time_id = 1 group by Cliente_Id, Produto_Id*

Dicing possui um *"modus operandi"* similar à operação de slicing, no entanto, actua sobre os registos de 2 ou mais atributos dimensionais.

Ex2: *Select Cliente_Id, Produto_Id, sum(Quantidade) from Tabela_Factos where (Cliente_id = 3 or Cliente_Id = 4 or Cliente_Id = 5 or Cliente_Id = 6) and (Produto_Id = 3 or Produto_Id = 4) group by Cliente_Id, Produto_Id*

Embora possam ser efectuadas separadamente, a junção destas 2 operações é bastante usual entre os utilizadores dos sistemas OLAP, permitindo uma selecção da informação mais criteriosa e específica.

Ex3: *Select Cliente_Id, Produto_Id, sum(Quantidade) from Tabela_Factos where Time_Id = 1 and (Cliente_id = 3 or Cliente_Id = 4 or Cliente_Id = 5 or Cliente_Id = 6) and (Produto_Id = 3 or Produto_Id = 4) group by Cliente_Id, Produto_Id*

- *Roll Up* – esta operação está relacionada com a obtenção de dados agregados com menor grau de detalhe. Normalmente, está associada a agregações sobre atributos dimensionais presentes em níveis superiores de hierarquias definidas nas dimensões. No entanto, também, é efectuada quando se pretende agregar os dados através da redução da dimensionalidade do *DataCube*. O exemplo mais concreto é demonstrativo deste cenário ocorre quando pretendemos agregar todos os dados por nenhuma dimensão.

Ex4: *Select sum(Quantidade) from Tabela_Factos*

- *Drill Down* – esta operação tem um processo inverso à anterior. Por norma, parte de uma situação de agregação de menor detalhe para a obtenção de informação mais detalhada e específica. Partindo do exemplo anterior, basta-nos agregar os dados por uma dimensão para estarmos a efectuar *drill down*.

  Ex5: *Select Produto_Id, sum(Quantidade) from Tabela_Factos group by Produto_Id*

- *Pivoting* – esta operação está apenas relacionada com o modo de visualização da informação por parte do utilizador.

Os processos de agregação são parte integrante e funcional de um sistema de processamento analítico. Se observarmos com alguma atenção, podemos verificar que todas as operações permitidas se resumem á obtenção de dados agregados com maior ou menor detalhe. Na Figura 26 encontra-se um modelo, apresentado em [HRU96], que mostra os diferentes níveis de agregação de dados, assim como as relações e dependências entre os atributos dimensionais. O modelo foi construído tendo em conta os atributos dimensionais Cliente_Id (C), Produto_Id (P) e Tempo_Id (T) presentes na Tabela de Factos da Figura 23, sendo que no mesmo podemos observar a evolução dos processos de agregação, partindo de uma situação extrema – máxima agregação - representada por “ALL”, que significa a agregação de todos os dados, até ao cenário mais específico e detalhado com agregações menores, representado por CPT, demonstrativo dos valores agregados por cada atributo dimensional.

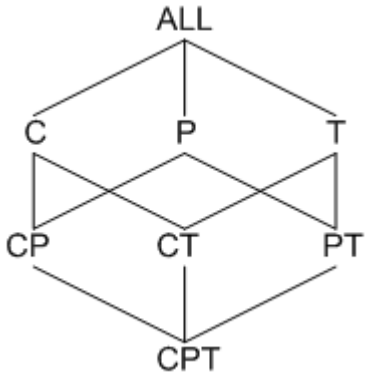

Figura 26 – *Lattice* de dependências referente às dimensões Cliente, Produto e Tempo

Feita esta ligeira introdução, nesta secção iremos demonstrar o comportamento de uma Ra*-Tree perante alguns exemplos de interrogações típicas destas operações, exceptuando a operação de pivoting, visto que apenas está relacionada com o modo de visualização da informação por parte dos utilizadores. Não obstante, o relevo desta estrutura consoante o nível de agregação requisitado também será alvo de análise e reflexão.

A Ra*-Tree, como estrutura de indexação, possuiu mecanismos muito definidos no exercício das suas operações base. No caso das acções de procura sobre *range queries* de valores agregados, o seu desempenho depende bastante da organização da própria estrutura, visto que a materialização de valores agregados nos nodos-superiores é dependente da definição das regiões multidimensionais que envolvem os dados.

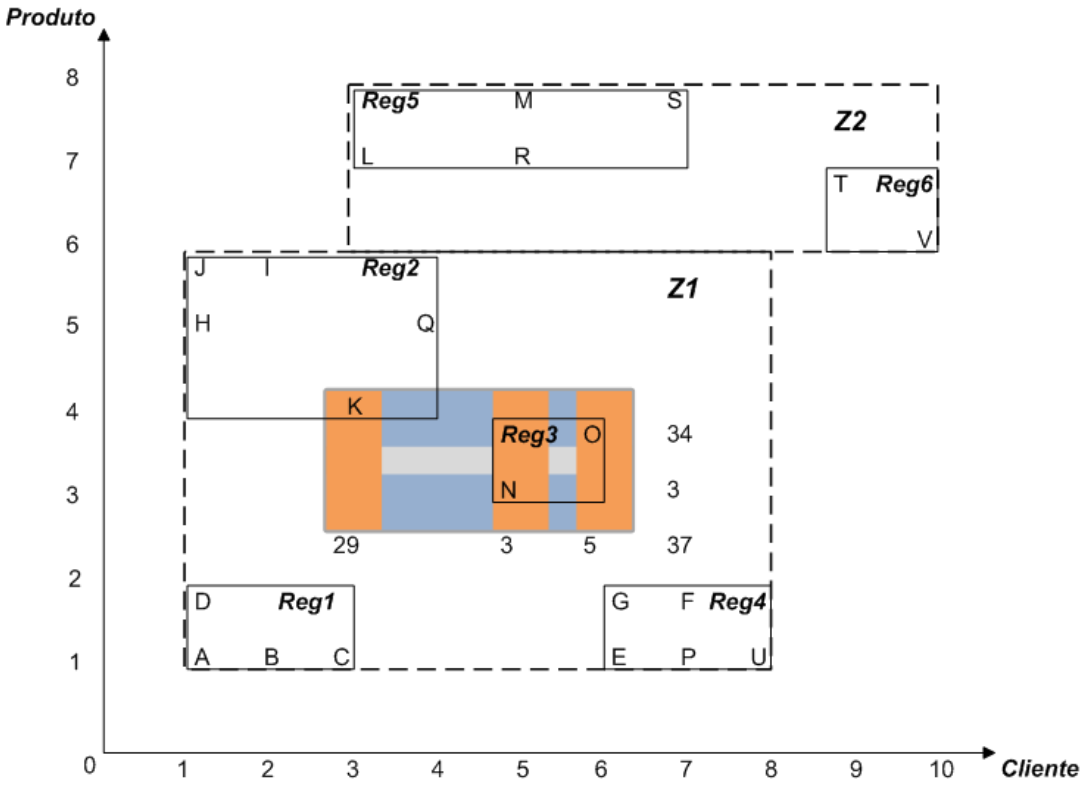

Figura 27 – Disposição dos dados obtidos pela *query* Ex3

Se observarmos a Figura 27, que nos indica as regiões de dados relativos à *query* exemplo da operação *slice and dice* (Ex3), facilmente concluímos que as regiões assinaladas a laranja correspondem à agregação pelos registos da dimensão Cliente, enquanto as regiões assinaladas a azul referem-se ás agregações sobre os registos da dimensão Produto. O rectângulo bidimensional que envolve estas regiões é representativa da agregação de todos os dados requisitados. Dentro deste contexto, e para o exemplo em concreto, podemos verificar que para as agregações por cada registo de um atributo dimensional existe a necessidade de se obterem os valores presentes nos nodos-folha, visto que cada uma destas regiões não sobrepõe algum nodo na sua totalidade. Em sentido contrário, verificamos que a região correspondente á agregação de todos

os dados compreende o nodo-folha envolvido pela região Reg3, o que significa que o valor agregado inerente a este, e presente numa das entradas do seu nodo-superior, poderá ser usado, gerando desta forma uma redução de acessos ao disco, visto que o nodo-folha em questão não necessita de ser acedido. Embora este seja um exemplo meramente simbólico, podemos realçar que quanto maior for a região multidimensional de dados requisitada por uma determinada operação, maior serão os ganhos relativos desta estrutura comparativamente à sua predecessora. Em termos práticos, isto significa que quanto maior o nível de agregação, maior será a diferença no número de acessos ao disco entre uma Ra*-Tree e a R*-Tree. Tal facto deve-se ao tamanho da região de dados abrangida por uma operação, o que se concluiu que quanto maior for a mesma, maior é a probabilidade de haver sobreposição, na sua totalidade, de *MBRs* – regiões multidimensionais que envolvem os vários nodos da estrutura. Não obstante, podemos, também, afirmar que, não só, é maior a probabilidade de conter um maior número de *MBRs*, como também, maior é a probabilidade disto acontecer em níveis mais altos da estrutura, originando menores incursões na hierarquia da estrutura durante o processo de procura, o que acentua, de forma significativa, as diferenças de performance entre estes 2 índices.

A eficácia desta estrutura para agregações desta amplitude pode ser demonstrada na Figura 28, onde está patente a região de dados requisitada pela *query* (Ex4), usada como exemplo na operação de *roll up* acima descrita. Neste cenário, podemos observar que a informação pretendida corresponde ao valor agregado de todos os dados da Tabela de Factos, sendo assim, a região de procura de informação cobre toda a estrutura da Ra*-Tree definida, e está identificada por um plano cinzento na respectiva figura. O pormenor mais ilucidativo, e importante, prende-se com as zonas acedidas no procura de obtenção de informação, estando as mesmas assinaladas a vermelho – Z1 e Z2. Tendo em conta que região de procura cobre todos os *MBRs*, apenas a raíz da estrutura necessita de ser percorrida, visto possuir nas suas entradas,

representadas por Z1 e Z2, os valores agregados de todos os dados indexados. Contrariamente à sua predecessora, R*-Tree, que necessitaria de atingir o último nível de forma a extrair a informação dos dados, a Ra*-Tree consegue garantir essa mesma informação poupando bastantes acessos ao disco e, consequentemente, tempo.

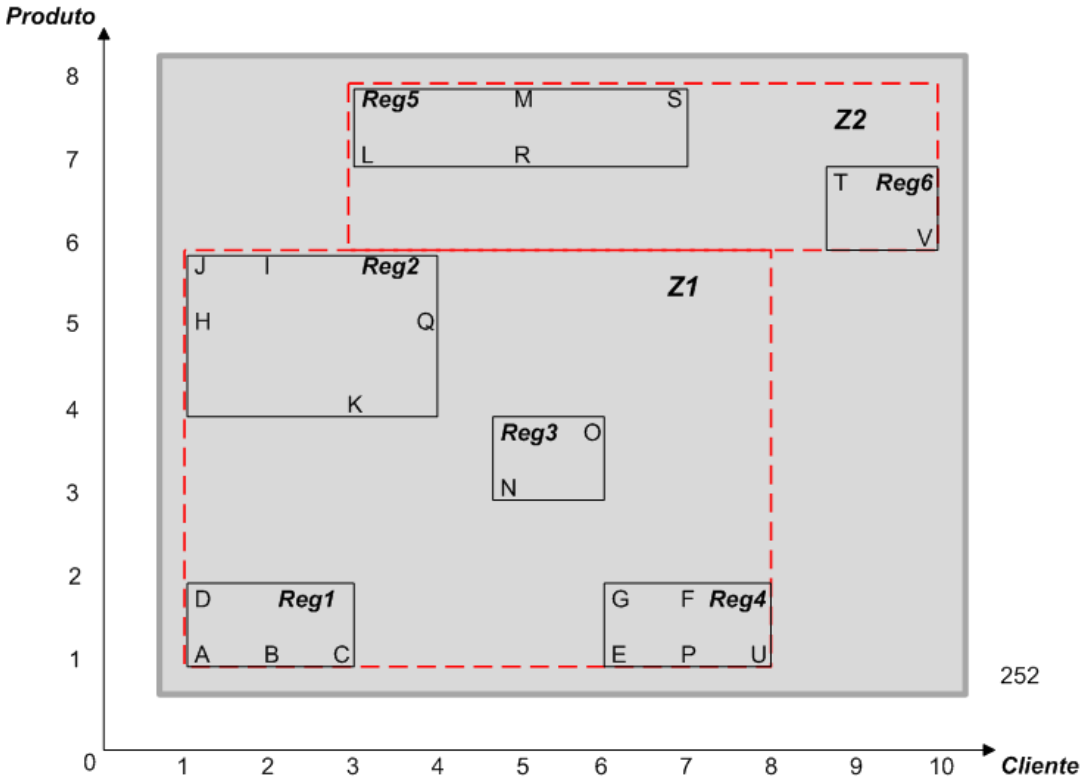

Figura 28 – Nodos acedidos na obtenção do resultado da *query* Ex4

Em seguimento desta análise, podemos atentar no exemplo usado para a operação de *drill down*. Através desta operação, navegamos para um nível de maior detalhe, passando para um nível inferior de agregações. Logo, devido a uma redução do volume das regiões de procura de dados, é intuitivo que haja

uma aproximação no desempenho entre a Ra*-Tree e a sua predecessora, pois a probabilidade de conter *MBR* baixa consideravelmente. Se olharmos para a Figura 29, conseguimos visualizar as regiões de procura de dados para a *query* Ex5 através das regiões rectangulares assinaladas a azul. Tendo em conta a figura anterior, podemos, de facto, concluir que as regiões de procura são bem mais reduzidas, assim como o seu impacto nos *MBRs* da estrutura. Contrariamente à situação anterior, em que a região de procura continha toda a estrutura, reduzindo drásticamente os acessos ao disco, para este caso tal não acontece, havendo, por isso, necessidade de visitar um número bem maior de nodos. Pelo facto de não haver uma única região de procura que contenha um *MBR*, o número de acessos ao disco seria semelhante ao ocorrido para a estrutura predecessora.

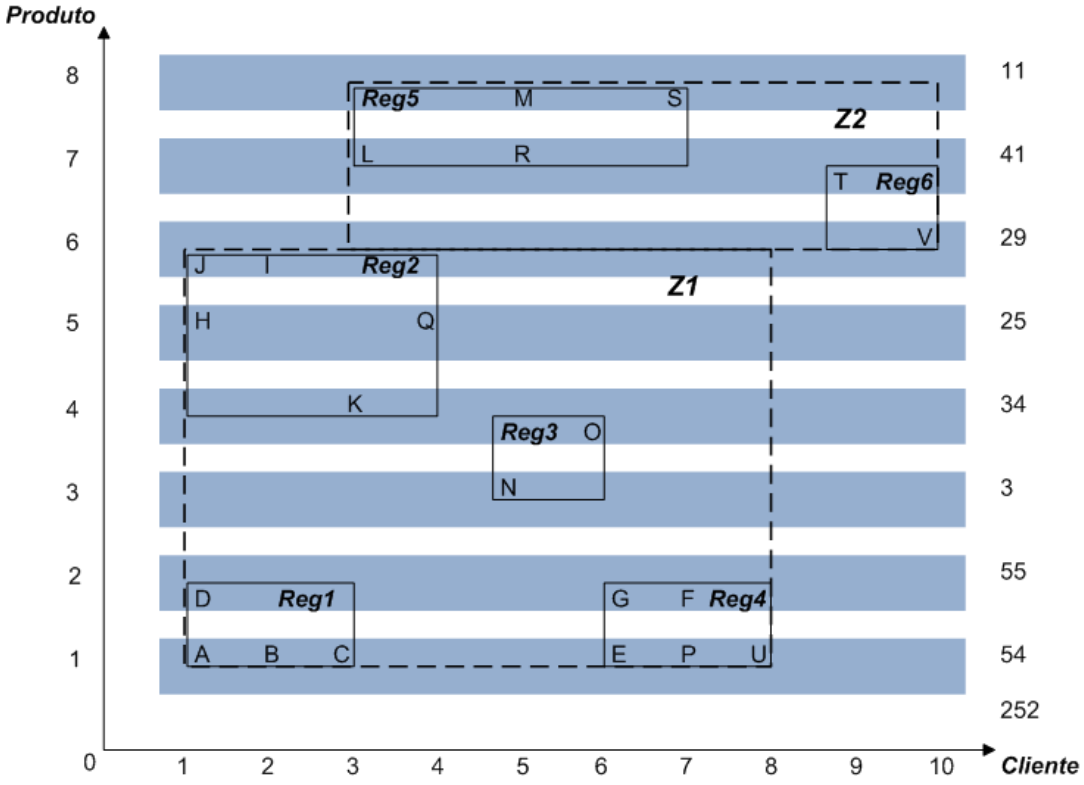

Figura 29 – Regiões de procura para a *query* Ex5

Embora os exemplos aqui demonstrados sejam relativos a uma pequena amostra de dados, o que de facto convém salientar é a potencialidade gerada pela materialização de dados agregados nos nodos-superiores. Esta alteração permite uma melhor adaptação ao tipo de operações e interrogações, normalmente presentes, nos sistemas de processamento analítico, visando, essencialmente, uma redução de acessos ao disco, menores incursões na estrutura e, consequentemente, menores tempos de resposta.

Independentemente do volume de dados usado, apenas para cenários cujas regiões de agregação sejam bastante reduzidas a performance da Ra*-Tree não se evidencia da sua predecessora. Neste sentido, tal como demonstrado em [HSL01, JL98, PKZT01, TPZ02], em operações de agregação, a inclusão de técnicas que materializam e alocam dados agregados na organização dos índices permite, comparativamente ás estruturas iniciais, obter diferenças de desempenho significativas.

Na secção seguinte serão demonstrados mais alguns factores que influenciam a organização da Ra*-Tree, e qual o impacto que geram no desempenho da mesma.

3.5 Dimensionalidade, Cardinalidade e Distribuição dos Dados

A avaliação de índices multidimensionais de dados é efectuada perante a análise do desempenho obtido na execução das operações base e mediante a conjugação e combinação de um conjunto de variáveis que, tendencialmente, têm uma forte influência na organização e funcionamento das estruturas, assim como se apresentam como factores vitais a ter em conta pelos sistemas alvo. Dentro deste prisma, e atendendo às características dos sistemas OLAP, para além do volume de dados e tamanho das regiões de procura, analisados na secção

anterior, a adequação da Ra*-Tree às mais diversas variações na dimensionalidade, cardinalidade e distribuição dos dados, necessita de uma reflexão e análise para melhor avaliar as potencialidades, assim como as circunstâncias mais favoráveis para o seu uso.

A dimensionalidade é um dos parâmetros mais importantes para o estudo de estruturas multidimensionais. O aumento do número de dimensões e atributos dimensionais aumenta, exponencialmente, a robustez e complexidade destas estruturas, em muito devido à necessidade de se manter toda uma organização espacial dos dados que favoreça a execução das operações base. Desta forma, o processamento destas operações torna-se computacionalmente mais exigente. Tendo em conta a sua orgânica, em muito similar à sua predecessora, o desempenho em ambientes de média/alta dimensionalidade tende a diminuir, derivado á existência de sobreposição de *MBRs*. Embora os processos de procura de dados agregados sejam mais acelerados, o facto de haver a necessidade de ter se efectuar várias incursões na estrutura degrada o desempenho à medida que o número de dimensões aumenta. Dentro deste contexto, a inclusão destas técnicas de agregação em estruturas mais adaptadas a ambientes de alta dimensionalidade, como a X-Tree [BKK01], podem resultar em melhores performances. Sendo assim, conclui-se que a Ra*-Tree é, especialmente, vocacionada para ambientes de média/baixa dimensionalidade, obtendo maiores ganhos com regiões de procura de maior volume.

A distribuição dos dados é, também, um importante factor de análise, visto ter uma implicância directa na organização espacial das estruturas. Em relação à Ra*-Tree, em [J02] é demonstrado que os ganhos obtidos face à sua predecessora, relativamente aos processos de procura, não estão dependentes de um qualquer tipo de distribuição em particular. Para distruibuições normais, uniformes e orientadas a um qualquer eixo do espaço multidimensional, os ganhos obtidos são, relativamente, semelhantes, ou seja, conforme se aumenta a

região de procura, maior são os ganhos obtidos, independentemente da localização dos dados.

Por último, a cardinalidade dos atributos dimensionais é mais um factor de relevo, pois está directamente ligado à construção da estrutura. Para além disto, juntamente com a dimensionalidade, influencia a densidade dos dados, isto é, se aumentarmos a dimensionalidade e a cardinalidade, a probabilidade de haver uma maior dispersão dos dados aumenta. Embora este cenário não afecte significativamente os ganhos da Ra*-Tree face à sua predecessora, pois estes estão, essencialmente, relacionados com o tamanho da região de procura e volume de dados, a comparação desta estrutura face ao comportamento dos bitmaps, presente em [J02], demonstra que para ambientes de baixa dimensionalidade, independentemente da cardinalidade dos dados, a Ra*-Tree possuia melhores desempenhos. No entanto, esse tipo de informação está sempre dependente do funcionamento dos mecanismos físicos actuais(hardware), visto serem índices com especificações e orgânicas muito distintas.

Capítulo 4

Conclusões e Trabalho Futuro

O universo R-Tree é composto por um variado leque de estruturas em diversas áreas de intervenção. Com esta tese, pretendeu-se demonstrar a importância deste mundo para as estruturas multidimensionais de dados. A R-Tree é catalogada como uma das mais importantes estruturas dentro desta categoria. A base deste sucesso acentou, essencialmente, em 2 factores: a similaridade com a B-Tree e a sua adequação à distribuição dos dados. A sua orgânica, baseada numa disposição hierárquica, totalmente balanceada, de regiões multidimensionais criadas a partir das coordenadas dos seus dados, confere-lhe um maior aproveitamento e uma acção de rastreio do espaço multidimensional abrangido. Para além disto, é garantido um controlo e uma estabilidade em toda a sua estrutura, ou seja, independentemente da existência de uma qualquer tendência distributiva nos seus dados, todos os nodos se encontram no mesmo nível. Desta forma, há zelo por uma estrutura com baixa altura, o que facilita a execução das operações base. Por outro lado, a R-Tree providencia um bom aproveitamento da memória, visto que cada um dos seus nodos é associado a uma página do disco. Sendo assim, este índice pode ser configurado para efectuar o melhor aproveitamento possível destes componentes.

Através da informação da capacidade máxima de cada página do disco pode-se definir o número máximo de registos em cada nodo. Este factor permite, não só optimizar a utilização da memória, como, também, diminuir o número de acessos à mesma, o que implica, directamente, numa redução nos tempos de acesso aos dados. No entanto, esta estrutura possui uma característica que se reflecte negativamente no seu desempenho. O facto de permitir sobreposição nas suas regiões multidimensionais leva a uma degradação no tempo de acesso aos dados. Tendo em conta que o processo de procura é baseado em relações topológicas entre a informação pretendida e as regiões multidimensionais que constituem a R-Tree, é natural haver correspondência com mais que uma região multidimensional em cada nível da árvore, o que aumenta o número de travessias necessárias para se obterem os dados em causa. Este factor torna-se bastante evidente para processos de procura em ambientes de média/alta dimensionalidade com elevado volume de informação, onde a performance da R-Tree sofre uma degradação acentuada.

As potencialidades e defeitos evidentes na R-Tree funcionaram como elemento impulsionador para a comunidade científica. Se relacionarmos, apenas, todos os seus conceitos positivos, torna-se evidente que esta estrutura foi deveras inovadora e , rapidamente, se tornou numa referência, um *"ícone"*, para este género de estruturas. A necessidade de suprimir os seus pontos negativos e extender as capacidades foi o desafio, encontrado adoptado pelos investigadores, que motivou o aparecimento de inúmeras variantes.

Dentro das variantes analisadas, a R^+-Tree mostrou ser, especialmente, adequada para *point queries*. Esta estrutura aboliu a sobreposição de regiões multidimensionais presentes no mesmo nível hierárquico e aplicou uma política de replicação dos dados que sobrepusessem 2 ou mais regiões multidimensionais do nível superior. A ausência de sobreposição de regiões multidimensionais nos níveis intermédios reduz drásticamente o número de travessias da estrutura. Além do mais, a presença de duplicados implica que apenas uma travessia seja

efectuada até ao dado pretendido. No entanto, esta estrutura apresenta limitações evidentes. Por um lado, a existência de duplicados obriga a um controlo mais rigoroso nas operações sobre *range queries*, visto que a existência de dados repetidos degrada a qualidade da informação. Esta situação motiva um tempo e custo computacional suplementar. Por outro lado, esta estrutura possui regras rígidas sobre o particionamento espacial relativo à indexação das regiões multidimensionais presentes nos níveis superiores. Na presença de uma operação de *splitting* de uma destas estruturas, pode haver uma propagação desta operação, não só para os níveis superiores, mas também para níveis inferiores. Desta forma, a gestão e manutenção desta estrutura torna-se mais árdua e dispendiosa em termos computacionais e temporais. A R*-Tree provou ser uma estrutura mais robusta, no entanto, mais promissora que as anteriormente referidas. A este facto não é alheio um melhor manuaseamento do espaço multidimensional. Esta estrutura introduziu novas heurísticas de particionamento espacial e novos comportamentos para uma mais eficaz adequação da sua orgânica à localização dos dados. Reinserção dos dados e critérios de divisão espacial baseados na redução de área e perímetro das regiões multidimensionais foram o mote para esta evolução. Tal como a R-Tree, a R*-Tree permite sobreposição de regiões multidimensionais, embora a um nível mais reduzido.

A Hilbert R-Tree é, também, vista como uma estrutura de destaque. A introdução da *hilbert curve*, uma *space filling curve*, não só era vista como uma tradução da localização espacial dos dados para ambientes unidimensionais, como funcionava como uma excelente técnica para a ordenação dos dados dentro da estrutura. Para além deste factor, a Hilbert R-Tree usa um conceito diferente de *Splitting*. Esta operação deixou de ser efectuada sempre que um nodo entrava em situação de *overflow*. A política adoptada consistia em distribuir algumas das entradas deste nodo pelos seus vizinhos, dependo da configuração adoptada. Se os nodos vizinhos entrassem em situação de *overflow*, então, nesse

caso, procedia-se á criação de um novo nodo na estrutura. Esta estratégia permitia poupar tempo e custos computacionais, principalmente, para as operações de inserção. No geral, o desempenho obtido por esta estrutura foi significativo, tendo obtido resultados mais satisfatórios que as estruturas predecessoras.

Estas estruturas, apesar dos progressos efectuados, pecavam na abordagem aos ambientes de média e alta dimensionalidade. Neste sentido, a X-Tree elevou o universo R-Tree a outros níveis. A possibilidade de aumentar a capacidade dos seus nodos intermédios, para evitar sobreposição de regiões multidimensionais dentro do mesmo nível, e a sua capacidade para se moldar aos ambientes onde se encontra inserida, adoptando um carácter hierárquico, para média/baixa dimensionalidade, e linear, para média/alta dimensionalidade, permitiu-lhe obter resultados bem mais animadores que as restantes estruturas. Até hoje, a X-Tree é vista como uma estrutura de relevo para ambientes de média/alta dimensionalidade.

A influência destas estruturas é visível em várias áreas, nomeadamente aplicações geográficas, multimédia, científicas, médicas, suporte à decisão e bases de dados espaciais.

Os sistemas de processamento analítico foram um dos grandes focos deste trabalho. Estes sistemas estão intimamente ligados a *sistemas de Data Warehousing*, sendo responsáveis por efectuar uma panóplia de cálculos e agregações dos dados em ambientes multidimensionais. Neste sentido, a Ra*-tree surgiu como uma estrutura capaz de corresponder a algumas necessidades destas aplicações. Um dos tipos de *queries* mais usuais neste sistemas baseiam-se na procura de informação agregada em 2 ou mais dimensões. A necessidade de obter informação agregada correspondente a um vasto número de dados no menor espaço temporal possível tornou-se um desafio para a comunidade científica, e em particular, para as estruturas de indexação multidimensional. A Ra*-Tree, através da incorporação de dados agregados nos seus nodos

intermédios permitiu reduzir drásticamente o número de travessias da estrutura, assim como a profundidade das mesmas. Foi, também, demonstrado que quanto maior for o nível de agregação mais satisfatório é o seu desempenho.

Como trabalho futuro, seria interessante efectuar uma implementação destas estruturas para uma melhor visão das suas vantagens e desvantagens. No entanto, como referido em [GG98], existem factores que podem provocar impactos inesperados, tais como a linguagem de programação usada, o tamanho e o uso das páginas do disco, os *data sets* e a própria codificação. Embora a literatura nos permita obter dados bastante esclarecedores sobre o desempenho destas estruturas, não seria de desprezar usar os processos de agregação adoptados pela Ra*-Tree noutras estruturas e efectuar estudos comparativos entre as mesmas.

Bibliografia

[AAD+96] Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the computation of multidimensional aggregates. In VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases, pages 506-521, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[ABHY08] Lars Arge, Mark De Berg, Herman Haverkort, and Ke Yi. The priority r-tree: A practically e_cient and worst-case optimal r-tree. ACM Trans. Algorithms, 4(1):1-30, 2008.

[AHVV99] Lars Arge, Klaus Hinrichs, Jan Vahrenhold, and Jeffrey Scott Vitter. Efficient bulk operations on dynamic r-trees. In ALENEX '99: Selected papers from the International Workshop on Algorithm Engineering and Experimentation, pages 328-348, London, UK, 1999. Springer-Verlag.

[BBKM00] Christian Bohm, Stefan Berchtold, Hans-Peter Kriegel, and Urs Michel. Multidimensional index structures in relational databases. Journal of Intelligent Information Systems, 15(1):51-70, 2000.

[BDJ+06] Doug Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. fficient allocation algorithms for olap over imprecise data. In VLDB '06: Proceedings of the 32nd international conference on Very large data bases, pages 391-402. VLDB Endowment, 2006.

[BKK01] S. Berchtold, D.A. Keim, and H.P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. Readings in Multimedia Computing and Networking, 2001.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pages 322-331, 1990.

[BPT02] Sotiris Brakatsoulas, Dieter Pfoser, and Yannis Theodoridis. Revisiting rtree construction principles. In ADBIS '02: Proceedings of the 6th East European Conference on Advances in Databases and Information Systems, pages 149-162, London, UK, 2002. Springer-Verlag.

[CCR02] Li Chen, Rupesh Choubey, and Elke A. Rundensteiner. Merging r-trees: Efficient strategies for local bulk insertion. Geoinformatica, 6(1):7-34, 2002.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: analysis of r-trees using the concept of fractal dimension. In PODS '94: Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 4-13, New York, NY, USA, 1994. ACM.

[FR89] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 247-252, New York, NY, USA, 1989. ACM.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, 1(1):29-53, 1997.

[GG98] V. Gaede and O. Gunther. Multidimensional Access Methods. ACM Computing Surveys, 30(2), 1998.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J.D. Ullman. Index Selection for OLAP. Proceedings of the Thirteenth International Conference on Data Engineering, pages 208-219, 1997.

[GLL98] Yván J. García, Mario A. Lopez, and Scott T. Leutenegger. On optimal node splitting for r-trees. In VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases, pages 334-344, San Francisco CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[GMUW01] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. Database Systems: The Complete Book. Prentice Hall, October 2001.

[Gup97] H. Gupta. Selection of Views to Materialize in a Data Warehouse. Database Theory-ICDT'97: 6th International Conference, Delphi, Greece, January 8-10, 1997: Proceedings, 1997.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. ACM Press New York, NY, USA, 1984.

[HRU96] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. Proceedings of the 1996 ACM SIGMOD international conference on Management of data, pages 205-216, 1996.

[HSL01] Seokjin Hong, Byoungho Song, and Sukho Lee. Efficient execution of range aggregate queries in data warehouse environments. In ER '01: Proceedings of the 20th International Conference on Conceptual Modeling, pages 299-310, London, UK, 2001. Springer-Verlag.

[J02] Marcus Jurgens. Index structures for data warehouses. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[JL98] M. Jurgens and H.J. Lenz. The R-tree: An Improved R*-tree with Materialized Data for Supporting Range Queries on OLAP-Data. Proceedings of the 9th International Workshop on Database and Expert Systems Applications, 1998.

[jLS97] Hans j. Lenz and Arie Shoshani. Summarizability in olap and statistical data bases. In In Proc. of SSDBM, pages 132-143. IEEE Computer Society, 1997.

[JLS99] H. V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava. Snakes and sandwiches: optimal clustering strategies for a data warehouse. SIGMOD Rec., 28(2):37-48, 1999.

[Jur] Marcus Jurgens. Modeling and improving the performance of multidimensional indexstructures for range queries on olap data.

[KDFB99] Prof A. Kemper, Ph. D, Peter M. Fischer, and Moderation Silvia Breu. Persistent data, 1999.

[KF93] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. In CIKM '93: Proceedings of the second international conference on Information and knowledge management, pages 490-499, New York, NY, USA, 1993. ACM.

[KF94] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved rtree using fractals. In VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, pages 500-509, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[KR98] Yannis Kotidis and Nick Roussopoulos. An alternative storage organization for rolap aggregate views based on cubetrees. In SIGMOD Conference, pages 249-258, 1998.

[KRSB99] K. V. Ravi Kanth, Siva Ravada, Jayant Sharma, and Jay Banerjee. Indexing medium-dimensionality data in oracle. In SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data, pages 521-522, New York, NY, USA, 1999. ACM.

[KRT+98] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, and W. Thornwaite. The Data Warehouse Lifecycle Toolkit: Expert Methods

for Designing, Developing and Deploying Data Warehouses with CD Rom. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[LHJ+03] Mong Li Lee, Wynne Hsu, Christian S. Jensen, Bin Cui, and Keng Lik Teo. Supporting frequent updates in r-trees: a bottom-up approach. In VLDB '2003: Proceedings of the 29th international conference on Very large data bases, pages 608-619. VLDB Endowment, 2003.

[LJF94] King Ip Lin, H. V. Jagadish, and Christos Faloutsos. The tv-tree: an index structure for high-dimensional data. The VLDB Journal, 3(4):517-542, 1994.

[MNPT03] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. R-tree have grown everywhere. ACM Computing Surveys, 5:1-07, 2003.

[MNPT05] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Y. Theodoridis. R-Trees: Theory and Applications (Advanced Information and Knowledge Processing). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[MRB99] V. Markl, F. Ramsak, and R. Bayer. Improving OLAP performance by multidimensional hierarchicalclustering. Database Engineering and Applications, 1999. IDEAS'99. International Symposium Proceedings, pages 165-177, 1999.

[Neu01] L. Neumann. The R-Tree. Algorithms and Data Structures for Persistent Data, 2001.

[PKZT01] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. E_cient OLAP Operations in Spatial Data Warehouses. Proc. of SSTD, 280, 2001.

[PSTE95] Dimitris Papadias, Timos Sellis, Yannis Theodoridis, and Max J. Egenhofer. Topological relations in the world of minimum bounding rectangles: a study with r-trees. SIGMOD Rec., 24(2):92-103, 1995.

[RKR97] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cubetree: organization of and bulk incremental updates on the data cube. Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pages 89-99, 1997.

[RL85] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed r-trees. In SIGMOD '85: Proceedings of the 1985 ACM SIGMOD international conference on Management of data, pages 17-31, New York, NY, USA, 1985. ACM.

[Sar97] Sunita Sarawagi. Indexing OLAP data. Data Engineering Bulletin, 20(1):36-43, 1997.

[SDNR96] Amit Shukla, Prasad Deshpande, Jeffrey F. Naughton, and Karthikeyan Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases, pages 522-531, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[SRF87] T.K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1987.

[tHAMS97] Ching tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in olap data cubes. In In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, pages 73-88, 1997.

[TPZ02] Yufei Tao, Dimitris Papadias, and Jun Zhang. Aggregate processing of planar points. In EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology, pages 682-700, London, UK, 2002. Springer-Verlag.

[TS94] Yannis Theodoridis and Timos K. Sellis. Optimization issues in r-tree construction (extended abstract). In IGIS '94: Proceedings of the International Workshop on Advanced Information Systems, pages 270-273, London, UK, 1994. Springer-Verlag.

[TS96] Yannis Theodoridis and Timos Sellis. A model for the prediction of r-tree performance. In PODS '96: Proceedings of the _fteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 161-171, New York, NY, USA, 1996. ACM.

Referências WWW

[1] http://www.google.com

Motor de busca de informação. Essencial para qualquer tipo de pesquisa.

[2] http://www.wikipedia.org

Enciclopédia online que contém informação sobre os temas abordados.

[3] http://citeseer.ist.psu.edu/

Repositório online de artigos científicos.

[4] http://scholar.google.com/

Motor de pesquisa de artigos científicos e materiais de estudo criado pela Google.

[5] http://www.olapcouncil.org

Site com informação actualizada e relevante sobre o universo dos sistemas de processamento analítico.

[6] http://portal.acm.org/

Repositório online de artigos científicos.

[7] http://www.rtreeportal.org

Site com informação detalhada e dedicada ao universo R-Tree.