Universidade do Minho
Escola de Engenharia

Jorge Alberto Faria Miranda

Manutenção de índices multidimensionais recorrendo a Quadtrees como estruturas de indexação

Dezembro de 2008

Universidade do Minho
Escola de Engenharia

Jorge Alberto Faria Miranda

Manutenção de índices multidimensionais recorrendo a Quadtrees como estruturas de indexação

Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor Orlando Belo

Dezembro de 2008

Agradecimentos

Esta tese representa, para mim, o culminar do meu percurso académico. Contudo, este não teria sido uma experiência tão extraordinária se não fosse pela presença, companhia e suporte de um conjunto de pessoas às quais devo um agradecimento pela sua influência no meu crescimento académico e pessoal.

Ao Professor Doutor Orlando Belo, sem o qual este projecto não seria possível, pelo apoio e análise crítica que manifestou ao longo de todos destes anos de ensino e orientação.

Ao pessoal de Barcelos, Esposende e Braga, vocês sabem quem são =). Destes, não posso deixar de nomear alguns, que tiveram uma participação especialmente activa no decorrer deste projecto: Abreu, Azz, Cris, Mauro, Mariana, Pi e Schroder.

Ao meu pai, que não teve uma tarefa fácil, mas que soube lutar e vencer todas as vicissitudes da vida e que se tornou um exemplo de trabalho, de sacrifício e de amor. Melhor era impossível, pai. À Isabel, que soube ser mãe quando nunca teve tal obrigação e que desde cedo foi um pilar no meu desenvolvimento. As palavras nunca serão suficientes para vos agradecer tudo aquilo que fizeram e continuam a fazer por mim.

Aos outros membros da minha família mais próxima, irmãs Raquel e Paula e, de forma muito especial, à minha mãe a quem a vida não permitiu que me acompanhasse nesta aventura.

À Carlène, pelos dezassete motivos que nós sabemos, por todas as experiências que vivemos e ainda vamos viver, e por se mostrar mais do que a concretização dos meus melhores sonhos.

Resumo

Cada vez mais as aplicações de Sistemas de Suporte à Decisão utilizam Sistemas de *Data Warehousing* para compreender e analisar o seu modelo de negócios. Estas aplicações requerem, frequentemente, um tempo de resposta diminuto para uma grande variedade de perguntas numa vasta quantidade de dados. Os sistemas OLAP foram criados com o objectivo de ir ao encontro da necessidades destas aplicações. No entanto, a sua performance tende a diminuir com o aumento do número de dimensões, bem como o aumento de volume de dados. Uma das soluções para minorar esta perda de desempenho consiste na utilização de estruturas de indexação.

Actualmente, as Quadtree representam um conjunto de estruturas utilizadas na manipulação de dados em ambientes multidimensionais, bem como em muitas outras áreas que vão desde a criação de jogos tridimensionais a Sistemas de Informação Geográfica. Optou-se por esta estrutura devido à sua natural aptidão no manuseamento não só de pontos, como de regiões de dados. .

A presente dissertação irá debruçar-se no estudo e análise dos algoritmos desenvolvidos pela comunidade científica utilizados em sistemas multidimensionais, bem como alguns existentes em sistemas comerciais, incidindo principalmente nas Point Region Quadtree. Discute-se o seu funcionamento, bem como as suas implicações no que diz respeito ao *tuning* de bases de dados multidimensionais e apresenta-se ainda alterações efectuadas a estas árvores visando sempre o aumento dos índices de desempenho nas operações relevantes em motores OLAP.

Palavras chave: Sistemas de Suporte à Decisão, *Data Warehouse*, modelação dimensional, OLAP, índices multidimensionais, Quadtree.

Abstract

Decision Support Systems are increasingly depending on data warehousing to scrutinize their business model. These applications often require little response time to a wide diversity of queries in a enormous amount of data. OLAP databases have been created in order to meet the needs of these applications. However, their performance tends to decrease with the increasing number of dimensions and data volume. One of alternatives to relieve this performance loss is to apply indexing methods designed for spatial data.

A Quadtree is a data structure used in data manipulation in multidimensional environments as well as in many other areas ranging from videogames creation to Geographic Information Systems. We chose this structure because its natural ability in handling points and regions of data.

In this thesis we study and analyze spatial data algorithms developed by the scientific community as well as some existing commercial systems, focusing on Point Region Quadtree. We discuss their operations and its implications in multidimensional databases and propose an adapted Quadtree aimed at increasing the rates of performance in OLAP environments.

keywords: Decision Support Systems, Data Warehouse, dimensional modelling, OLAP, multidimensional indices, Quadtree.

Conteúdo

Lista de Figuras

1 Introdução

1.1 Contextualização

Ao longo das últimas décadas, o emergir de novas tecnologias e a consequente melhoria na capacidade de processamento pela parte dos computadores e outros equipamentos conduziu a um aumento na geração e armazenamento de dados. A difusão da utilização de códigos na vendas de produtos ou serviços, os registos de transacções entre indivíduos e instituições ou a recolha de dados através de satélites ou sensores são exemplos da evolução observada nas tecnologias da informação e dos quais derivam uma grande quantidade de dados.

Assim, ocorreu uma proliferação exponencial da utilização de bases de dados pela parte de diferentes entidades. Acompanhando toda esta progressão, os Sistemas de Gestão de Bases de Dados (SGBD) evoluíram no sentido de tentar suprir as necessidades impostas pelo mercado, munindo-se de novas funcionalidades, melhor desempenho e uma mais eficaz gestão do espaço. No entanto, à medida que o volume de dados gerados aumenta, estes sistemas mostram-se cada vez mais ineficazes em responder às perguntas que lhes são colocadas e, consequentemente, surge a necessidade de utilizar novas ferramentas que sejam capazes, de forma automática e inteligente, de processar os dados e transformá-los em informação útil.

Frequentemente, quando se opera com sistemas complexos, o conhecimento necessário para a tomada de decisões supera a nossa capacidade cognitiva. Mesmo com um estudo aprofundado que conduza ao entendimento conceptual das variáveis existentes, torna-se sempre difícil prognosticar a forma

como o sistema irá reagir a manipulações externas que possam eventualmente resultar das decisões tomadas. Existe uma quantidade substancial de provas empíricas e estudos que revelam que a capacidade intuitiva de tomar decisões do ser humano é longe da óptima e vai-se deteriorando à medida que aumenta a complexidade e o *stress*. Como muitas vezes as consequências das escolhas feitas são determinantes, estudar formas para auxiliar a melhorá-las sempre foi um grande foco da comunidade científica. Áreas de investigaçao como a estatística e a economia, entre outras, foram, ao longo do tempo, desenvolvendo métodos para uma tomada de decisões racional. Mais recentemente, estes têm sido implementados em *software*, sendo utilizados em ambientes direccionados para o processamento de tomadas de decisão. A estes ambiente dá-se o nome de Sistemas de Suporte à Decisão (SSD) e, conceptualmente, podem ser definidos como sistemas computacionais interactivos que auxiliam o utilizador nas actividades de julgamento e escolha.

Os sistemas de suporte à decisão fazem parte de um conjunto de tecnologias que permitem o cruzamento de informações e suportam a análise dos indicadores de desempenho de um negócio, que se denomina de *Business Intelligence*. Estes envolvem o processo de recolha, organização e disponibilização das múltiplas fontes de dados que as organizações gerem no seu quotidiano. Inclui, ainda, todos os processos de análise e manipulação de dados com vista a promover a sua partilha global e suportar a monitorização em tempo real do desempenho da empresa. O objectivo final de *Business Intelligence* é melhorar o suporte aos processos de tomada de decisão nos diferentes níveis de gestão das organizações.

Estes sistemas têm ganho proeminência e popularidade em diversos sectores como o dos negócios, o da engenharia, o militar e o da saúde. São especialmente úteis em situações em que a quantidade de informação disponível torna inadequada a tomada de decisões por parte do agente de decisão sem algum tipo de apoio. Fornecem auxílio através da integração de diversas fontes de informação, providenciando acesso a informação relevante e permitindo ainda a sua estruturação. Aplicações de suporte à decisão aumentam em produtividade, eficiência e eficácia, representando numerosas vantagens para as instituições e permitindo-lhes efectuar melhores decisões na gestão do seu modelo de negócios. São responsáveis pela obtenção e tratamento de todos os dados operacionais de uma instituição, registando toda a informação relevante num repositório central, integrado e não volátil, denominado de *Data Warehouse*, tendo surgido o conceito em [Devlin and Murphy, 1988].

Devido à sua não volatibilidade, o que significa que os seus registos nunca serão apagados, pode-se dizer que um *Data Warehouse* é de uma base de dados que cresce periodicamente e que comporta uma grande quantidade de dados. Desta forma, toda a informação é protegida e um registo histórico é mantido podendo-se assim, posteriormente e sempre que for necessário, proceder a uma análise relativa a qualquer intervalo de tempo. A sua versatilidade estende-se a diversos sectores de mercado, podendo os seus dados ser tão diversificados como as vendas de uma instituição e os seus produtos, informação sobre os recursos humanos, dados sobre leituras sensoriais, entre outros.

Para além de funcionar como arquivo para uma grande quantidade de dados, os meios para apresentar e analisar a informação contida, para extrair, transformar e integrar registos são também componentes essenciais a um sistema de *Data Warehousing*. Em suma, permite o aparecimento de factos e tendências que possibilitam uma melhor tomada de decisões através dos dados existentes, e a sua utilização proporciona numerosas vantagens como, por exemplo:

- As inconsistências verificadas nos dados são identificadas e resolvidas anteriormente à sua inserção num *Data Warehouse* o que facilita a análise e o suporte às decisões.
- Facilita a observação das tendências e relações de negócio.
- Aumenta a consistência dos dados.
- Como são mantidos em separado dos restantes sistemas, facilitam o processo de análise sem impacto para os sistemas operacionais.
- Flexibilidade para atender diferentes análises.
- Devido ao facto de a maior quantidade de *Data Warehouses* serem integrados, é possível aceder aos dados de diferentes áreas de negócio que pertençam a uma instituição, e não apenas ao de um departamento.

Como se pode concluir, a utilização de um *Data Warehouse* traduz-se num claro aumento de desempenho e, consequentemente, uma produtividade mais elevada. No entanto, não será correcto assumir-se que se trata de uma ferramenta adequada para todos os ambientes institucionais, uma vez que apresenta também um conjunto de desvantagens que é importante ter em conta. De entre elas, passa-se a enumerar as que mais se destacam:

- Elevado consumo de tempo na sua criação e manutenção.
- Possíveis incompatibilidades entre os sistemas e os dados.
- Custos elevados de manutenção.

De modo a facilitar análises complexas e respectivas visualizações das respostas obtidas, os *Data Warehouses* são modelados multidimensionalmente. Numa instituição cujo propósito é a venda de produtos, dimensões de interesse seriam, por exemplo, o produto, dados do cliente, a altura da venda, entre outras. Normalmente define-se cada dimensão, hierarquicamente, o que significa que a dimensão tempo poderia ser estruturada como ano→trimestre→ mês, por exemplo ou, a dimensão produto como área→categoria→ identificador de produto. É a intersecção das hierarquias que permite a obtenção de resultados, pois cruzamento da informação permite aceder a um conjunto de células que guardam valores sobre a respectiva transacção.

Para tornar este conceito perceptível tome-se o seguinte exemplo. Como acima referido, encontram-se os dados sobre uma determinada compra de um produto. O valor contido no interior de cada célula representa o valor final da transacção. Observando as três dimensões existentes, facilmente se compreende que um cliente hipotético A comprou um produto B na data C e o valor dessa operação foi de 55 unidades monetárias. Devido às ligações das hierarquias é possível, para um agente de decisão de uma instituição perguntar *“Quantas unidades de arroz foram vendidas no primeiro semestre de 2008“* ou *”Qual foi a diferença para a mesma época no ano transacto?”*, etc.

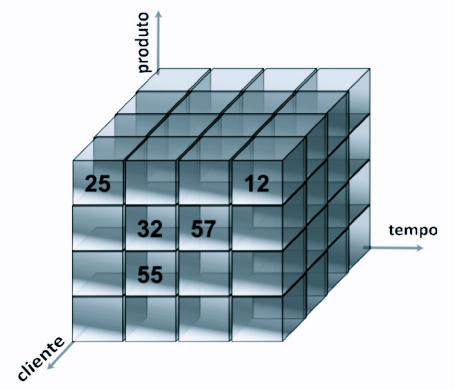

Figura 1.1: *DataCube*

Toda esta estrutura organizacional dos dados permite um alargado leque de opções e e a colocação das mais diversas questões. No entanto, estes devem-se disponibilizar de forma intuitiva e perceptível, de modo a fornecer aos agentes de decisão uma noção clara da actividade relevante da instituição. Esta apresentação dos dados torna-se possível com a utilização de sistemas *Online Analytical Processing* (OLAP) [Codd E.F. and C.T., 1993]. Estes, empregam estruturas multidimensionais de armazenamento e representação da informação denominados de *DataCubes* (figura 1.1). Conceptualmente, estas estruturas podem ser definidas como cubos num espaço euclidiano *k*-dimensional, onde a localização de cada dado corresponde à intersecção das dimensões que o definem.

Os sistemas OLAP representam uma abordagem cujo objectivo é providenciar rápida e intuitivamente respostas a *queries* complexas, *ad-hoc* e de natureza multidimensional. Permitem a analistas, gestores e executivos obterem uma melhor visão dos dados através de um acesso rápido, consistente e interactivo, bem como uma grande variedade de vistas possíveis.

1.2 Motivação e Objectivos

Devido ao tamanho de um *Data Warehouse*, bem como a complexidade das *queries* realizadas, o tempo de resposta poderá ser elevado. Este atraso, em grande parte dos ambientes de Sistemas de Suporte à Decisão é impraticável, limitando seriamente a produtividade. Nesta dissertação pretende-se estudar e apresentar alguns dos métodos existentes, bem como alternativas que permitam reduzir, em determinadas situações, o tempo de execução. Para isso, existem diversas técnicas que visam um aumento da performance, tais como *Tuning* e *query optimizers*, que conduzem a um melhor manuseamento das agregações existentes [Harinarayan et al., 1996] [Papadias et al., 2001b], bem como a utilização recorrente de índices multidimensionais ou a pré-computação das *queries* realizadas mais frequentemente.

As estruturas de indexação multidimensional representam um dos métodos utilizados quando se procura obter um melhor desempenho em ambientes multidimensionais. E, através desta abordagem, pretende-se, desenvolver um algoritmo utilizando as estruturas das *Quadtree* e que permita atingir os seguintes objectivos:

- Um acesso rápido a pontos e regiões de dados.

- Uma eficaz resolução das eventuais colisões que ocorrem em sistemas OLAP.
- Um balanceamento total ou parcial do índice de modo a diminuir o número de acesso aos nodos.
- Um atenuamento dos efeitos colaterais, como o espaço ocupado pelo índice.
- Umas agregações mais eficientes de regiões adjacentes e não adjacentes.

1.3 Estrutura da dissertação

A tese de mestrado apresentada neste documento aborda temáticas relacionadas com a indexação multidimensional, apresentando diversos tipos de índices com especial destaque para as estruturas Quadtree. De seguida, é proposta uma extensao a estas estruturas, expondo-se e discutindo-se os resultados obtidos no que concerne à sua implementação, e as principais conclusões retiradas a partir deste trabalho reservam-se para o capítulo final. Assim, a presente dissertação encontra-se estruturada da seguinte forma:

- No capítulo dois são descritas algumas situações que representam uma perda de desempenho em ambientes OLAP e algumas abordagens que visam o minoramento deste problema. Explica-se ainda os conceitos de materialização de vistas e um pouco das quatro classes de indexação multidimensional.
- As Quadtree são introduzidas no capítulo 3, no qual se refere a sua aplicação actualmente. Estuda-se a estrutura e operações de todas as variantes que foram relevantes para esta dissertação e ambientes aos quais seriam mais adequadas.
- De seguida, no capítulo quatro, apresentam-se as modificações propostas por esta dissertação. São explicadas as alterações estruturais e operacionais e a motivação que levou à transformação do índice.
- No quinto capítulo, explica-se em que estado é que os testes se realizaram, apresentam-se e discutem-se os resultados. São realizadas comparações com a mesma ou com outras variantes das *Quadtree*.

- No último capítulo, é efectuada a conclusão sobre toda esta experiência a sobre a dissertação, definindo-se conteúdo para trabalho futuro.

2

Sistemas Multidimensionais

Uma grande quantidade das base de dados actualmente existentes são desenhadas para ir ao encontro das necessidades quotidianas operacionais de negócio. Ao contrário do que acontece com os sistemas OLAP, estas são vocacionadas para operar sobre um pequeno volume de dados através de *querys* simples e directas não atingindo os seus índices de desempenho comuns quando se trata de *queries* complexas ou operações sobre uma grande quantidade de informação. Estes sistemas são denominados de OLTP Online Transaction Processing) e apresentam, como características:

- Trabalham sobre dados operacionais, sendo recorrentemente fontes de informação para sistemas de suporte à decisão.
- Têm como objectivo o controlo e a execução de tarefas de negócio, tornando-se possível obter pequenas informações sobre o mesmo.
- Possibilitam a realização de inserções e actualizações eficientes e de pouco tamanho executados por *end users*.
- Efectuam *queries* relativamente simples e que operam sobre um reduzido número de registos.
- Encontram-se altamente normalizados, com um elevado número de tabelas.

Ao contrário do que acontece com os sistemas OLTP, os sistemas de suporte à decisão são responsáveis pela obtenção e tratamento de todos os dados de uma instituição, registando toda a informação relevante no seu respectivo *Data Warehouse*. Sendo este a fonte de pesquisa de toda as análises de

negócio, questões como a coerência, qualidade e consistência da informação são essenciais para a sua validade. Dada a sua natureza e objectivos, as características deste tipo de sistemas diferem em grande escala dos OLTP:

- Os dados utilizados são provenientes de outras bases de dados, frequentemente, de sistemas OLTP.
- Orientado ao assunto de negócio.
- O objectivo deste tipo de sistemas é o auxílio na tomada de decisões, planeamento e resolução de problemas.
- A grande quantidade de dados em operações de inserção ou actualização podem demorar horas a serem executados.
- As *queries* são complexas dada a larga disponibilização de opções em que, frequentemente, são utilzadas agregações.
- O desempenho destes sistemas depende de diversos factores como a complexidade da *querie*, a quantidade de dados envolvida, a granularidade do sistema, etc. O desempenho pode ser aumentado com a utilização de índices ou outras técnicas de *tuning*.
- A desnormalização das tabelas, a utilização de índices, a existências de agregações e o funcionamento como histórico fazem com que estes sistemas costumem atingir grandes dimensões.
- As tabelas são desnormalizadas, privilegiando o desempenho em relação ao espaço ocupado.
- A existência de poucas tabelas utilizando os esquemas do tipo Estrela ou Floco de Neve.

2.1 Modelação dimensional

Os sistemas de suporte à decisão são fortemente orientados ao assunto de negócio da instituição. Este assunto varia consoante a a actividade de negócio e, consequentemente, torna-se necessário adaptar o modelo de dados às necessidades de cada instituição. A modelação dimensional é uma técnica vocacionada essencialmente para a implementação de um modelo de dados que permita a sua visualização de forma intuitiva e com altos índices de

performance. Deste modo, o modelo permite a visualização de dados na forma de um cubo, onde cada dimensão do cubo representa o contexto de um determinado facto, e a intersecção entre as dimensões representa as medidas do facto. O modelo dimensional é baseado, essencialmente, em três elementos:

1. Factos.
2. Dimensões.
3. Métricas.

O modelo dimensional é composto por uma ou mais tabelas de factos que possuem uma as métricas relacionadas com o negócio. Cada métrica é obtida através do cruzamento das dimensões, que são tabelas nas quais se encontram discriminados os atributos de negócio.

Um facto é constituído por um conjunto de dados compostos de métricas e dados contextuais. Deve representar uma determinada transacção ou evento do negócio ocorrido num determinado contexto, obtido na intersecção das diversas dimensões. Essencialmente, caracterizam o negócio através de tudo aquilo que possa ser medido. As características mais comuns de um facto são:

- Mudam ao longo do tempo.
- São representados por valores numéricos.

As dimensões referem-se a um ou mais contextos de negócios onde ocorrem os factos, tais como períodos de tempo, produtos, mercados, clientes e fornecedores. São elementos que permitem descrever o contexto de um determinado facto, os pontos de entrada para a tabela de factos e permitem implementar a interface de utilizador para o *Data Warehouse*. A maioria dos factos envolve, pelo menos, quatro dimensões fundamentais:

A localização, que determina o local onde o facto ocorreu.

A altura, que determina a própria dimensão do tempo

As entidades, identificando as participaram no facto.

O objecto do facto.

As métricas são atributos que quantificam um determinado facto, representando a performance de um indicador em relação às dimensões que contextualizam o facto. Podem possuir uma hierarquia, de forma a estruturar o seu grau de detalhe e importância (valor). Associado a este conceito, pode ainda reflectir-se sobre a importância da granularidade das métricas, dada a sua relevância no *Data Warehouse* uma vez que está relacionada com o nível de detalhe dos dados armazenados e, consequentemente, pode afectar o desempenho do sistema, pois quanto mais elementar for a informação, maior será o volume de dados armazenados.

O principal tipo de modelo dimensional é o modelo do tipo estrela (figura 2.1) [Kimball et al., 1998] no qual existe uma tabela dominante no centro, denominada de tabela de factos que está ligada, através de múltiplas ligações, a diversas tabelas auxiliares com o nome de tabelas de dimensão, e cada uma destas ligada apenas à tabela de factos. Estas tabelas possibilitam, posteriormente, consultas segundo o atributo presente em cada uma. A tabela de factos armazena uma grande quantidade de dados, em função do tempo, obtidos a partir da intersecção de todas as dimensões do esquema.

A versão normalizada de um esquema do tipo Estrela é um esquema Floco de Neve onde, para cada nível de agregação, existe a tabela correspondente (figura 2.1). Uma das maiores vantagens deste esquema diz respeito ao facto de ocupar menos espaço do que o modelo do tipo Estrela, não influenciando no entanto o desempenho por este evidenciado. Esta diferença ocorre devido à normalização de tabelas existente no esquema do tipo Floco de Neve. Segundo Kimball em [Kimball et al., 1998], "o modelo Estrela tem uma arquitectura padrão e previsível e, consequentemente, as ferramentas de consulta e interface do utilizador podem ser mais intuitivas, necessitando de menos alterações, e atingir um melhor desempenho", algo que o modelo Floco de Neve (figura 2.2), não fornece. No entanto, representa uma abordagem menos dispendiosa em recursos. O modelo Estrela tem ainda a vantagem de ser simples e intuitivo, e torna-se actualmente comum pretender optimizar o desempenho em detrimento de outras características como o espaço em disco despendido pelo sistema.

As ferramentas OLAP são utilizadas muito frequentemente como *front end* em ambientes de *data warehousing*. Estes modelos servem-lhes de base, integrando os dados com uma interface poderosa para fornecer toda a informação ao utilizador. Permitem uma análise interactiva de dados multidimensionais. Desta forma a tecnologia de bases de dados multidimensionais tem-se tor-

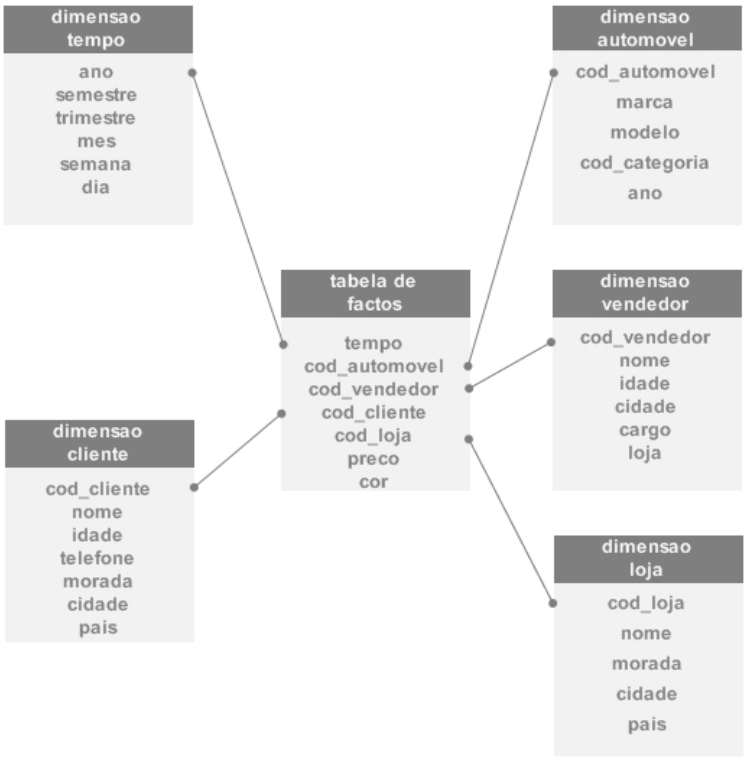

Figura 2.1: Esquema dimensional do tipo Estrela

nado progressivamente mais importante e independentemente das arquitecturas que as implementam em termos de armazenamento, o paradigma multidimensional através do qual os dados são apresentados ao utilizador é central para toda esta temática.

Apesar de serem desenhados de forma a permitir uma enorme variedade de *queries*, *ad hoc* ou pré processadas com altos índices de desempenho, torna-se comum, estes sistemas terem quebras de performance. As causas mais comuns para a existência dessas quebras são:

- Aumento da cardinalidade. Naturalmente, o desempenho em ambientes OlAP será pior quando existe uma alta cardinalidade, aumentado consequentemente os cálculos e as agregações existentes no sistema.
- Aumento das dimensões. Ainda que mantendo a cardinalidade, caso se aumente as dimensões do sistema o número de intersecções necessárias para responder às *queries* executadas aumenta na mesma

proporção, levando a um maior desgaste por parte do sistema para devolver resultados.

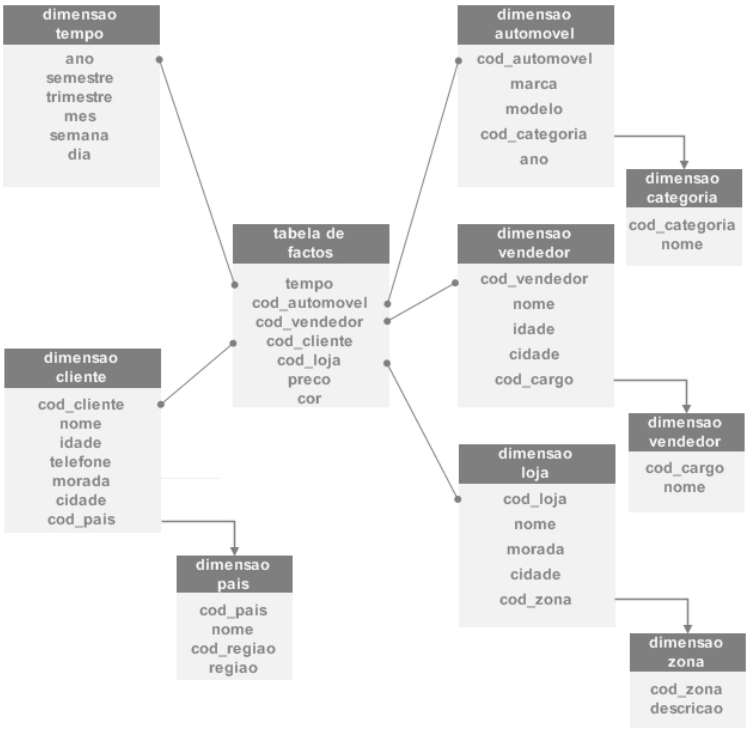

Figura 2.2: Esquema dimensional do tipo Floco de Neve

Ao longo do tempo, diversas abordagens foram realizadas com o objectivo de minimizar tempo e recursos, melhorando os acessos multidimensionais [Gaede and Gunther, 1998]. A utilização de índices multidimensionais, as técnicas de *clustering* ou a materialização de vistas [Gupta, 1997] [Baralis et al., 2005] [Shukla et al., 1998] são duas das técnicas utilizadas para aumentar a performance dos sistemas OLAP. Na secção seguinte são abordadas algumas destas técnicas.

2.2 Técnicas de optimização presentes em ambientes multidimensionais

A análise de dados multidimensionais conduz ao processamento de um grande número de agregações sobre um largo volume de dados. Para ir ao encontro dos requisitos de performance impostos por este tipo de aplicações, é comum realizar-se um pré-processamento parcial das agregações existentes. O desempenho do sistema será proporcional à quantidade de agregações previamente processadas, devido ao aumento de rapidez na obtenção das respostas às *queries* realizadas. Seleccionar o conjunto de vistas a materializar revela-se uma tarefa nada trivial, visto que afecta não só o seu desempenho mas também, indirectamente, o de outras *queries* relacionadas. Diversas abordagens já foram tentadas no que diz respeito à materialização de vistas, incluindo algoritmos *greedy*, procuras aleatórias e utilização de algoritmos genéticos. Em [Harinarayan et al., 1996] e [Gupta, 1997] são propostos modelos que consistem numa materialização parcial do *DataCube*, baseados nos dados mais relevantes para a análise do utilizador, mantendo acessos rápidos a qualquer tipo de informação requisitada e evitando assim a materialização total do *DataCube*, o que resulta imediatamente numa substancial redução de espaço alocado.

O objectivo das técnicas de *clustering* consiste na identificação de focos densos de dados para, desta forma, as estruturas multidimensionais que registam os dados poderem ser alteradas de modo a garantir melhores tempos de resposta e redução de acessos ao disco [Roussopoulos et al., 1997] [Markl et al., 1999].

Outra das soluções apresentadas para reduzir a perda de desempenho, e sobre a qual o presente trabalho incide, consiste na utilização de estruturas de indexação multidimensionais. Sendo certo que seria possível aplicar índices unidimensionais em ambientes OLAP, esta não seria uma abordagem adequada para a obtenção de melhores tempos de resposta, devido à necessidade existente de aplicar separadamente, um índice para cada dimensão intersectando-se, posteriormente, os resultados de cada dimensão. A solução reside, portanto, na utilização de índices multidimensionais.

Nas estruturas multidimensionais, todas as dimensões existentes recebem tratamento equivalente, evitando-se desta forma operações de cálculo desnecessárias, o que conduz também a um aproveitamento de espaço. Estas es-

truturas organizam os tuplos de informação de acordo com a sua disposição espácio-temporal, recorrendo a técnicas de particionamento de espaço de forma a dimininuir quer o tempo de acesso aos dados como o número de acessos ao disco. Segundo Sarawagi [Sarawagi, 1997] classificam-se os métodos de indexação em quatro classes:

1. A primeira classe consiste em métodos baseados em *arrays* multidimensionais.

2. Os métodos da segunda classe são baseados em *bitmap*.

3. Os métodos hierárquicos, que definem a prioridade das dimensões.

4. Os métodos da quarta classe, que incluem índices multidimensionais originalmente desenhados para dados espaciais.

2.2.1 Métodos baseados em *arrays* multidimensionais

Conceptualmente, a estrutura de um sistema OLAP pode ser vista como um conjunto de *arrays* multidimensionais cujos atributos chave representam as coordenadas nos respectivos eixos. Caso o cubo fosse denso (o que significaria que os dados seriam aproximadamente contíguos) o método ideal de indexação seria a aplicação directa de um *array* multidimensional. Para a obtenção de qualquer região de dados, seria suficiente cruzar a informação disponível para cada dimensão. No entanto, grande parte dos dados contidos em ambientes OLAP não são contíguos ou seguem uma tendência exacta. Isto levou à apresentação de alternativas com o objectivo de manusear todos os dados tentando-se manter tão fiel quanto possível à ideologia original.

Numa das alternativas apresentadas, define-se quais as dimensões que são consideradas como dispersas e quais as densas. É criado então um índice com o formato de uma árvore contendo o conjunto de valores pertencentes às dimensões dispersas (figura 2.3). Em cada folha da árvore definem-se apontadores para os *arrays* multidimensionais formados com o conjunto de valores das dimensões densas onde, em cada célula, estão contidas as medidas. Esta abordagem permite salvaguardar tempo percorrendo o índice em forma de árvore e espaço pois, através de uma função de mapeamento, torna-se possível comprimir os arrays multidimensionais das folhas do índice de modo a não conterem nenhuma célula vazia.

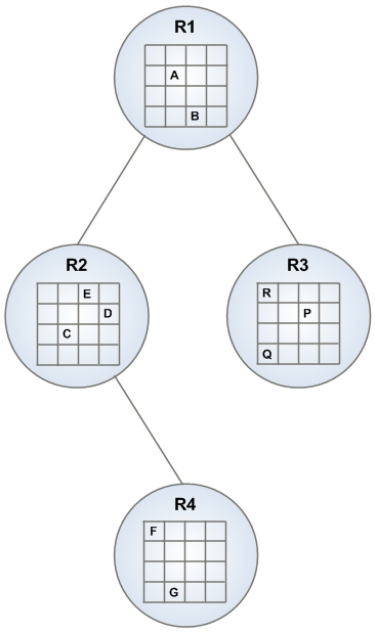

Figura 2.3: Índices com *arrays* multidimensionais

Considere-se, como exemplo, a venda de um produto, que existem as dimensões produto, região, cliente e loja, sendo que se consideram as dimensões "região" e "loja" como dispersas e "cliente" e "produto" apresentam-se como dimensões densas. Suponha-se ainda que a pergunta executada teria sido sobre o volume de vendas total de uma determinada loja. Assim que um dos parâmetros directos de procura fosse um elemento de uma dimensão dispersa, o índice seria percorrido, neste caso, procurar-se-ia pela área e pela loja até se encontrar uma folha da árvore. Aí, o *array* multidimensional seria percorrido devolvendo todas as transacções existentes na loja em questão.

2.2.2 Índices baseados em*bitmaps*

Quando os dados são assumidamente dispersos, a utilização de *arrays* multidimensionais poderá não ser uma boa escolha. Uma outra opção adequada seria a indexação separada de cada dimensão utilizando índices baseados em*bitmaps* [e D. Shasha, 1996].

A cada dimensão do cubo está associado um índice *bitmap*. Este, no seu formato mais simples, não é mais do que uma árvore B-Tree onde cada folha

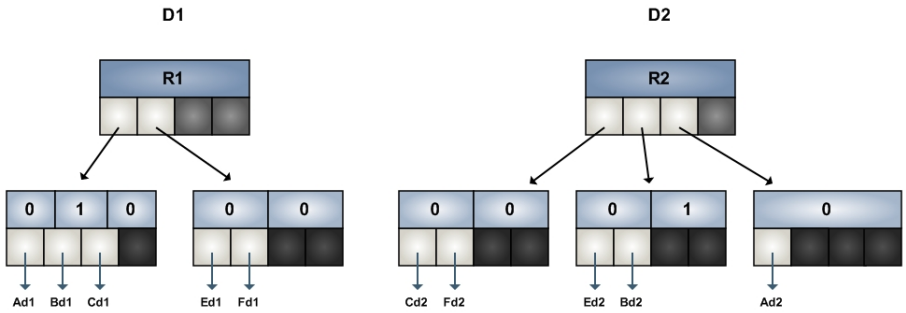

Figura 2.4: Índices baseados em *bitmaps*

contém um *bitmap* (figura 2.4). Dado um determinado valor, o seu *bitmap* é constituído por um *array* de *bits* cada um representando uma linha na tabela de factos. Como o nome indica, trata-se de um mapa ou *array* de *bits* em que cada um corresponde a uma célula do cubo e o seu valor só será "1" caso seja idêntico ao valor do parâmetro pesquisado. Como se pode observar na figura 2.4, é efectuada uma pesquisa pelo ponto B.

As *point queries* nas quais seja necessária a pesquisa em mais do que uma dimensão podem ser respondidas através da intersecção dos *bitmaps*. Os operadores *OR* e *AND* tornam possível a execução de *range queries*. Esta abordagem tem como vantagens:

- Na sua generalidade, os operadores binários operam mais rapidamente que os seus congéneres decimais. Torna-se relativamente simples e com pouco custo devolver os resultados de uma *range query*, visto que para tal apenas é necessário percorrer o *bitmap* em determinada ordem. Por tudo isto, para dados com baixa cardinalidade, é possível obter altos índices de desempenho sem ocupar muito espaço.
- Todas as dimensões recebem tratamento equivalente, ao contrário do que acontece nos métodos baseados em *arrays* multidimensionais, sendo que se manuseia as dimensões dispersas e densas de formas iguais.

As maiores desvantagens de índices baseados em *bitmaps* são:

- Aumento de espaço para guardar os *bitmaps* em ambientes de alta cardinalidade.
- Actualizações demoradas e pouco eficientes, pois uma inserção pode conduzir a grandes alterações nos *bitmaps*.

Em suma, esta abordagem torna-se viável quando o domínio de cada atributo é limitado, e em ambientes de baixa cardinalidade e preferencialmente estáticos, sem actualizações. É um dos métodos com maior proeminência no mercado, no entanto, os fabricantes comprimem, aplicam métodos híbridos e utilizam técnicas de criação de *bitmaps* dinâmicos de modo a reduzir o impacto das desvantagens nos utilizadores.

2.2.3 Métodos de indexação hierárquicos

Nesta proposta, um índice no formato de uma árvore é construído em primeiro lugar para uma determinada dimensão, tipicamente a que ocupa um lugar de maior importância para o agente de decisão, ficando todo o índice organizado consoante essa dimensão [e D. Shasha, 1996]. Repete-se o processo recursivamente para as restantes dimensões, ficando estas encadeadas entre si (figura 2.5).

A grande vantagem deste método é que a informação contida nos níveis mais altos é, habitualmente, acedida com mais frequência e pode ser devolvida com uma maior rapidez que os restantes dados. É também exequível uma permutação de níveis de modo a ser possível procurar por qualquer dimensão com os benefícios de esta ser colocada no topo do índice.

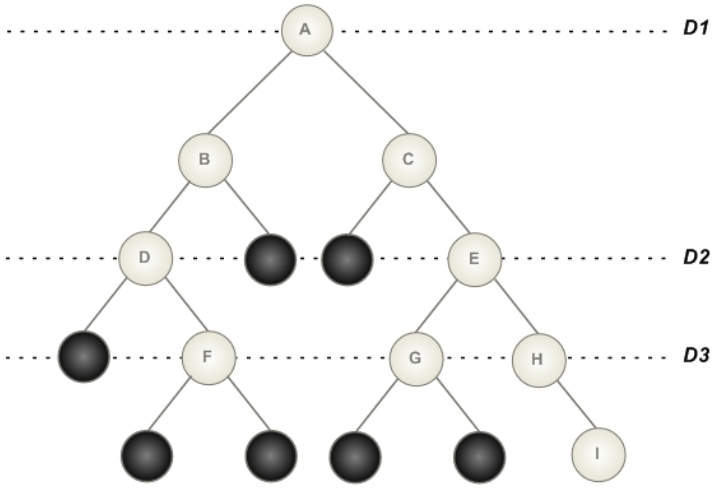

Figura 2.5: Métodos de indexação hierárquicos

As desvantagens dizem respeito ao facto de o tamanho do índice aumentar

fácil e rapidamente, o que conduz a um *overhead* de gestão e de espaço, existindo uma perda de desempenho. Ao contrário do que acontece em grande parte dos métodos de indexação que recorrem à memória para o processamento do índice e apenas ao disco para aceder e devolver os dados, no presente método torna-se impraticável tal abordagem, o que também leva a uma redução da performance.

2.2.4 Índices Multidimensionais

Outra alternativa para a indexação de dados em ambientes OLAP diz respeito à aplicação de índices multidimensionais especificados para dados espaciais, ver [Güting et al., 1994] para um estudo genérico sobre o assunto. Esta alternativa, apesar de existente em alguns produtos comerciais, não foi largamente adoptada pelos fabricantes. A fraca escabilidade e o mau desempenho em ambientes de muitas dimensões são alguns dos problemas apontados, no entanto, alguns destes índices oferecem certas vantagens que poderiam trazer proveitos na indexação de dados em OLAP. Uma das suas principais características diz respeito ao tratamento equivalente das dimensões sem o *overhead* de espaço inerentes aos métodos de indexação hierárquicos ou de processamento e que é o resultado dos índices baseados em *bitmaps* [Sarawagi, 1997] [Gupta et al., 1997].

O tema da presente tese incide na metodologia dos índices multidimensionais: o estudo de índices multidimensionais recorrendo a quadtrees como estruturas de indexação, bem como a implementação de algumas funcionalidades que aumentem a performance em determinados ambientes OLAP. De seguida, as estruturas multidimensionais quadtrees serão estudadas.

3
Quadtrees

As Quadtree [Finkel and Bentley, 1974a] representam um tipo de estruturas hierárquicas de dados que se baseia no princípio da decomposição recursiva do espaço.

Devido à revolução de placas gráficas 3D observou-se um consequente *boom* em vídeojogos. As Quadtree ganharam relevo por possuirem uma orgânica que lhes atribui a capacidade de representação do espaço de um modo simples e intuitivo, passando a ser bastante utilizadas na renderização do espaço em grande parte dos jogos actuais. Para além disso, a sua utilização extende-se pelas seguintes áreas:

- Indexação de bases de dados espaciais, sendo utilizadas em alguns sistemas comerciais, estando presentes, por exemplo, no *Oracle Spatial* [Kanth and Kothuri, 2002].
- Geometria computacional, entre outros, que ocupam um lugar de grande proeminência em sistemas de reconhecimento de padrões em imagens [Spann and Wilson, 1985].
- Sistemas de Informação Geográfica [Gahegan, 1989].
- Indexação Multidimensional [Gupta et al., 1997].
- Computação gráfica, onde se aplicam tanto no desenvolvimento de motores de jogo como em técnicas de *View frustum culling* [Güdükbay et al., 2002].

No trabalho que aqui se apresenta, a escolha das estruturas de indexação recaiu sobre as Quadtrees devido à sua capacidade de obterem altos índices

de desempenho na indexação de dados, à sua representação simples e intuitiva do espaço e à sua aptidão para resolver eficientemente as colisões de regiões, tão comuns em ambientes OLAP. No entanto, um dos seus maiores problemas reside no consumo de espaço que lhe está associado. Em cada nodo é preciso alocar memória para os apontadores dos seus filhos. Num universo k-dimensional em que k é um valor elevado, este tipo de índice exige uma enorme quantidade de memória que nem sempre será utilizada.

Na implementação de uma Quadtree, são diversos os factores que devem ser tidos em consideração, além da decomposição, para a obtenção de um melhor desempenho. Em [Samet, 1984] são apresentados alguns:

1. O tipo de dados que se pretende representar.
2. O modelo do processo de decomposição
3. O tamanho de cada quadrante, que pode ser variável ou estático.

A extensão de Quadtrees para a representação de objectos tridimensionais com a utilização de Octrees foi já proposta por diversos investigadores [Hunter, 1978] [Jackins and Tanimoto, 1983] [Meagher, 1981]. Tal extensão para um espaço euclidiano k-dimensional é também facilmente perceptível aplicando a mesma regra da decomposição recursiva do espaço em que cada nodo tem 2^k filhos, sendo k o número de dimensões do espaço.

Num universo a duas dimensões, no entanto, cada nodo intermédio contém quatro filhos, normalmente apelidados de NW, NE, SW e SE para as posições noroeste, nordeste, sudoeste e sudeste, respectivamente. Cada filho tem definido um limite máximo que representa o número de pontos que pode conter sem ser preciso o seu particionamento. Quando esse limite é atingido, particiona-se o nodo em questão. Caso a decomposição seja regular, o quadrante é dividido em quatro subquadrantes iguais, sendo esta abordagem particularmente útil no processamento de imagens. De outra forma, caso a decomposição não seja regular, os limites dos subquadrantes são definidos consoante os dados inseridos, através de atribuições pré-definidas ou calculadas dinamicamente.

Tal como acontece com outras árvores utilizadas no âmbito da indexação multidimensional, também nas Quadtrees existem diversas variantes. Algumas destas, as que foram mais relevantes para o trabalho desenvolvido, serão apresentadas nas secções seguintes.

3.1 As Variantes

Desde a sua formação, em 1974, que as Quadtree sofreram algumas alterações ao longo dos tempos. Aproveitando-se da sua aptidão para uma rápida indexação de dados, investigadores de diversas áreas cientificas diferentes propuseram as seguintes apresentadas nas secções seguintes.

3.1.1 Region Quadtree

A variante de Quadtree mais utilizada para a representação de regiões é denominada de Region Quadtree. Estas baseiam-se no particionamento do espaço através da decomposição regular de regiões e subregiões [Klinger, 1971]. O método repete-se recursivamente até cada folha ou bloco serem atingidos. Estes são a unidade mínima, não divisível, contida numa Region Quadtree e são definidos pelos seus limites e pela informação contida.

Habitualmente utilizadas no processamento de imagens, a uma Region Quadtree com profundidade n torna-se possível representar uma imagem que consiste em 2^kx2^k pixeis. São representadas por uma matriz de k dimensões, onde a raíz corresponde a toda a matriz e cada filho é um subquadrante desta. Assim, cada dado inserido será alocado através de todos os quadrantes adequados até atingir o bloco ao qual pertence. Os subquadrantes que contêm apenas dados da região são denominados pretos, aos que não contêm dados da região chamam-se brancos e os mistos são apelidados de cinzentos.

Devido à sua representação matricial, uma Region Quadtree não pode ser considerada verdadeiramente uma árvore. É mais correcto pensar nelas como *tries* ou *prefix tree*. Esta abordagem leva a um melhor manuseamento do espaço ocupado.

As Region Quadtree puras não são utilizadas em ambientes multidimensionais, não existindo por elas, nesta tese, mais do que um interesse intelectual, comum a todas as abordagens basilares e das quais resultam técnicas vantajosas para o tema. No entanto, delas derivam duas das mais influentes Quadtree nestes ambientes:

- MX Quadtree, que mantêm a representação matricial existentes nas Region Quadtree, no entanto, o seu particionamento funciona de forma diferente, dividindo equitativamente o espaço.

- PR Quadtree, uma estrutura híbrida que recebe influências tanto da Point

como da Region Quadtree. Tal como o nome indica, mostra-se eficiente tanto no manuseamento de pontos como de regiões de dados.

Estas árvores representam adaptações da Region Quadtree para o manuseamento de pontos e serão apresentadas nas secções seguintes.

3.1.2 Point Quadtree

Estas estruturas [Finkel and Bentley, 1974a] foram criadas por Finkel e Bentley em 1974 e representam uma adaptação da Árvore Binária de Procura orientadas ao manuseamento de dados multidimensionais. Partilham características comuns a todas as Quadtree, no entanto, o centro de particionamento é sempre um ponto, mantendo assim a informação tanto em nodos folha como em intermédios. O formato da árvore (figura 3.1) depende da ordem de inserção dos dados.

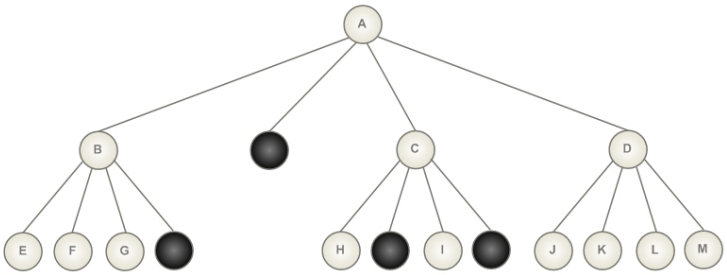

Figura 3.1: Point Quadtree

Os nodos são similares aos da Árvore Binária de Procura, sendo uma das maiores diferenças a existência de 2^k filhos, onde k representa o número de dimensões. Contém também uma chave decomposta em k segmentos, remetendo para o valor das coordenadas nos eixos de cada dimensão. Isto totaliza um total de 7 campos, caso se esteja a trabalhar num ambiente a duas dimensões. Os filhos ocupam 4 campos, os 2 campos seguintes são ocupados pelas coordenadas x e y e, por fim, o campo final é ocupado pelo valor da medida em causa. Se num ambiente bidimensional o número de campos aparenta ser razoável, facilmente se percebe que, em ambientes de alta dimensionalidade, a alocação de memória para os apontadores dos filhos não é sustentável. Tome-se como exemplo um ambiente com 12 dimensões o que conduz a 4096 filhos por nodo. Convém, portanto, recorrer a técnicas

na implementação de uma Point Quadtree para não existirem consumos de memória desnecessários

3.1.3 K-D Tree

As Point Quadtree possuem várias características que as levam a obter um pior desempenho face a outras alternativas. Para começar, cada comparação entre um registo e o valor atribuído a um nodo requer que se teste as coordenadas de todas as k dimensões. Segue-se a grande quantidade de espaço ocupado quer por cada nodo folha devido à existência de todos os apontadores *null* existentes, quer pelos nodos intermédios, que crescem substancialmente com o aumento do número de dimensões.

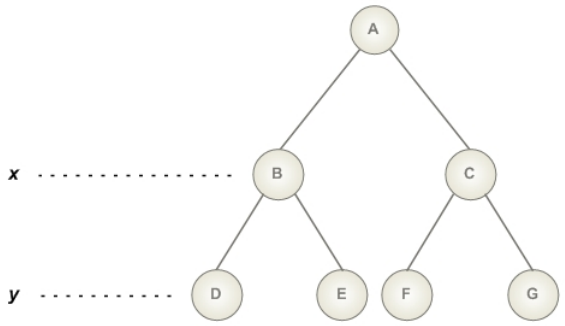

Figura 3.2: K-D Tree

As K-D Tree (figura 3.2) representam uma estrutura para armazenamento e indexação de dados multidimensionais. Tal como a Point Quadtree, também as K-D Tree - de k-Dimensional Tree - funciona de forma análoga às Árvore Binária de Procura com a alteração que, em cada nível diferente da árvore, testam-se diferentes dimensões do índice. A escolha da dimensão a ser percorrida é levada a cabo por um descriminador.. Em cada nível da árvore existem apenas duas ramificações que tomam menores e maiores valores do nodo e do eixo em questão, à esquerda e à direita, respectivamente. Proposta por Bentley em [Bentley, 1975], representa uma abordagem com as seguintes melhorias:

1. Assumindo um espaço k-dimensional, reduz as comparações em cada nodo para 2, em vez de k. De forma semelhante, o espaço ocupado

pelos nodos terminais, bem como os restantes, não é significativo visto que o número de apontadores por nodo é 2.

2. Devido à sua organização, torna-se eficiente na procura por regiões ou pontos adjacentes.

Existem, basicamente, dois tipos diferentes de K-D Tree, a K-D Tree Original e a K-D Tree Adaptativa, sendo que as diferenças entre ambas consiste na política adoptada pelo discriminador, bem como no próprio armazenamento dos dados no índice.

Na K-D Tree original, o discriminador toma sempre o valor de uma das dimensões existentes. A escolha da dimensão a utilizar é realizada de forma alternada. A duas dimensões, significa que nos níveis ímpares o eixo a ser testado seria o x e nos níveis pares testar-se-ia a coordenada y. Na K-D Tree Adaptativa é possível adoptar outras alternativas para a determinação da dimensão a ser utilizada como discriminador. O ideal seria verificar qual a dimensão e qual o seu conjunto de valores que melhor divide os pontos em dois conjuntos. Para obter esse resultado, recorre-se a cálculos de média, média ponderada e variância.

Ao contrário do que acontece com a sua congénere original, os dados são apenas contidos nos nodos folhas, enquanto que os intermediários armazenam um valor associado a um discriminador, que identificam o *path* correcto para a construção do índice. A estrutura da árvore não depende da ordem em que são inseridos, estes são conhecidos *a priori* o que leva a uma melhor organização e a uma consequente melhoria no desempenho. No entanto, esta característica faz com que a utilização deste género de índices não seja exequível em sistemas de suporte à decisão.

3.1.4 MX Quadtree

Existem diversas abordagens possíveis na adaptação de uma Region Quadtree para a representação de pontos, sendo que a MX Quadtree (MX de matriz) está estruturada de forma similar, com a diferença que os pontos são tratados singularmente. Esta é obtida através de uma função de mapeamento que recebe, como parâmetros, os valores das coordenadas dos pontos a inserir.

Actualmente, as MX Quadtree são utilizadas em sistemas de manipulação de matrizes, nomeadamente em videojogos e em Sistemas de Informação Geográfica. O objectivo é tirar partido da compressão existente para atingir me-

lhores índices de desempenho e tempos de execução [Saunders et al., 1989] [Hall and Wise, 1987]. Outros estudos sobre o seu funcionamento e vantagens podem ainda ser encontrados em [Abdali and Wise, 1989].

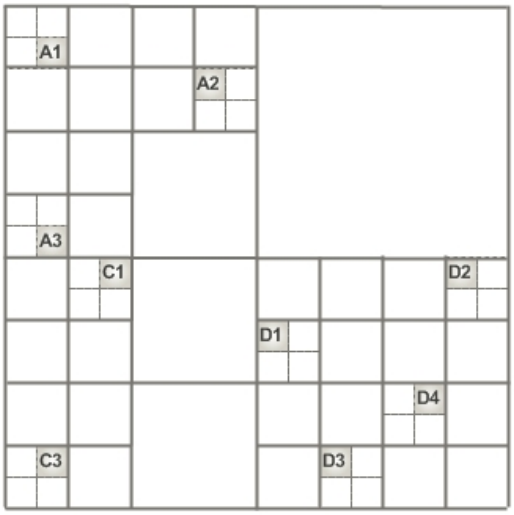

Figura 3.3: MX Quadtree

Por se tratar de uma representação matricial (figura 3.3) a união de nodos adjacentes pode e deve ocorrer, salvaguardando-se assim muitas operações desnecessárias. No entanto, a junção só deve existir quando os nodos são nulos, caso contrário perder-se-ia informação. Em cada célula da matriz, existe um campo adicional, indicando se se trata de um nodo preto, cinzento ou branco, que representa, respectivamente a existência total ou parcial ou a ausência de pontos nessa região.

Cada nodo da árvore contém, para além de uma indicação sobre a sua posição na árvore (apontando se se trata de um nodo folha ou intermédio), informação sobre os 2^k filhos. Não existe necessidade de guardar as coordenadas pertencentes à chave uma vez que é derivável através da função de mapeamento.

Dada as suas características, o formato de uma MX Quadtree é independente da ordem pela qual os dados são inseridos ou o número de dados presentes no índice. Por defeito, assume-se que cada ponto é único, existindo no entanto diversas abordagens que lidam com o manuseamento de pontos com as mesmas coordenadas.

Um dos problemas apontados a este índice, bem como à grande maioria de representações matriciais, é o excesso de alocação de memória que lhes é inerente. Não partilham da compressão utilizada nas Region Quadtree devido ao facto de serem orientadas ao tratamento de pontos. Não é, também, adequado quando o domínio de pontos não é discreto ou finito, pois, sendo a decomposição regular, torna-se impossível prever as consequências adversas que daí poderão resultar em ambientes multidimensionais onde não exista a garantia de uniformidade.

Outra desvantagem que lhes é característica em ambientes multidimensionais diz respeito à sua decomposição. Para cada registo inserido, a MX Quadtree armazena-o numa célula já na sua unidade mínima e indivisível. Esta metodologia leva à criação de regiões desnecessários que conduzem à realização de testes que poderiam ser evitados.

3.1.5 PR Quadtree

A Point Region Quadtree [Samet, 1990] representa uma adaptação da Region Quadtree para o manuseamento de pontos, possuindo no entanto, estrutura e organização similares. A cada ponto é associado um quadrante e, ao contrário do que acontece com as Point Quadtree, o particionamento é, por norma, regular, sendo que cada quadrante possui, no máximo, um ponto.

A principal desvantagem da PR Quadtree reside no facto de o nível máximo para a existência de *splitting* depender da distância mínima entre dois pontos sendo esta, no entanto, regular (figura 3.4). Caso existam dois pontos muito próximos um do outro, a decomposição da árvore pode ocorrer muito frequentemente. Não é, por isso, aconselhada a sua utilização em ambientes densos, pois o seu tamanho e forma dependem dos dados que estão actualmente na árvore.

Nos nodo de uma PR Quadtree estão guardados os 2^k apontadores para os seus filhos, bem como uma referência ao tipo de nodo de que se trata, isto é, nodo folha vazio, preenchido ou nodo intermédio. Contém ainda as coordenadas correspondentes e os valores a guardar. Em [Gargantini, 1982], Gargantini apresenta uma implementação sem apontadores da PR Quadtree, reduzindo assim a memória ocupada pelo índice.

Apresentam melhorias face às MX Quadtree, visto que apenas serão alocados os nodos necessários à medida que os registos vão sendo introduzidos. Poupa-se, assim, uma larga quantidade de testes e ocupação de memória

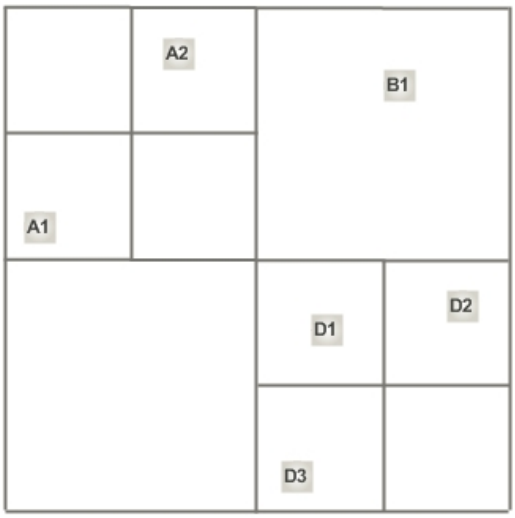

Figura 3.4: Point Region Quadtree

desnecessariamente. Esta característica levou-as a serem eleitas na presente tese para a realização de optimizações em ambientes multidimensionais.

3.2 Operações de Inserção e de Pesquisa

.

3.2.1 Region Quadtrees

Numa Region Quadtree, quando se pretende inserir um registo, efectua-se uma busca pelo mesmo. Em cada nível do índice compara-se o registo com as coordenadas existentes escolhendo-se, consequentemente, o subquadrante pelo qual deverá prosseguir a pesquisa. Quando finalmente se encontra a localização na qual se deveria inserir o registo, verifica-se se essa determinada região se encontra ocupada. Caso se trate de uma região cinzenta, particiona-se, sempre de forma regular, essa zona da árvore de modo a separar as regiões brancas e pretas para, desta forma, o registo poder ser alocado numa região que para ele tenha espaço alocado.

O facto de a decomposição ser regular afecta toda a estrutura do índice, retirando-lhe flexibilidade. Torna-se proveitosa quando é conhecida e dado como certa, previamente, a uniformidade do espaço. No entanto, se esta ca-

racterística os torna úteis para o processamento de imagens, retira-lhes vantagens quando se trata da indexação em ambientes OLAP.

O factor mais interessante, apesar de irrelevante em ambientes multidimensinais - a menos que se utilize métodos de indexação hierárquicos para posterior intersecção de todas as dimensões - é a possibilidade de inserção de áreas completas. Esta característica torna as Region Quadtree fundamentais para o processamento de imagens pois torna-se, por vezes, desnecessário procurar a informação até aos nodos folha, caso estes estejam inseridos em toda uma região que partilhe a mesma característica.

Grande parte das operações utilizadas numa Region Quadtree, tais como a inserção, simplesmente percorrem a árvore utilizando um algoritmo *preorder* [Samet, 1984]. Cada elemento da matriz contém uma informação sobre se este pertence, ou não, a uma determinada área procurada. Caso um quadrante seja totalmente sobreposto pela área pesquisada, uma agregação é efectuada entre esse quadrante e os seus subquadrantes, tornando-se toda uma área comum. Esta abordagem permite um melhoramento significativo no desempenho de *range queries*.

3.2.2 Point Quadtree

Os registos são inseridos de forma análoga à Árvore Binária de Procura. Essencialmente, procura-se pelo registo a inserir com base nas suas coordenadas. O registo é inserido assim que se atinge o final da árvore, isto é, quando o nodo onde é realizada a comparação é nulo. Nos casos em que as coordenadas de um ponto o situem entre dois filhos de um nodo, convencionou-se em [Samet, 1990] que as fronteiras correspondentes ao menor valor de um nodo estão fechadas, enquanto que as de valores mais elevados permanecem abertas à inserção.

O trabalho dispendido a construir uma Point Quadtree está directamente relacionado ao seu TPL (*Total Path Length*) [Knuth, 1997], sendo que o custo médio de inserção é de $O(\log_4 N)$. O desempenho da inserção numa Point Quadtree é dependente da ordem da inserção dos pontos. O pior caso ocorre quando a posição de cada registo inserido é, sucessivamente, a de filho do nodo mais profundo da árvore. Como consequência disto, tem existido interesse em reduzir o TPL. Em [Samet, 1990], são apresentadas duas técnicas para obter essa redução:

1. Finkel e Bentley em [Finkel and Bentley, 1974b] assumem que todos os

nodos são conhecidos *a priori*. Define-se uma Point Quadtree optimizada em que dado um nodo *A*, o conjunto de todos os seus subnodos não excede a metade do número contido pelo nodo-pai de *A*. Este método requer que todos os registos sejam percorridos e ordenados antes de serem inseridos, sendo que a raiz da árvore será o registo cujo valor mais se aproxime da média de todos os dados, e os restantes serão agrupados para formar os respectivos filhos. Este método aplica-se recursivamente a toda a árvore.

2. Em [Overmars and van Leeuwen, 1982] discute-se uma abordagem alternativa que representa uma adaptação dinâmica ao método supramencionado, o que significa que a Point Quadtree optimizada vai sendo construída consoante em que os registos vão sendo inseridos, mantendo-se sempre um balanceamento da árvore.

Naturalmente, em ambientes OLAP onde se realizam actualizações periodicamente, o primeiro método apresentado não é adequado, pois seria necessário recriar toda o índice em cada operação.

A eficiência de uma Point Quadtree reside na sua operação de procura. Em cada ponto, efectua-se uma comparação entre todas as coordenadas da chave com as do registo a inserir. Consoante o resultado, será escolhido o filho para a comparação seguinte, repetindo-se este método recursivamente. Supondo que é realizada uma procura por uma área que está contida apenas num dos nodos, não existe a necessidade de procurar pelos restantes. Desta forma, reduz-se substancialmente a quantidade de testes a realizar. No entanto, esta eficiência esbate-se quando o número de dimensões aumenta. Isso acontece não só porque o espaço necessário para guardar uma Point Quadtree aumenta exponencialmente com o aumento do número de dimensões, mas também porque quer o número de nodos terminais que não contêm nenhum valor, quer a quantidade de testes a efectuar por nodo sofrem um incremento significativo.

Ainda a este respeito analisou-se em [Bentley et al., 1977] [Lee and Wong, 1977] o custo de uma operação de pesquisa numa Point Quadtree. Concluiu-se que atingia, no pior caso, $O(k.N^{1-1/k})$ onde k representa o número de dimensões.

3.2.3 K-D Tree

Também na K-D tree os registos são inseridos de forma semelhante à Árvore Binária de Procura. Pesquisa-se pelos valores do ponto a inserir e este será inserido no local onde a procura terminar.

Tal como acontece com as Point Quadtree, o formato final da árvore depende da ordem de inserção dos registos e a quantidade de trabalho despendido na sua criação é igual ao seu TPL. Em [Bentley, 1975] demonstra-se que o custo médio desta operação é de $O(\log_2 N)$, em que N representa o número de inserções ocorridas. O pior caso possível acontece quando, a cada inserção, o registo é inserido no nodo mais profundo existente, aumentando assim, drasticamente, o TPL. Em [Bentley, 1975] são apresentados dois métodos para diminuir o tamanho de uma K-D Tree:

1. Conhecendo todos os registos *a priori* é possível, de forma equivalente ao que acontece com as Point Quadtree, obter uma árvore optimizada.

2. Modificar a K-D Tree para para uma estrutura em que a informação é guardada apenas nos nodos folha mantendo, em cada nodo intermédio, um valor que define em que região um ponto será inserido. Caso o valor do registo a inserir nesse eixo de coordenadas seja menor, a pesquisa continua recursivamente pela subárvore; caso contrário, será a subárvore da direita a ser pesquisada. O processo repete-se recursivamente até se atingirem os nodos folha, aí, o registo será inserido numa lista ligada. Também para este método é necessário o conhecimento prévio dos registos.

3. É possível, tal como nas Point Quadtree, manter um balanceamento da árvore através de um algoritmo que, dinamicamente, verifica e equilibra em caso de desajuste. Naturalmente, métodos como o referido contém um *overhead* de gestão na inserção, compensado porém, em performance, nas operações de pesquisa.

Tal como acontece com as Point Quadtree, as K-D Tree são úteis em aplicações que envolvam numerosas operações de procura, pois conduzem a pesquisa pelos nodos relevantes, o que exclui grande parte da árvore. Em cada nível da árvore testa-se um eixo de coordenadas diferentes, e esta comparação resulta na continuação da pesquisa através de um dos seus dois filhos.

As K-D Tree são geralmente eficientes em consultas que envolvam proximidades. Esse tipo de consulta caracteriza-se por receber como parâmetros, limites de uma determinada região e o resultado obtido representa o conjunto de pontos existentes nessa região. O índice é percorrido de forma semelhante à Árvore Binária de Procura onde, em cada nível, é realizada uma comparação entre o ponto existente e os parâmetros de pesquisa. O processo repete-se na subárvore da esquerda ou da direita e assim recursivamente até a pesquisa terminar.

Como pode ser observado, numa operação de pesquisa existe uma larga possibilidade de apenas alguns nodos do índice serem percorridos para se encontrarem os pontos que satisfaçam a condição de consulta. Esta possibilidade de poupar drasticamente processamento, é das maiores vantagens existentes na utilização de K-D Tree como estruturas de indexação multidimensionais. A sua performance foi estudada por Lee e Shacter em [Lee and Schachter, 1980] concluindo que, no pior caso, o custo de uma operação de pesquisa numa K-D Tree é de $O(k.N^{1-1/k})$.

3.2.4 MX Quadtree

Tal como nos casos anteriores, um nodo é inserido executando-se uma pesquisa baseada na localização do ponto na matriz. Em cada nodo, ou região da matriz, determina-se antecipadamente o tipo de nodo. Caso seja preto ou cinzento, a procura continua nessa região; caso seja branco a busca termina, evitando-se assim a perda de tempo e processamento em cálculos desnecessários.

Devido ao facto de todos os dados serem armazenados em unidades mínimas e indivisíveis, a inserção de um registo ocorrerá assim que encontre um nodo branco, podendo este ser particionado, ou, sendo já de si um nodo folha, determina a localização da inserção do registo.

Independentemente da inserção, o ponto será introduzido num bloco, a mínima e indivisível unidade de uma MX Quadtree. Toda a decomposição - regular - ocorre desde a raiz da árvore até ao respectivo bloco, retirando assim uma certa flexibilidade a este tipo de índices. O índice de crescimento da árvore torna-se maior do que o das suas congéneres, podendo levar a um desbalanceamento, à criação desnecessária de nodos e ligações e aos consequentes testes comparativos que poderiam ser evitados.

Uma operação de pesquisa na MX Quadtree ocorre de forma similar à de

uma Region Quadtree. Em cada região, caso se trate de uma cinzenta ou preta, é realizada uma comparação entre as suas coordenadas e os parâmetros de pesquisa. Esta operação é repetida de forma recursiva até ser encontrado um nodo onde se possa alocar espaço para os dados a serem inseridos. Se em grande parte das áreas em que são utilizadas é possível procurar por áreas directamente, em ambientes OLAP tal não é exequível perdendo-se assim uma das maiores vantagens associadas a este tipo de índices.

O custo do pior caso da procura é de $O(N + 2^n)$, onde N é o número de registos a serem inseridos e n representa é a profundidade da árvore.

3.2.5 PR Quadtree

A inserção de pontos na PR Quadtree ocorre de forma análoga à da Point Quadtree com a diferença de que os dados estão apenas contidos nos nodos folha. Inicialmente, cria-se o registo do ponto com a devida informação inserida. De seguida, geralmente, pesquisa-se pelo quadrante ao qual o registo a inserir pertence e, caso esteja ocupado por um ponto de coordenadas diferentes, divide-se o quadrante repetidamente até se atingir uma parcela que se ajuste. Este método é denominado de *splitting* e a sua principal desvantagem reside no facto de o *splitting* depender da distância mínima entre dois pontos. Caso existam dois pontos muito próximos um do outro, a decomposição da árvore pode ocorrer muito frequententemente. Não é, por isso, aconselhada a sua utilização em ambientes densos, pois o seu tamanho e forma dependem dos dados que estão actualmente na árvore.

Apesar das desvantagens apresentadas, a PR Quadtree tem a seu favor o facto de a inserção não colocar automaticamente um registo num bloco. A decomposição vai ocorrendo à medida das necessidades do índice, poupando-se assim tempo de execução, processamento e testes.

O custo médio desta operação é de $O(k.N)$, sendo que a forma final da árvore não depende da ordem de inserção dos registos.

A procura na PR Quadtree também é levada a cabo de forma semelhante à da Point Quadtree. Em cada quadrante são efectuados testes comparativos entre os parâmetros de pesquisa e os limites atribuídos cada nodo, concluindo a operação num nodo folha. Não ocorre, ao contrário do que acontece com as MX Quadtree, a realização de testes desnecessários, uma vez que o crescimento do índice é proporcional aos pontos que lá foram inseridos. O pior dá-se ocorre quando se procura por um rectângulo cujos lados são paralelos

a um quadrante e, nesse caso, o custo é de $O(N + 2^k)$.

4

Extensão da Point Region Quadtree

Actualmente, a variante mais explorada nestes sistemas é a PR Quadtree devido à sua capacidade de manusear pontos com eficiência não prescindindo, no entanto, dos seus altos índices de desempenho no que respeita ao tratamento de regiões. Por este motivo, a estrutura base utilizada ao longo do desenvolvimento desta tese foi a PR Quadtree. Foram, todavia, estudadas diversas técnicas de *tuning* do índice de modo a optimizar a sua performance, enquanto se minimizam os custos de processamento e memória.

Tendo em conta a natureza das operações mais frequentes neste tipo de ambientes e as respectivas necessidades,foi dado maior destaque ao seu desempenho, em detrimento do espaço ocupado que, à partida, já é elevado.

Como foi referido, o índice desenvolvido é, na sua essência, uma Point Region Quadtree. No entanto, foram alteradas algumas características de forma a melhorar o desempenho em relação às Point Region Quadtree nos ambientes testados. Em seguida apresentam-se as alterações efectuadas:

1. Ao contrário do que acontece nas Point Region Quadtree, a decomposição da árvore não depende da distância mínima entre dois pontos.
2. O algoritmo utilizado na procura não é o utilizado nas Point Quadtree, mas um melhoramento desenvolvido ao longo deste projecto, que tem garantido melhores desempenhos.
3. A cada nível de um índice, é fornecida informação sobre um hipotético ponto central comum a todos os filhos.
4. Os nodos finais, onde a informação está guardada, não são representados por uma matriz, mas por tabelas de *hash*.

5. A distribuição dos pontos do índice é equitativa, ocorrendo só em determinadas ocasiões, visando a redução do TPL.

6. É efectuada uma ligação entre os nodos adjacentes da árvore, relacionando regiões diferentes, mas próximas em cada dimensão.

Nas secções seguintes, será apresentada a estrutura da árvore, bem como as alterações realizadas e os motivos que levaram à sua aplicação.

4.1 Alterações e motivação

4.1.1 Decomposição irregular

A decomposição utilizada nas Point Region Quadtree é extremamente ineficiente em ambientes densos sendo, no entanto, indicada para ambientes em que exista uma uniformização dos dados. O particionamento utilizado neste projecto representa uma tentativa de melhoramento para ambos os ambientes, em geral. Existente nas Point Quadtrees, o *splitting* de cada quadrante depende única e exclusivamente dos dados do nodo em questão, tendo-se utilizado a média ponderada entre os pontos como algoritmo de particionamento.

Cada nodo folha contém um valor para o *bucket* limit, que representa o número de pontos máximo que pode guardar, e quando esse limite é atingido, ocorre um particionamento do nodo. Esta operação consiste, basicamente, em 3 pontos:

- Determinação de um ponto central, seguindo as normas da decomposição regular.

- Atribuir um peso P a cada região consoante o número de nodos aí residentes.

- Efectuar a média ponderada entre as coordenadas de cada dimensão dos pontos existentes pelo seu respectivo peso. Aplicam-se os vectores resultantes desta operação para a translação do ponto central ocorrendo, a partir daí, a decomposição.

Como se pode ver na figura 4.1, a aplicação desta metodologia alivia as regiões mais ocupadas do nodo. Ao lhes reduzir o espaço leva a que, pos-

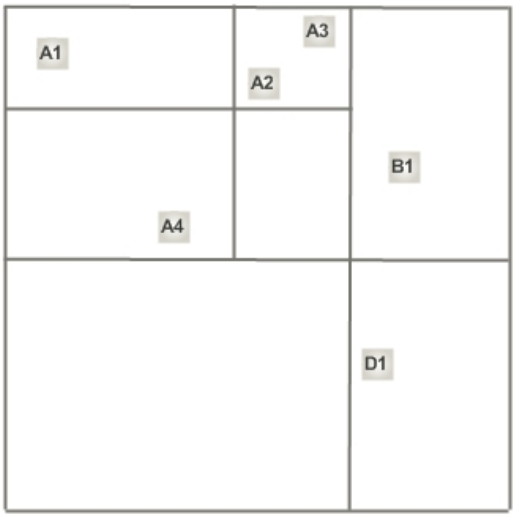

Figura 4.1: PR Quadtree: Decomposição irregular

teriormente, uma maior quantidade de registos dirigidos a esse nodo serão inseridos nas outras regiões.

Esta abordagem, conduz a uma maior uniformização do índice para a grande maioria de ambientes existentes. No caso de ambientes densos permite uma adequação no particionamento de regiões que resulta na semelhança em tamanho entre regiões adjacentes.

4.1.2 Algoritmo de pesquisa proposto

O algoritmo de procura utilizado nas Point Region Quadtree representa uma extensão do algoritmo existente numa Árvore Binária de Procura para ambientes k-dimensionais. No entanto, quando k é elevado, esse algoritmo implica ainda a realização de uma grande quantidade de testes de modo a determinar o subquadrante indicado para continuar a operação. O algoritmo utilizado, mais à frente descrito em detalhe, permite a redução do número de testes para um, no caso de não existir *overlapping*, independentemente do valor de k. Caso ocorra *overlapping* dos nodos, então o mesmo algoritmo percorre os subquadrantes que estão sobrepostos à região pesquisada.

Segundo a definição de Quadtree, cada nodo tem 2^k filhos, sendo k o número de dimensões. Cada nodo contém a informação sobre um ponto hipotético apontando o centro de determinada região e que é o ponto comum

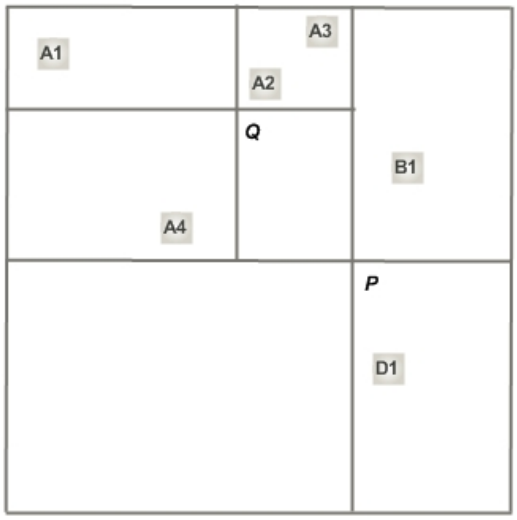

Figura 4.2: PR Quadtree: Pesquisa

a todos os filhos, registando os limites entre eles. A cada pesquisa, efectua-se um teste entre os parâmetros de procura e o ponto central do nodo em questão, comparando as coordenadas das respectivas dimensões, segundo a seguinte fórmula:

$$key = \sum_{d=1}^{k} \frac{f \times 2^{k-1}}{d} \tag{4.1}$$

A variável $f \in \{0, 1\}$ representa a inferioridade ou superioridade numérica, respectivamente, entre a coordenada do ponto na dimensão d. O número total de dimensões do ambiente é representado por k, e key indica o subquadrante pelo qual prosseguir a pesquisa.

Como exemplo, veja-se o caso presente na figura 4.2. Atribuindo os valores aos pontos $A3 \rightarrow (19, 39)$, $Q \rightarrow (30, 20)$ e $P \rightarrow (17, 30)$, pretende-se efectuar uma pesquisa por $A3$ (1).

Inicialmente, efectua-se uma pesquisa no quadrante com ponto central P, procurando-se obter o filho através do qual prosseguir a pesquisa. Aplicando a fórmula proposta, conclui-se que o subquadrante em causa está armazenado com a chave 1, o que significa que é o quadrante a Noroeste de P. De seguida, repete-se o processo em Q sendo que, desta vez, o resultado obtido indica o filho a Nordeste de Q, que é um nodo folha. Por fim, procura-se o ponto na

Operação 1 PR Quadtree: Pesquisa

Require: Conjunto de dimensões d, Ponto de pesquisa X

Pesquisa no quadrante de ponto central P:

$key = \frac{0*2^1}{2} + \frac{1*2^0}{1}$

$key = 1$

Pesquisa no quadrante de ponto central Q:

$key = \frac{1*2^1}{2} + \frac{1*2^0}{1}$

$key = 2$

K-D Tree existente em todos os nodos folha.

Como se trata de um ambiente bidimensional, esta operação não aparenta ser relevante mas, em ambientes de alta dimensionalidade mostra-se uma melhoria significativa nos índices de desempenho.

A escolha de um *array* para guardar todos os nodos filhos pareceu a mais óbvia. No entanto, ao longo da realização de testes ficou claro que o espaço ocupado pelo índice era maior do que o necessário devido à alocação de memória para o *array* que, nos níveis mais baixos do índice, raramente era preenchido. Havia, portanto, uma alocação desnecessária de memória para nodos que poderiam nunca existir. A solução encontrada foi a mudança de estrutura de armazenamento.

O conjunto dos filhos de um quadrante é guardado numa tabela de *hash*. A escolha incidiu sobre esta estrutura devido à sua capacidade de alocar memória apenas para os nodos existentes, não perdendo, no entanto, consistência no que diz respeito aos nodos que ainda poderão ser criados. A chave indica a precisa localização do filho e é atribuida através do algoritmo já apresentado. Este repete-se então no filho com a chave key sucessivamente até se atingir um nodo folha. Nesse caso, a pesquisa continua através da K-D Tree.

4.1.3 Redução do *Total Path Length*

Numa inserção, o pior caso possível ocorre quando, sucessivamente, o nodo a introduzir é filho do último nodo inserido no índice. Neste caso, a árvore cresceria rapidamente perdendo-se, em larga escala, os índices de performance que poderia atingir. Naturalmente, não é comum a ocorrência deste tipo de situações, no entanto, é possível - e até provável - que, em determinados ambientes, a Point Region Quadtree cresça desbalanceada, pen-

dendo demasiado numa determinada direcção. A escolha do algoritmo de decomposição apresentado nas secções anteriores da presente tese atenua esta tendência. No entanto, de modo a melhorar a organização da estrutura da Quadtree, estudaram-se alternativas que permitem controlar o crescimento do índice.

A proposta feita nesta tese não consiste de uma operação de balanceamento total da árvore. Doug Moore em [Moore, 1995] estudou essa abordagem mas, para ambientes OLAP, trata-se de uma difícil implementação com repercussões significativas no que diz respeito ao desempenho na inserção de dados. Em vez disso, enveredou-se por uma abordagem mais minimalista: Balanceando parcial e momentaneamente, só em determinadas ocasiões (figura 4.4).

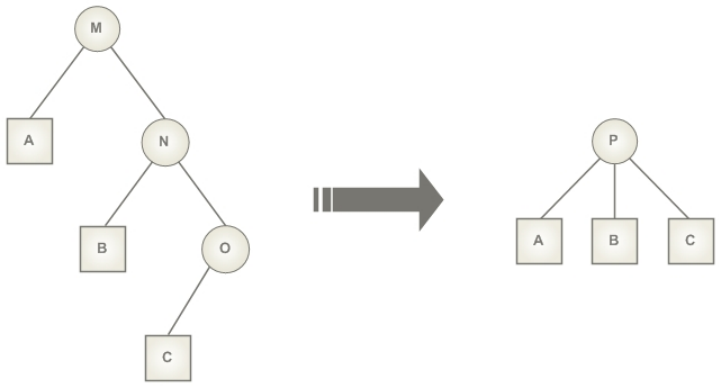

Figura 4.3: PR Quadtree: Redução do TPL

Considere-se o conjunto U, num ambiente bidimensional, tal que $U = A, B, C$, sendo que U representa um segmento desbalanceado do índice em que cada elemento representa um subnodo do anterior. Definem-se os pontos P, Q e R tal que:

- P toma o valor do ponto de U com menor coordenada no eixo do x. Seja, por exemplo, A.
- Nos pontos restantes, Q representam as mesmas coordenadas que o menor ponto de U no eixo y, suponha-se, B.
- R toma o valor do ponto restante, neste caso C.

Cria-se então um ponto hipotético, Z, tal que:

- $Zx > Px$, $Zx < Qx$ e $Zx < Rx$.
- $Zy > Qy$ e $Zy < Ry$.

O exemplo acima apresentado ocorreu num ambiente bidimensional, sendo intuitiva a adequação para um ambiente k-dimensional. Esta abordagem permite uma redução do TPL obtendo-se assim alguma influência, independentemente do ambiente, no crescimento do índice. Visto que não se trata de uma operação complexa, o seu custo é razoável.

A aplicação deste método não ocorre arbitrariamente; Apenas é executado quando, neste caso, Q e R são filhos únicos. Optou-se por esta alternativa dado que um balanceamento parcial do índice, com ramos com diversos filhos cada, torna a operação mais lenta e exige um elevado poder de processamento. No entanto, foi também abordado um método com implicações mais profundas no balanceamento da árvore, mas a relação entre o tempo de execução de pesquisa, o de balanceamento do índice , e o custo de processamento não se apresentou tão satisfatória como a da metodologia aqui descrita.

4.1.4 Estrutura de armazenamento: K-D Tree

Uma das diferenças entre a presente implementação e a Point Region Quadtree original diz respeito à localização dos registos de dados no índice. Enquanto que na árvore criada por H. Samet em [Samet, 1990], os pontos são armazenados apenas nos nodos folha da árvore, nesta alternativa - que tem como objectivo uma maior capacidade de manuseamento de pontos - estes são guardados ao longo de toda a árvore. Esta característica permite que, em muitas *queries* efectuadas, não seja necessário conduzir a operação de pesquisa até aos níveis mais baixos do índice.

Assim, as características apresentadas pela estrutura de armazenamento de pontos teriam de ser as seguintes:

- Altos índices de desempenho na operação de pesquisa.
- Reduzido número de comparações, de modo a ser possível poupar alguns recursos.

Em suma, o ideal seria uma estrutura leve de modo a não causar nenhum *overhead* de gestão ou processamento. Como o número de pontos a serem armazenados por esta estrutura é, tipicamente, baixo, dá-se preferência aos índices de desempenho nas operações relevantes, em detrimento da estrutura e organização da árvore, não deixando de ser, no entanto, uma estrutura multidimensional.

Devido a todos estes requisitos, a escolha incidiu sobre a utilização das K-D Tree como estruturas de armazenamento de pontos. As K-D Tree são estruturas multidimensionais com altos índices de desempenho nas operações de inserção e pesquisa, reduzindo ainda o número de testes necessários para atingir determinado registo. Tem, como inconveniente, uma pobre organização quando comparada com as Quadtree mas, uma vez que o número de registos por nodo é reduzido, esta contrariedade torna-se de pouca importância.

4.1.5 Ligação entre nodos adjacentes

Em ambientes OLAP, a esmagadora maioria de operações de pesquisa efectuadas são *range queries*. Interessa, aos agentes de decisão, saber o resultado final de uma temporada, o número de vendas de um produto, quantas pessoas aderiram a uma determinada campanha, etc. Mas, para devolver os resultados dessas pesquisas, torna-se necessário procurar por cada ponto e observar os seus valores.

Até à data, a Point Region Quadtree apresentada processa as pesquisas de forma semelhante às outras estruturas de indexação. Em cada nodo realizam-se uma determinada quantidade de testes de acordo com as coordenadas do nodo em questão e os parâmetros de procura e, consoante os resultados, prossegue-se a busca ao longo de um, ou mais, subnodos até toda a região seleccionada ser percorrida. No entanto, decidiu-se implementar neste índice, à semelhança do que aconteceu em 1972 em [Bayer and McCreight, 1972] nas B Tree, a ligação entre nodos adjacentes em cada dimensão.

Inicialmente, em cada nodo da Point Region Quadtree, existia um conjunto de $2k$ apontadores, em que k representa o número de dimensões presentes. Para cada dimensão, existiam dois apontadores que referenciavam o nodo anterior e o nodo sucessor na respectiva dimensão. Dado que o seu número seria sempre fixo, estes, estariam armazenados num *array*. Se esta abordagem representava uma melhoria significativa nas operações de pesquisa, o inverso ocorria nas operações de inserção, principalmente em ambientes de

grande dimensionalidade. Após a inserção de um determinado registo, torna-se necessária a atribuição de referências aos apontadores e a rectificação, nos nodos adjacentes ao ponto introduzido, dos seus próprios apontadores pois, na grande maioria das vezes, o novo nodo funcionava como *man in the middle*. Na figura 4.4 apresenta-se uma PR Quadtree com ligação entre os nodos adjacentes no eixo dos x.

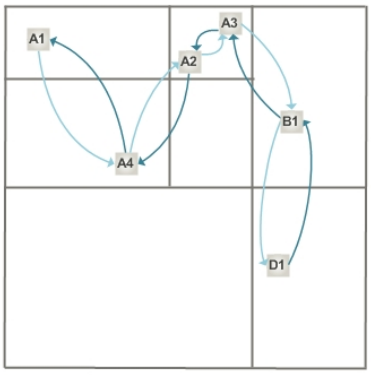

Figura 4.4: PR Quadtree: Ligação entre nodos adjacentes

A metodologia adoptada então, e que seria sempre a aconselhada a ter, foi a escolha de determinadas dimensões, as de maior proeminência e pelas quais as pesquisas seriam mais vezes efectuadas, para aplicar a ligação entre os nodos. Diminui-se o *overhead* referenciado, apesar de este ainda depender do número de dimensões existentes e escolhidas, mas obtém-se uma pesquisa com uma melhor performance.

No mínimo, e por ser sempre sequencial, aconselha-se que pelo menos 1 dimensão, a do tempo, utilize sempre este tipo de abordagem, visto que não é necessária qualquer reparação aos apontadores previamente definidos, exceptuando aqueles que estão a *null* no caso de se tratar do nodo adjacente anterior.

4.2 Estrutura proposta

O índice implementado consiste numa estrutura relativamente complexa (figura 5.3) que utiliza diferentes tipos de estrutura para o processamento das operações mais comuns em OLAP. Já foram referidas as alterações e a motivação subjacente a cada uma. A aplicação de todas estas resulta numa

estrutura que, em cada nodo, contém a seguinte informação:

- Um conjunto de coordenadas que define a localização dos nodos ao longo de cada dimensão.
- Os limites de cada nodo nas diferentes dimensões.
- Um ponto central que não existe de facto, mas funciona como um referencial sobre a região de cada nodo.
- um *array* que guarda os apontadores para os nodos adjacentes das regiões definidas como mais importantes.
- Nos nodos intermédios, apontadores para os filhos de cada nodo, armazenados numa tabela de *hash*.
- Uma K-D Tree onde os pontos existentes em cada nodo folha são guardados.

Se, de forma geral, se trata de uma estrutura um pouco pesada no que diz respeito a recursos, por outro lado, também é optimizada para operações de pesquisa. A existência de um ponto central conduz a uma redução no número de testes a efectuar, compensando assim, o gasto adicional de memória no seu armazenamento. A ligação entre os nodos adjacentes embora constitua um tipo de informação que se poderia evitar conter, facilita significativamente a pesquisa reduzindo o tempo de execução. As estruturas de armazenamento, tabelas de *hash* e K-D Tree, foram consideradas as de melhor relação desempenho / custo adicional.

Semelhante a todas as outras variantes das Quadtree, também no índice proposto torna-se necessário efectuar uma pesquisa para cada inserção de pontos. Consequentemente, também a operação de introduzir dados é indirectamente beneficiada pelas optimizações realizadas para as operações de pesquisa (figura 4.6). Esta figura demonstra as operações realizadas pelo índice na inserção de um ponto. Suponha-se, como exemplo, que se pretende inserir o ponto $A5$ na árvore, ocorrendo as seguintes operações:

- Ao aplicar o algoritmo de pesquisa proposto à raíz da árvore conclui-se que o ponto $A5$, caso exista, está contido no subquadrante de chave 1.
- A tabela de *hash* apresenta a localização do ponto mediante a sua chave e a busca prossegue por esse quadrante.

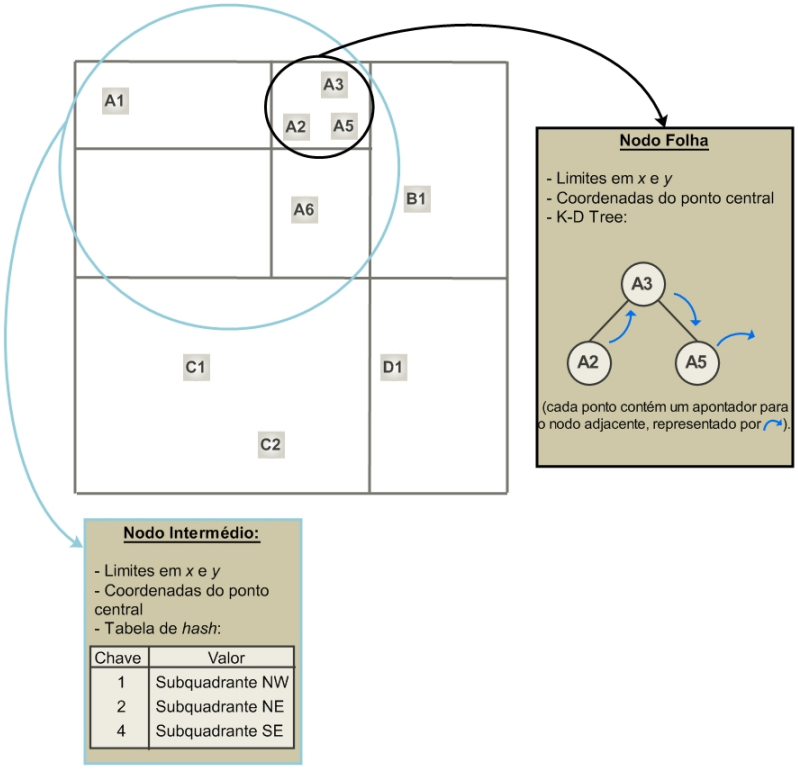

Figura 4.5: Estrutura do índice desenvolvido

- O método repete-se para subquadrante a Noroeste do ponto central inicial. Este devolve o valor 2, que indica o nodo folha onde $A5$ deverá ser inserido.
- O *bucket limit* deste quadrante foi atingido, logo torna-se necessária o seu particionamento, desfazendo a K-D Tree aí existente e criando novos nodos, com novas estruturas para o armazenamento dos dados.
- Os pontos já inseridos no índice, neste caso $A2$ e $A3$ são realocados para os nodos folha recentemente criados.
- O ponto $A5$, após o particionamento do nodo, é inserido num novo nodo folha que não contém pontos adicionais.

Como se pode observar, no exemplo apresentado não ocorre a operação de redução do *total path length*. No entanto, é exposto o restante funciona-

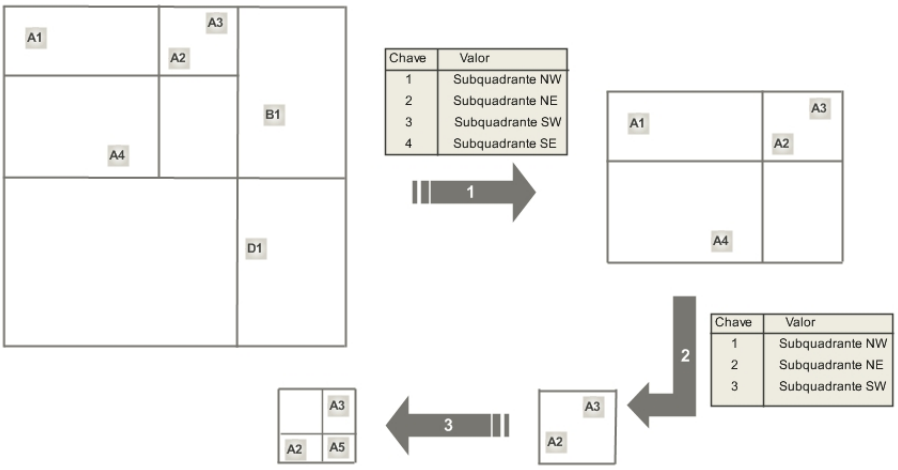

Figura 4.6: Inserção de um ponto na Point Region Quadtree Extendida

mento do índice, quer nas operações de pesquisa, quer nas de inserção.

No capítulo seguinte serão apresentados os resultados dos testes efectuados, bem como as implicações que as alterações demonstradas tiveram no desempenho do índice em diversos ambientes.

5

Avaliação

Periodicamente, e visto que um dos objectivos dos Sistemas de Suporte à decisão consiste na manutenção de um registo histórico dos dados, são executadas operações de inserção. Este processo pode revelar-se bastante dispendioso no que diz respeito a tempo e processamento. Em ambientes multidimensionais é necessário actualizar os índices à medida que os dados vão sendo inseridos no sistema e, visto que se trata sempre de grandes quantidades dados, a tarefa de criação, actualização e manutenção de um índice multidimensional, não é uma tarefa simples ou trivial. Tomando regularmente o factor ”desempenho“ como preferencial torna-se necessária a utilização de um índice que não absorva de forma radical os recursos computacionais da máquina onde é executado.

Também as operações de pesquisa são efectuadas diariamente. Para os agentes de decisão, questões como ”O número de automóveis vendidos da na região norte ao longo do primeiro semestre do ano transacto?” ou “Qual o tipo de modelos de automóveis que mais vende no interior¿‘ são recorrentes. É, por isso, necessária a existência de índices que permitam a busca através largas quantidades de dados, devolvendo a resposta, no máximo, dentro de minutos.

5.1 Factores decisivos na construção de *datasets*

Em Sistemas de Suporte à Decisão, as operações de maior relevância são as de actualização e pesquisa. Apesar de existirem políticas de substituição de dados que levam à extinção de determinados dados e respectiva actualização do índice, este tipo de operações não toma um lugar de especial proeminência face às outras operações, que ocorrem frequentemente, e, consequentemente, não foi aprofundado o seu estudo ao longo da presente tese, nem foram executados testes referentes à supressão de nodos.

Para a realização dos testes utilizámos vários *datasets* com diferentes características para ser possível observar o desempenho dos índices em diversos tipos de ambientes. Os factores de variação para os diferentes *datasets* foram:

1. As dimensões.
2. A cardinalidade, isto é, o volume dos dados.
3. A homogeneidade dos dados no *dataset*.

Foi criado um elevado número de *datasets*, realizando possíveis combinações entre estes três factores, tentando-se simular o maior número de ambientes multidimensionais. O número de dimensões variou entre 3 e 20 sendo que, actualmente, a utilização de ambientes com um número tão escasso de dimensões como o 3, ou tão abastado como o 20, não são frequentes. Foram, por isso, considerados como limites inferior e superior para a realização de testes.

No que diz respeito à cardinalidade, foram testados ambientes entre os 10 000 (dez mil) e 10 000 000 (dez milhões) de registos foram testados. Também estes valores funcionaram como limites para os testes, existindo no entanto o reconhecimento de que 10 000 000 não é um valor incomum a grande parte dos ambientes multidimensionais. Contudo, devido à falta de recursos existentes, considerou-se este valor como o limite máximo possível para uma análise imparcial dos resultados obtidos.

Para além dos factores apresentados, foram testados também *datasets* com os dados uniformemente espalhados, e outros que simulavam ambientes densos. Para além disso, foram criados *datasets* com dados totalmente aleatórios, visando a simulação de ambientes mais imprevisíveis.

Para a geração de dados foram utilizadas ferramentas actualmente disponíveis no mercado, como o *SpatialDataGenerator* e foi desenvolvida uma aplicação para responder às necessidades específicas deste projecto, obtendo-se deste modo uma maior flexibilidade na realização de testes. As características que conduziram à escolha da linguagem são apresentadas de seguida.

5.2 Linguagem

Para a implementação dos índices que aqui se apresentam, a primeira escolha residiu sobre a linguagem na qual estes seriam desenvolvidos. Uma vez que o objectivo deste projecto é o estudo de índices em ambientes multidimensionais, a linguagem a utilizar teria de apresentar determinadas características, tais como:

- A portabilidade, um factor cada vez mais a ter em conta.
- A velocidade de processamento, a escolha teria de incidir numa linguagem de baixo nível, de modo a ser possível avaliar os rsultados obtidos mais claramente.
- A capacidade de manusear apontadores de memória, de modo a ser possível a maximização da flexibilidade do índice.

Através deste conjunto de características, tornou-se então possível a definição, optimização e realização de quaisquer operações de *tuning* com vista a um controlo sobre todos os detalhes. Constitui-se ainda como uma forma de tentar diminuir o número de acessos ao disco ou o espaço ocupado em memória mantendo em simultâneo um reduzido tempo de execução.

Actualmente, existe um compilador de código C para quase todas as plataformas, daí que se trate de uma linguagem com elevada portabilidade. Esta característica, associada a todas as outras que lhes são amplamente reconhecidas (como a possibilidade de manipulação de blocos de memória, o manuseamento directo de bits e a obtenção de altos índices de desempenho devido ao facto de ser uma linguagem de baixo nível) levaram a que a escolha da linguagem de construção dos índices incidisse sobre a linguagem C.

5.2.1 Ferramentas utilizadas

O *SpatialDataGenerator* [Theodoridis, 2003] permite a criação de *datasets* com os mais variados parâmetros, que vão desde a determinação do número de dimensões presentes, até ao tamanho do *dataset*, sendo ainda possível definir o tipo de distribuição presente nos dados (uniforme, Gauss ou Zipf) ou as predefinições em relação ao espaçamento dos dados sempre com um forte factor aleatório presente.

Esta ferramenta é, no entanto, limitada. Orientada para Sistemas de Informação Geográfica, a falta de flexibilidade no que respeita ao número de dimensões (que seriam 3) e de registos (perto dos cem mil), não permite a realização de testes aprofundados, uma vez que que não simula ambientes muito fidedignos em sistemas OLAP. Foi, portanto, necessária a criação de uma aplicação que gerasse grandes quantidades de dados com definições que permitissem a flexibilidade dos testes. Os parâmetros considerados mais importantes foram:

- A extensão do número de dimensões para o valor que se desejar.
- A uniformidade dos dados. É possível a simulação de mais do que um ambiente diferente.
- A cardinalidade de dados sem qualquer tipo de restrição, não contando com a que é imposta pelo *hardware*.

Como já foi referido, o número máximo de dimensões utilizadas para testes foi 20 e o de registos foi 10 000 000 (dez milhões). Enquanto que o primeiro foi definido por ter sido considerado como um número aceitável em ambientes OLAP e que já permitia um intervalo de dimensões significativo, o último foi imposto pelo *hardware*. Em grande parte das variantes do índice testado, um valor superior a dez milhões não poucas vezes esgotou a memória RAM da máquina, sendo necessário recorrer à SWAP e comprometer, assim, os resultados. Foi, portanto, definido este valor para obter uma maior imparcialidade e exactidão nos testes efectuados.

No que diz respeito à simulação de ambientes, foram implementadas 3 abordagens diferentes. Apesar de mais lineares do que as impostas pelo *SpatialDataGenerator*, representam, com alguma fidelidade, da recriação de ambientes mais comuns em OLAP:

- Ambientes uniformes com os dados espalhados de forma relativamente equitativa ao longo do espaço.

- Ambientes densos em que existem focos densos de dados.
- Ambientes aleatórios que não seguem qualquer ordem ou regra. Foram criados de modo a simular os ambientes mais imprevisíveis, com o intuito de se observar a adaptabilidade do índice.

O índice foi testado em todos estes ambientes, e em cada um realizaram-se testes com diferentes números de dimensões e cardinalidade. Estas operações repetiram-se para todas as variantes apresentadas do índice, com todas as abordagens implementadas ou só com algumas de modo a tornar-se perceptivel a influência de cada uma delas no seu desempenho.

5.3 Discussão dos resultados

De forma a avaliar as consequências apresentadas pelas alterações implementadas decidiu-se avaliar, separadamente, cada uma delas, comparando o índice alterado com o original em cada um dos ambientes simulados.Findo o conjunto de testes a cada abordagem, é realizada uma comparação entre a Point Region Quadtree original e a que foi construída para o presente trabalho, com todas as alterações existentes.

Os resultados obtidos são, ao longo de todo este capítulo, representados por gráficos. Optou-se por agrupar os resultados de acordo com determinados intervalos de dimensoes, por forma a não se fragmentar a análise dos valores. Embora não constituam resultados exactos, podem ser tidos como apontadores de tendências existentes.

5.3.1 Decomposição

Em ambientes uniformes, com baixa cardinalidade e reduzido número dimensões, observou-se que esta metodologia do índice não produz mudanças significativas no seu desempenho, nunca ultrapassando os 30% de vantagem ou -6% de desvantagem face ao índice original. Existiu, de forma geral, um aumento no tempo de inserção, que se deve ao maior número de cálculos e processamento efectuado no particionamento do índice. Esta desvantagem desapareceu, no entanto, assim que o número de dimensões e cardinalidade aumentou para valores mais elevados. Este fenómeno dá-se devido ao facto de o índice ir albergando um volume de dados cada vez maior, sendo que mais comparações serão realizadas, bem como operações de particionamento de

árvore. O aumento de gestão traduz-se numa melhor organização da árvore, acabando por economizar tempo de execução. Ainda assim, visto tratarem-se de registos inseridos de forma relativamente equitativa ao longo do espaço, as melhorias apresentadas não foram significativas (figura 5.1).

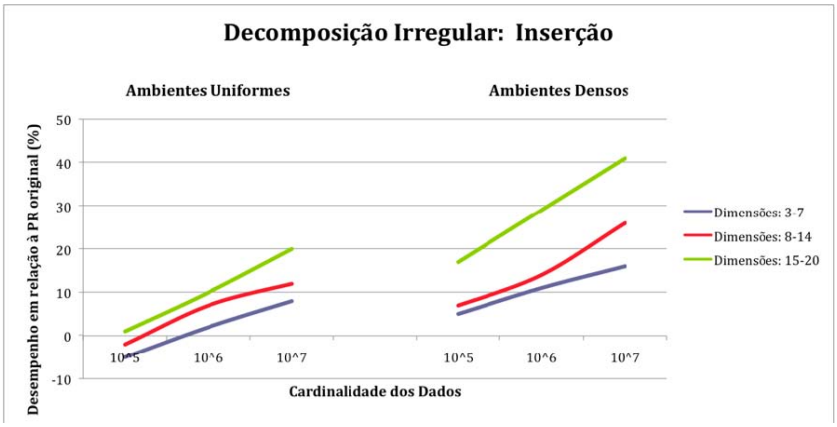

Figura 5.1: Decomposição: Inserção

Sabendo-se de antemão que a decomposição definida nas Point Region Quadtree original não é orientada a ambientes densos (dado tratar-se sempre de um particionamento efectuado à medida da menor distância entre dois pontos), a diferença de desempenho entre as duas árvores foi claramente notória, chegando a atingir os 40%. Para qualquer número de dimensões ou cardinalidade dos dados, a existência de uma decomposição baseada na média ponderada entre os pontos da região em questão mostrou-se muito mais eficiente do que o particionamento original. Tal diferença nos tempos de execução de procura e inserção foi aumentando à medida que o número de dimensões e volume de dados também aumentava. Apesar do excesso de gestão causada por esta alteração, os resultados demonstraram que se trata de uma mais valia dada a sua contribuição para uma melhor organização do índice.

Também em ambientes imprevisíveis esta alteração se apresentou como um benefício para a Point Region Quadtree. Ter definido, de forma estática, a distância de decomposição de uma árvore pode resultar em baixos índices de performance. Apesar de, por vezes, ter sido mais eficiente, na esmagadora maioria dos testes efectuados, tal não aconteceu levando a crer que a abordagem que particiona os dados consoante a sua disposição obtém melhores desempenhos.

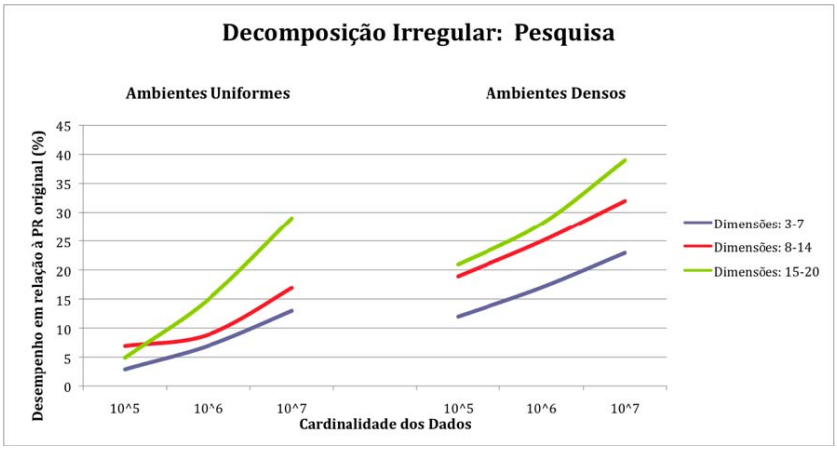

Figura 5.2: Decomposição: Pesquisa

5.3.2 Algoritmo de pesquisa

O algoritmo de procura introduzido através do sistema de cálculo de chaves representa uma significativa melhoria em todos os aspectos. A procura do filho pelo qual a pesquisa deverá continuar é efectuada, não sequencialmente ou através de uma árvore, mas sim por um simples cálculo, determinando assim rapidamente o quadrante a escolher para prosseguir a busca. O facto de reduzir o número de testes de k (em que k representa o número de dimensões) para 1, permite a obtenção de elevados índices de desempenho quando comparado com a sua congénere original.

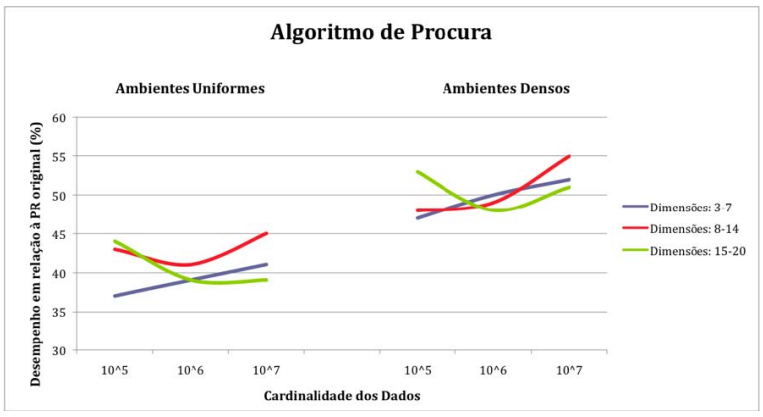

Figura 5.3: Algoritmo de pesquisa

Em todos os ambientes testados, independentemente do número de dimensões ou da cardinalidade, a diferença entre os dois métodos de pesquisa foi expressiva, verificando-se ainda uma economia no que diz respeito aos recursos utilizados.

5.3.3 Redução do TPL

Esta alteração representa uma abordagem que se torna especialmente útil quando se opera em ambientes densos. Por norma, neste tipo de ambientes, é comum a inserção de registos de forma a desbalancear o índice, fazendo com que este penda demasiado para determinada direcção. Apesar de este método não inverter totalmente essa tendência, ajuda particularmente no que diz respeito às operações de pesquisa. Com um custo adicional na inserção, consegue-se obter um melhor desempenho nas restantes operações, sendo significativa a diferença em relação ao índice original.

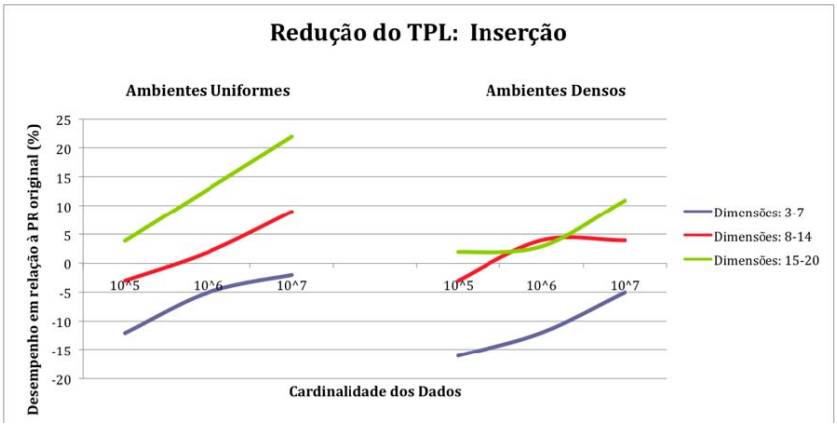

Figura 5.4: Redução do TPL: Inserção

No que diz respeito a ambientes uniformes, quando se trata de ambientes relativamente pequenos, a diferença de desempenho entre os índices não é significativa. No entanto, as vantagens são notórias quando aumenta a cardinalidade e são ainda maiores à medida que o número de dimensões existentes vai aumentando. Sendo um método que organiza, dentro de certos parâmetros, o índice, é mais susceptível aos efeitos da variação do número de dimensoes do que da cardinalidade

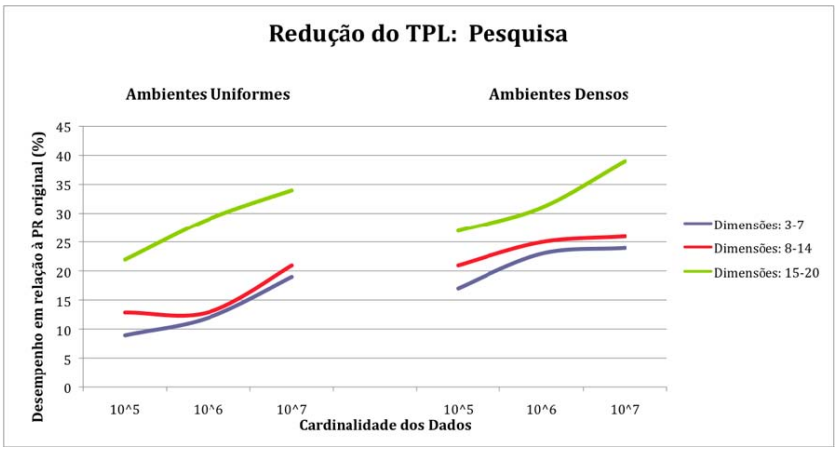

Figura 5.5: Redução do TPL: Pesquisa

5.3.4 K-D Tree

A introdução de pontos nos nodos intermédios dos registos conduz, consoante a estrutura de armazenamento, a uma maior rapidez no que diz respeito às operações mais comuns em ambientes OLAP. Ao longo da realização dos testes, os resultados demonstraram que as K-D Tree atingem índices de desempenho mais elevado do que as Árvores Binárias de Procura ou as Point Quadtree, duas estruturas que foram também testadas.

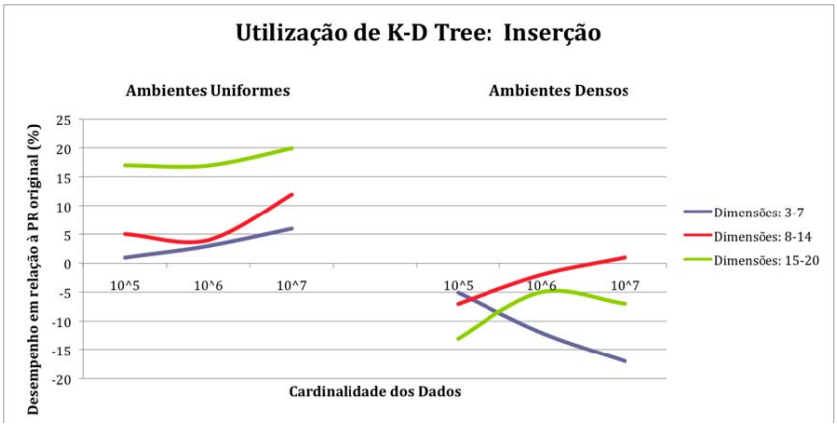

Figura 5.6: K-D Tree: Inserção

Em ambientes uniformes, comparando o desempenho de uma Point Re-

gion Quadtree original com a desenvolvida no presente estudo, ficou comprovado que a utilização das K-D Tree levam a um melhoramento na performance, sem prejudicar de forma significativa o espaço ocupado pelo índice.

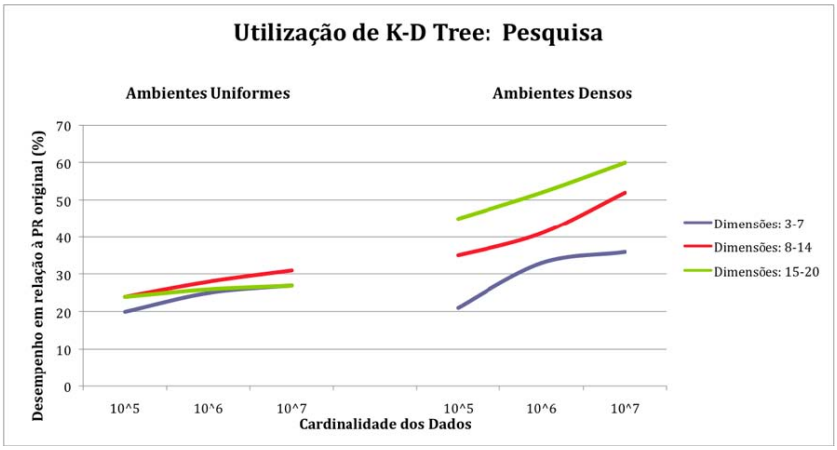

Figura 5.7: K-D Tree: Pesquisa

A melhoria no desempenho evidenciado pela utilização desta metodologia reduz-se, no entanto, quando aumenta a cardinalidade. Um maior número de registos conduz à ocorrência de mais particionamentos o que significa que as K-D Tree serão destruídas, sendo necessário construir novas árvores para o armazenamento de pontos. A perda de perfomance agrava-se quando se trata de ambientes densos, nos quais os dados ficam concentrados em pequenas áreas no espaço multidimensional. Nesse caso, o número de particionamentos aumenta de forma drástica, reduzindo levando a uma menor performance.No entanto, por tratar-se de uma estrutura leve, de fácil manuseamento de pontos, e em que não é necessário verificar sequer todas as coordenadas, representa ainda assim uma ligeira melhoria face à sua congénere original.

5.3.5 Ligação entre os nodos adjacentes

Tal como foi referido no capítulo anterior, a aplicação deste método a todas as dimensões existentes não é aconselhável. O *overhead* de gestão faz com que as operações de inserção se tornem drasticamente mais lentas do que as da Point Region Quadtree original. No entanto, caso se defina um pequeno (mas considerado relevante), grupo de dimensões para a utilização da ligação entre nodos adjacentes, as operações de pesquisa podem atingir altos índices

de desempenho.

Em ambientes densos a sua performance diminui, visto que sempre que existe um particionamento torna-se necessário o reajuste dos apontadores de todos os pontos pertencentes à região em questão. Nesse caso, e como este método por si só não introduz qualquer melhoria no que diz respeito à organização da árvore, a performance revela-se mais pobre do que a da Point Region Quadtree. No entanto, em conjunto com os métodos anteriores, que melhoram a estrutura, organização e funcionamento do índice, atenua-se a perda de desempenho observada neste caso.

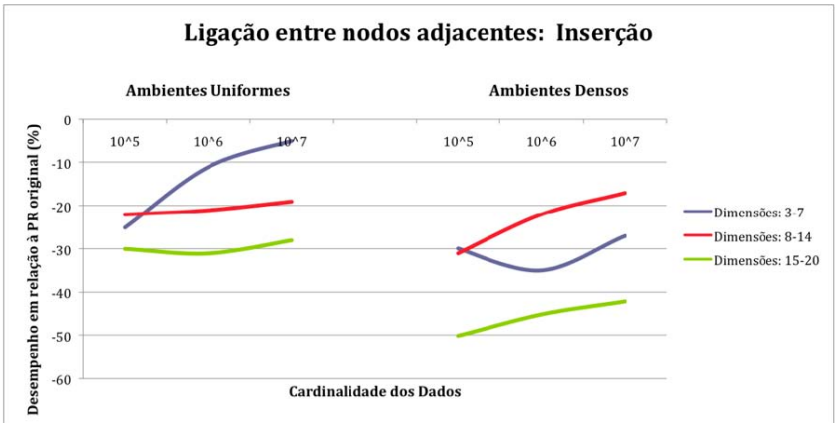

Figura 5.8: Ligação entre nodos: Inserção

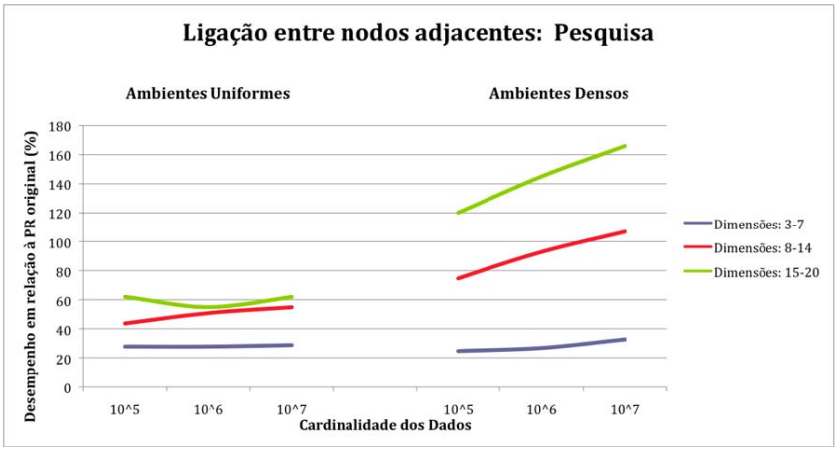

Figura 5.9: Ligação entre nodos: Pesquisa

Nas operações de pesquisa, as vantagens deste método são significativas. Deixa de ser necessário percorrer uma árvore de forma *top-down* ou *bottom-up*. Numa *range query*, o primeiro ponto pertencente à região a ser pesquisada funciona como catalisador da restante busca. Basta apenas necessário prosseguir por onde os apontadores conduzirem.

Com um problema ligeiramente semelhante ao da materialização de vistas, as dimensões que não foram escolhidas para a aplicação da ligação entre nodos adjacentes têm uma maior tempo de execução e a sua procura ocorre de forma tradicional.

5.4 Point Region Quadtree Original *vs* Extendida

Uma vez que foram já apresentados, separadamente, os resultados obtidos com todas as metodologias implementadas para a Point Region Quadtree e comparados com o índice original, resta agora a sua comparação com a árvore construída ao longo deste estudo, com todas as alterações presentes.

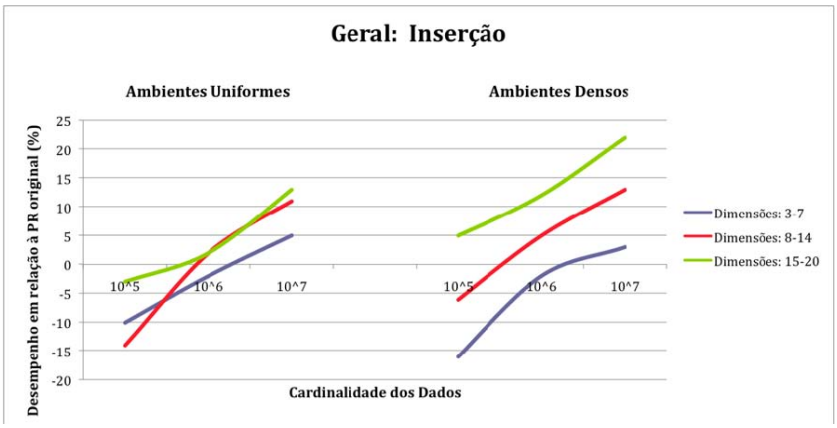

Figura 5.10: Geral: Inserção

Na inserção de registos, observou-se que o desempenho atingido pela Point Region Quadtree Extendida era menor do que o da sua congénere original chegando a atingir uma diferença de 15% entre as duas árvores. Este facto deve-se à complexidade do índice, sendo necessário uma vasta quantidade de operações para manter a sua estrutura organizada. No entanto, e à medida que aumenta a cardinalidade e a dimensionalidade, a tendência inverte-se conduzindo a menores tempos de execução pertos dos 15% no

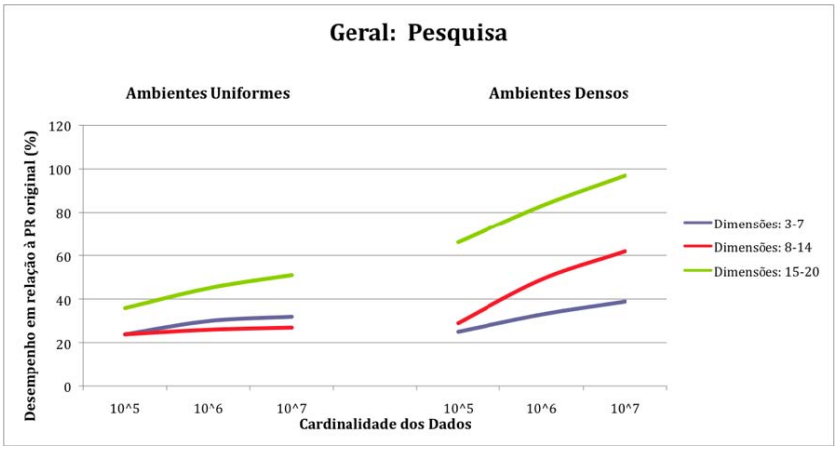

Figura 5.11: Geral: Pesquisa

índice alterado em relação ao original. Isto acontece devido à complementaridade das suas alterações: se, por um lado, demonstra-se um índice com uma maior organização poupando tempo e testes efectuados, por outro lado a sua estrutura permite uma procura mais eficaz.

Em ambientes densos, nos quais os particionamentos ocorrem frequentemente, as diferenças aumentam. O facto de, em cada particionamento se destruir as K-D Tree para, de seguida, criar novas realocando os pontos existentes, ou a reorganização de apontadores existentes, levam a um desempenho claramente inferior ao índice original na ordem de, no pior caso, atingir os 17%. Este apresenta uma clara perda de performance, naturalmente, quando se aumenta o volume de dados, sendo também aí a diferença maior do que acontece com ambientes uniformes, ultrapassando os 20%.

O proveito da estrutura desenvolvida apresenta-se em ambientes de maior complexidade. Logo, o aumento de dimensões demonstra, invariavelmente, a um melhor desempenho assinalado pela Point Region Quadtree Extendida devido à utilização do algoritmo de pesquisa apresentado, bem como a utilização de K-D Tree que obtêm melhores desempenhos em ambientes multidimensionais.

As inserções e actualizações realizam um conjunto de operações computacionalmente pesadas. A redução do *total path length*, o particionamento, a ligação entre os nodos adjacentes fazem com que seja necessária uma grande quantidade de tempo e recursos para o processamento deste tipo de

operações. No entanto, as pesquisas requerem menos recursos pois as suas operações são de menor complexidade. Consequentemente, a Point Region Quadtree Extendida apresentou melhores resultados do que a original em todos os ambientes testados, independentemente do número de dimensões ou cardinalidade, nunca descendo abaixo dos 20%. Naturalmente, a diferença foi maior com um elevado número de dimensões e com um grande volume de dados, bem como em ambientes densos. Nestes, a utilização da ligação entre nodos adjancentes foi particularmente útil, visto existirem *clusters* de dados. Nalguns casos, de maior cardinalidade e complexidade, a Point Region Quadtree Extendida apresentou tempos de execução cujo desempenho representava uma melhoria de 100% em relação à original.

6

Conclusões e trabalho futuro

Actualmente, vivemos num mundo de constante revolução tecnológica. As leis de Moore, lançadas há sensívelmente 40 anos, ainda hoje prevalecem. Com todo este desenvolvimento, a quantidade de informação actualmente disponível regista recordes, dia após dia e, a acompanhar tudo isto, também as bases de dados sofrem um crescimento exponencial. No entanto, devido ao crescente volume de dados, torna-se cada vez mais complexa a manutenção de registos, e os sistemas transaccionais vão perdendo a capacidade de executar, em tempo útil, as tarefas de análise que lhes são pedidas.

Os Sistemas de Suporte à Decisão estão, cada vez mais, presentes no quotidiano das grandes empresas. Estes apresentam, devido às suas características, uma grande aptidão para a análise de uma vasta quantidades de dados sem que com isso perturbem os sistemas operacionais ou levem demasiado tempo. Estas aplicações permitiram a abertura de uma nova porta no mundo empresarial ao possibilitar, de forma rápida e intuitiva, a análise de tendências de negócio e a manutenção de um registo histórico. A implementação de motores OLAP torna possível a apresentação dos dados ao agente de decisão de forma perceptível e intuitiva.

Os sistemas OLAP empregam estruturas multidimensionais de armazenamento e representação de dados sendo, conceptualmente, como cubos de diversas dimensões num universo euclidiano. De modo a optimizar os índices de desempenho das operações recorrentes nestes sistemas, realizaram-se várias abordagens distintas ao longo dos anos. O estudo apresentado nesta dissertação de mestrado incidiu na análise e construção de índices multidimensionais.

Ao longo da realização da presente tese, procurou-se compreender os conceitos associados aos ambientes multidimensionais, os índices a eles aplicados e os Sistemas de Suporte à Decisão. Foram estudados os motores OLAP, sistemas de *Data Warehousing* e as estruturas de indexação multidimensional. Nestas últimas, aprofundou-se o conhecimento referente às Quadtree, estruturas amplamente utilizadas actualmente nos mais diversos ramos das novas tecnologias, desde a construção de vídeojogos até aos Sistemas de Informação Geográfica. Foi estudada a sua estrutura, implicações, consequências e variantes, estruturas com uma visão revolucionária do espaço onde cada nodo contém 2^k filhos (em que k representa o número de dimensões).

Na escolha de uma estrutura foram ponderadas as características que esta deveria inicialmente conter, como a capacidade para lidar quer com pontos ou registos de dados, quer com regiões inteiras, bem como a indexação de dados com altos índices de desempenho. Dada a natureza das *queries* em ambientes OLAP, entendeu-se que as Point Region Quadtree seriam as mais adequadas e, consequentemente, foram eleitas para, a partir de si, ser construído um novo índice orientado para ambientes multidimensionais.

Ao longo da presente tese, estas estruturas, já de si complexas, foram sofrendo alterações visando a sua optimização para ambientes OLAP através da construção de um índice com um melhor desempenho no que respeita à inserção e procura de dados multidimensionais. De entre as várias alterações introduzidas, as consideradas mais relevantes foram:

- Utilização de um ponto fictício central, comum a todos os pontos de uma região.
- Decomposição da árvore irregular e adequada às necessidades de cada caso.
- Aplicação de métodos de balanceamento parcial do índice de modo a reduzir o comprimento da árvore.
- Algoritmo de procura que reduz o número de testes efectuados para o número de dimensões existentes.
- Utilização de estruturas de armazenamento de dados ao longo do índice, visando aliviar o custo computacional das suas operações

- Ligação através de apontadores entre nodos adjacentes nas respectivas dimensões.

Através da comparação directa com as suas congéneres originais no decurso dos testes efectuados, ficou claro que é possível, para estas quadtree, atingir níveis de desempenho elevados em ambientes multidimensionais, mantendo um índice organizado e funcional. Trata-se, no entanto, de uma estrutura complexa e o custo computacional adicional, sendo que se desaconselha a sua utilização a menos que a necessidade assim o dite.

Ao longo das diversas sessões de testes ficou claro que, para um estudo mais aprofundado deste índice, bem como de qualquer índice aplicado a ambientes OLAP, seria necessária uma máquina de testes com melhores especificações do que as daquela em que estes foram efectuados. Assim, poderiam ter sido criados ambientes com maior cardinalidade, aumentanto assim a exactidão dos resultados obtidos.

Devido ao tamanho dos *datasets* utilizados, verificou-se, com frequência, que o processamento das operações do índice ocupava toda a memória RAM e obrigava a recorrer à memória SWAP, comprometendo assim o resultado dos testes devido à diferença de desempenho entre estas. Naturalmente, todos os testes em que tal ocorreu foram terminados e os resultados por eles obtidos, descartados, não tendo sido tomados em consideração.

É relevante observar que nem todos os resultados obtidos ao longo dos testes foram ao encontro das conclusões aqui descritas. A isso deve-se o facto de as ferramentas utilizadas para a geração de *datasets* conterem uma alta componente de aleatoriedade. O tempo de execução das operações realizadas não depende apenas da cardinalidade dos dados ou do número de dimensões, mas também dos valores que estes contêm. Valores substancialmente diferentes podem ser obtidos com *datasets* com as mesmas características e, como tal, as conclusões alcançadas representam apenas as tendências observadas na maioria dos testes. Os resultados em causa não devem ser tomados como absolutos, mas antes como indicadores do que acontece em determinadas situações, simulando certos ambientes.

Existe ainda um longo caminho a percorrer até ser atingida uma solução realmente óptima para o aperfeiçoamento dos índices multidimensionais. Como trabalho futuro, recomenda-se o estudo uma alternativa que permita balancear totalmente o índice, e propõe-se ainda a utilização de um *threshold* de cada nodo dinâmico, em vez de estático. Seria também interessante aprofundar o

estudo da ligação entre os nodos adjacentes da árvore, de modo a possibilitar a sua execução em todas as dimensões sem que, para isso, consuma a actual quantidade de recursos. Por fim, considera-se ainda pertinente uma análise assimptótica do índice e a sua formalização.

Bibliografia

S. Kamal Abdali and David S. Wise. Experiments with quadtree representation of matrices. In *ISAAC '88: Proceedings of the International Symposium ISSAC'88 on Symbolic and Algebraic Computation*, pages 96–108, London, UK, 1989. Springer-Verlag. ISBN 3-540-51084-2.

E. Baralis, S. Paraboschi, and E. Teniente. Materialized View Selection in Multidimensional Database. 2005.

Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 156–165, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7.

Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Inf.*, 1:173–189, 1972.

J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Jon Louis Bentley, Donald F. Stanat, and E. Hollings Williams Jr. The complexity of finding fixed-radius near neighbors. *Inf. Process. Lett.*, 6(6):209–212, 1977.

Codd S.B. Codd E.F. and Salley C.T. Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. 1993.

Barry Devlin and Paul Murphy. An architecture for a business and information systems. 1988.

T. Johnson e D. Shasha. Hierarchically split cube forests for decision support; description and tuned design. 1996.

RA Finkel and JL Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974a.

Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974b.

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

M. N. Gahegan. An efficient use of quadtrees in a geographical information system. *Int. J. of Geographical Information Systems*, 3(3):215–232, 1989.

Irene Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, 1982. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/358728.358741.

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.

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.

Ugur Güdükbay, Ieee Computer Society, and Türker Yilmaz. Stereoscopic view-dependent visualization of terrain height fields. *IEEE Transactions on Visualization and Computer Graphics*, 8, 2002.

Ralf Hartmut Güting, Praktische Informatik Iv, and Fernuniversität Hagen. An introduction to spatial database systems. *VLDB Journal*, 3:357–399, 1994.

C. V. Hall and D. S. Wise. Compiling strictness into streams. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 132–143, New York, NY, USA, 1987. ACM. ISBN 0-89791-215-2. doi: http://doi.acm.org/10.1145/41625.41637.

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.

Gregory Michael Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Princeton, NJ, USA, 1978.

CL Jackins and SL Tanimoto. Quad-trees, oct-trees, and K-trees: a generalized approach to recursive decomposition of euclidean space. *IEEE transactions on pattern analysis and machine intelligence*, 5(5):533–539, 1983.

Ravi Kanth and V Kothuri. Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In *Proceedings of ACM SIGMOD Conference*, pages 546–557, 2002.

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.

A. Klinger. Patterns and search statistics. *Optimizing Methods in Statistics*, pages 303–337, 1971.

Donald E. Knuth. *Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition)*. Addison-Wesley Professional, July 1997.

D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *j-INT-J-COMPUT-INF-SCI*, 9(3):219–242, June 1980.

D. T. Lee and C. K. Wong. Worsleew77t-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Inf.*, 9:23–29, 1977.

V. Markl, F. Ramsak, and R. Bayer. Improving OLAP performance by multidimensional hierarchical clustering. *Database Engineering and Applications, 1999. IDEAS'99. International Symposium Proceedings*, pages 165–177, 1999.

D. Meagher. Geometric modeling using octree encoding. 1981.

Doug Moore. The cost of balancing generalized quadtrees. In *In Proceedings of the 3rd Symposium on Solid Modeling and Applications*, pages 305–312, 1995.

Mark H. Overmars and Jan van Leeuwen. Dynamic multi-dimensional data structures based on quad- and *k - d* trees. *Acta Inf.*, 17:267–285, 1982.

D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. *Proc. of SSTD*, 280, 2001a.

Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 443–459, London, UK, 2001b. Springer-Verlag. ISBN 3-540-42301-X.

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.

H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley Reading, Mass, 1990.

Sunita Sarawagi. Indexing olap data. *Data Engineering Bulletin*, 20:36–43, 1997.

B. D. Saunders, H. R. Lee, and S. K. Abdali. A parallel implementation of the cylindrical algebraic decomposition algorithm. In *ISSAC '89: Proceedings of the ACM-SIGSAM 1989 international symposium on Symbolic and algebraic computation*, pages 298–307, New York, NY, USA, 1989. ACM. ISBN 0-89791-325-6. doi: http://doi.acm.org/10.1145/74540.74576.

Amit Shukla, Prasad Deshpande, and Jeffrey F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 488–499, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-566-5.

Michael Spann and Roland Wilson. A quad-tree approach to image segmentation which combines statistical and spatial information. *Pattern Recognition*, 18(3-4):257–269, 1985.

Yannis Theodoridis. The r-tree-portal, 2003. URL `http://www.rtreeportal.org`.