Departamento de Ciências e Tecnologias de Informação

OPTIMIZAÇÃO DE ESTRUTURAS MULTIDIMENSIONAIS DE DADOS EM AMBIENTES OLAP

Jorge Miguel Dias Afonso

Tese submetida como requisito parcial para obtenção do grau de
Mestre em Ciências e Tecnologias de Informação
Especialidade em Sistemas Integrados de Apoio à Decisão

Orientador:
Doutorado, Orlando Belo, Prof. Associado,
Departamento de Informática, Universidade do Minho

Co-orientador:
Mestre, José Farinha, Assistente,
Departamento de Ciências e Tecnologias de Informação, ISCTE

Outubro, 2009

Agradecimentos

Muitos foram aqueles que colocaram um empenho e dedicação especial e deram a sua contribuição e apoio para a realização deste trabalho. Na impossibilidade de enumerar todos, parece justo particularizar aqueles que merecem uma menção especial, pelo seu rigoroso acompanhamento. Mas a todos remeto, desde já, o meu sentido muito obrigado. Ao Prof. Dr. Orlando Belo, orientador cientifico deste trabalho, quero expressar a minha gratidão profunda, não só pelo seu incondicional apoio e incentivo mas também pela confiança e amizade em mim depositadas. Todo o seu conhecimento e metodologia foram pilares fundamentais para a realização deste trabalho. As suas reflexões, concelhos, críticas e vasta experiencia académica e profissional foram inestimáveis e muito contribuíram para que esta dissertação tomasse a direcção certa. A ele, muito devo. À Prof. Dra. Maria José Trigueiros, coordenadora cientifica do MSIAD, pela sua paciência e total dedicação que entregou ao ensinamento das disciplinas de sistemas de suporte à decisão. Muito agradeço as suas palavras de confiança, incentivo e, sobretudo, a sua disponibilidade inesgotável de acompanhamento e aconselhamento. Ao Mestre José Farinha, co-orientador deste trabalho, pela motivação e disponibilidade em colaborar no empreendimento de revisão prévia dos textos aqui publicados e demais orientações cientificas.

A todos os professores do departamento de ciências e tecnologias de informação do ISCTE, especialmente aqueles que foram envolvidos directamente no curso de mestrado, o meu sincero agradecimento pela experiência e saber transmitidos. A todos os colegas do MSIAD, em particular à Joana Eusébio, ao Pedro Malcata, ao Nuno Matamouros e ao Pedro Cardoso, o

meu muito obrigado pelas horas de trabalho conjunto e pelo incentivo mútuo, fundamental para concluir com aproveitamento as disciplinas e os trabalhos de grupo realizados. Aos membros da família que acompanharam, directa ou indirectamente, a realização deste trabalho, envio um forte abraço de agradecimento. Finalmente, mas não menos importante, expresso com todo o carinho, a minha gratidão à minha família mais chegada: aos meus pais e ao meu irmão. Especialmente à minha mãe, Maria de Jesus, pelo seu importante suporte psicológico e pela força transmitida nos momentos de maior desânimo. A ela, muito devo. Também pelos momentos que não lhes pude dedicar, pela compreensão de outras prioridades que me foram impostas no decorrer desta e de outras jornadas e por sempre desculparem as minhas maiores ausências. A todos, família, amigos e colegas, renovo os meus profundos agradecimentos pela motivação, paciência e compreensão.

Resumo

OPTIMIZAÇÃO DE ESTRUTURAS MULTIDIMENSIONAIS DE DADOS EM AMBIENTES OLAP

A evolução dos sistemas de *Data Warehousing* em dimensão e utilização impôs uma agitação contínua sobre os sistemas de processamento analítico. A materialização de estruturas multidimensionais de dados é, desde à muito, vista como uma forma de optimizar o tempo de resposta às interrogações de natureza agregada. Para além da temporalidade, é necessário considerar uma outra perspectiva: o espaço necessário para armazenar todas as agregações calculadas. Na prática, o problema da selecção de estruturas multidimensionais de dados traduz-se principalmente na escolha das vistas que mais evidenciam a diminuição dos custos de manutenção e consulta, tendo em consideração os subcubos (ou cubóides) mais vantajosos para responder às interrogações dos utilizadores. A proporção da relação tempo/espaço é reconhecida como um problema *NP-hard*. De facto, muitos sistemas de suporte à decisão efectuam o pré-processamento das estruturas multidimensionais de dados de modo a optimizarem o tempo de resposta às consultas efectuadas pelos agentes de decisão. Contudo, a materialização integral dos subcubos é praticamente inexequível quando confrontada com a elevada dimensionalidade e cardinalidade, intrínsecas à complexidade dos sistemas multidimensionais modernos, para além das suas exigências conhecidas ao nível do tempo e do espaço. A materialização parcial representa, por outro lado, um interessante *trade-off* entre o espaço de armazenamento e o tempo de pré-processamento de vistas. Neste domínio são

analisadas algumas técnicas para optimizar a selecção de estruturas multidimensionais de dados, denominadas "icebergue", como resposta à reformulação do problema de materialização integral de vistas. Na sua essência, estes algoritmos calculam apenas as células agregadas das estruturas de dados que satisfazem uma determinada condição, com o objectivo de identificar os valores que farão sentido considerar nas análises de suporte à decisão, qualificando apenas as agregações com mais significado analítico e, portanto, as que devem ser materializadas. Em resultado da investigação efectuada, são analisados diferentes algoritmos de selecção de estruturas multidimensionais de dados, dando especial ênfase às lógicas de selecção icebergue. Para além da caracterização multidimensional (em tempo e espaço) das soluções propostas, são identificadas as suas vantagens mais predominantes e quais os pontos mais delicados que devem merecer especial atenção.

Palavras-chave: Sistemas de Suporte à Decisão; *Business Intelligence*; *Data Warehousing*; *Online Analytical Processing*; Processamento de Estruturas Multidimensionais de Dados.

Classificação ACM: H.2.3. Database Management. Languages. Data manipulation language;
H.2.4. Database Management. Systems. Query processing;
H.4.2. Information Systems Applications. Types of systems.

Abstract

OPTIMIZING MULTIDIMENSIONAL DATA STRUCTURES IN OLAP ENVIRONMENTS

The Data Warehouse evolution in size and use imposed a continuous frenzy on the OLAP systems. The materialization of multidimensional data structures is, from early times, a way of improving the answering time of those systems to aggregated queries. In addition to time, it's necessary to consider another perspective: the space required to store all the calculated aggregates. In practice, the multidimensional data structures selection problem is mostly related with views selection that mainly reveals a decrease of interrogation and maintenance costs, according the variety of cuboids more useful to answer any inquires made by users. The proportion time/space is recognized as an NP-Hard problem. In fact, many decision support systems carry out multidimensional data structures pre-computing in order to optimize the answering time of the queries made by the decision makers. However, the computation of all the cuboids in a multidimensional data structure is nearly infeasible when confronted with high dimensionality and cardinality, inherit to the complexity of modern Data Warehouse and OLAP systems (in addition to its recognized requirements of time and space). On another hand, partial materialization offers an interesting trade-off between storage space and response time for materialized views pre-computation. In this work, we discuss some partial materialization techniques for improving computation and selection of the most valuable cuboids of a multidimensional data structure, knows as "iceberg" algorithms, in response to

the full materialization views selection problem. In essence, these algorithms calculate only a fraction of the cells in a multidimensional data structure whose aggregate value is above some minimum support threshold, in order to identify the aggregates that make sense reflect in a decision support analysis (this scenario allows to describe only the aggregates with more analytical meaning and, therefore, those that would be materialized). As a result of this research, different algorithms are analyzed for the views selection problem, principally the "iceberg" selecting logics. As well as the multidimensional characterization (in time and space) of the proposed solutions, this work identifies their most revealing advantages and what are the mainly fragile points that deserve special attention.

Keywords: Decision Support Systems; Business Intelligence; Data Warehousing; Online Analytical Processing; Multidimensional Data Structures Processing.

ACM Classification: H.2.3. Database Management. Languages. Data manipulation language;
H.2.4. Database Management. Systems. Query processing;
H.4.2. Information Systems Applications. Types of systems.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

1.1. Enquadramento do Problema

Vive-se, actualmente, numa sociedade controlada pelo poder da informação. Face às constantes oscilações no mercado, ao impacto da evolução tecnológica e à emergência de novos processos e modelos de gestão, as instituições modernas são obrigadas a reflectir, constantemente, as suas estratégias de negócio. A sua sobrevivência procede da forma como estas interpretam, tratam, avaliam e usam a informação organizacional nos processos de tomada de decisão. De facto, a geração de conhecimento através da estruturação da informação permite às organizações conhecerem-se a si mesmas e posicionarem-se no mercado, facilitando o processo de tomada de decisão a todos os níveis organizacionais. A capacidade de saberem "onde estão" e as decisões que devem tomar para compreenderem para onde "querem ir" são factores chave para resistir às mudanças imprevisíveis da economia e garantir a sua competitividade.

Os avanços tecnológicos permitem hoje às organizações conhecer a riqueza dos dados que são gerados diariamente pela monitorização das suas actividades e processos de negócio. De facto, a utilização desses eventos permite à empresa tomar decisões sobre o presente e projectar o futuro. Contudo, a heterogeneidade dos sistemas de informação organizacionais dificulta a contextualização dos dados, na medida em que as disparidades dos mesmos, em

teor e significado, variam de acordo com as necessidades operacionais a que se destinam. Na tentativa de combater esse fenómeno, assiste-se a um forte investimento, por parte das organizações, no desenvolvimento de repositórios centrais e autónomos - usualmente conhecidos por *Data Warehouses* [Inmon, 1996] - que integram e armazenam a informação considerada útil, proveniente dos sistemas operacionais. A preparação destes ambientes é um processo complexo que abrange a concretização de etapas que envolvem a recolha, limpeza, purificação e uniformização dos dados, concedendo-lhes um significado único e consistente.

A flexibilidade e o desempenho no acesso central a toda a informação cooperativa, fazem dos sistemas de *Data Warehousing* a escolha preferida dos técnicos e gestores para realizarem análises de dados complexas que auxiliem nos processos de tomada de decisão, em todos os níveis organizacionais. A disponibilização da informação em tempo útil que contextualiza toda a envolvência do negócio torna-se condição de sobrevivência, permitindo que as empresas tomem as decisões certas, no momento adequado. Estes sistemas são a base de todo o processamento analítico, cuja qualidade depende de factores como a organização, extensibilidade e coerência dos dados disponíveis e do acesso imediato à informação pretendida. De facto, a percepção real dos dados motivaram os decisores a visualizarem o negócio em diversas perspectivas de análise segundo várias caracterizações, numa estrutura multidimensional. Existem, no entanto, alguns constrangimentos na criação e manutenção deste tipo de estruturas, na medida em que o cálculo de muitas agregações produzidas sobre um elevado volume de dados históricos consome tempo de processamento e espaço de armazenamento dos dados.

A utilização exaustiva destes sistemas vem dar origem a novas preocupações que causam impacto directo na qualidade das decisões tomadas. Torna-se necessário investir em mecanismos optimizados de cálculo analítico de maneira a melhorar o desempenho e contribuir para a minimização do tempo de resposta dos pedidos, aumentando a satisfação e produtividade dos decisores. Este trabalho é motivado pela análise dessas preocupações e envolve o caracterização detalhada de métodos que permitem optimizar as estruturas multidimensionais de dados, focando-se primariamente na evolução do conceito de vistas

materializadas e suas dependências, na análise do processamento, total ou parcial, das agregações e no desenvolvimento de algoritmos que irão permitir melhorar o tempo de resposta às *querys* que são efectuadas.

1.2. Definição e Descrição do Problema

Substanciando o enquadramento efectuado na secção anterior, é possível depreender que as dificuldades decorrentes do problema da selecção e disponibilização atempada da informação analítica são motivadas, sobretudo, pelo tempo de cálculo das agregações produzidas e consequente necessidade de espaço de armazenamento, efeito da evolução e crescimento dos sistemas de processamento analítico. Por esses motivos, este trabalho de dissertação tenciona disponibilizar um estudo comparativo de diferentes algoritmos de selecção de estruturas multidimensionais de dados, recorrendo a técnicas de *iceberg cubing*, de forma a responder a seguinte questão:

> *De que forma é que os algoritmos de iceberg cubing podem contribuir para reduzir o tempo de processamento e o espaço de armazenamento de um cubo de dados?*

De facto, assiste-se a um período em que o crescimento dos sistemas de *Data Warehousing* dão lugar a novos desafios. Cada vez mais, gestores e analistas recorrem diariamente a sistemas que permitam processar, manipular e analisar um grande volume de dados sob múltiplas perspectivas de modo a facilitar o processo de tomada de decisões. O processo analítico dos dados assenta sobre uma estrutura multidimensional formada por relações entre factos e dimensões de negócio de modo a facilitar o cruzamento e investigação dos dados gerados. De facto, a evolução da dimensão e utilização a este nível é de tal forma exaustiva que as organizações estão a tornar-se inquietas face à capacidade insuficiente dos recursos para se efectuar, eficaz e eficientemente, o processamento analítico de todos os pedidos. A materialização de estruturas multidimensionais de dados é, desde à muito, vista como uma forma de optimizar o tempo de resposta a solicitações de natureza agregada. Este tipo de estruturas fundamenta qualquer ambiente de processamento analítico, sendo uma mais-valia

para os decisores que têm a possibilidade de visualizar os factos de negócio em diferentes perspectivas, passíveis de serem manipulados de acordo com as suas necessidades. Os sistemas que permitem este tipo de análises designam-se por sistemas OLAP (*On-Line Analytical Processing*) [Codd. et al., 1993] ou sistemas de processamento analítico cuja base resulta da utilização de dados organizados de forma multidimensional, representados no formato figurado de cubo de dados [Gray et al., 1996], [Chaudury & Dayall, 1997].

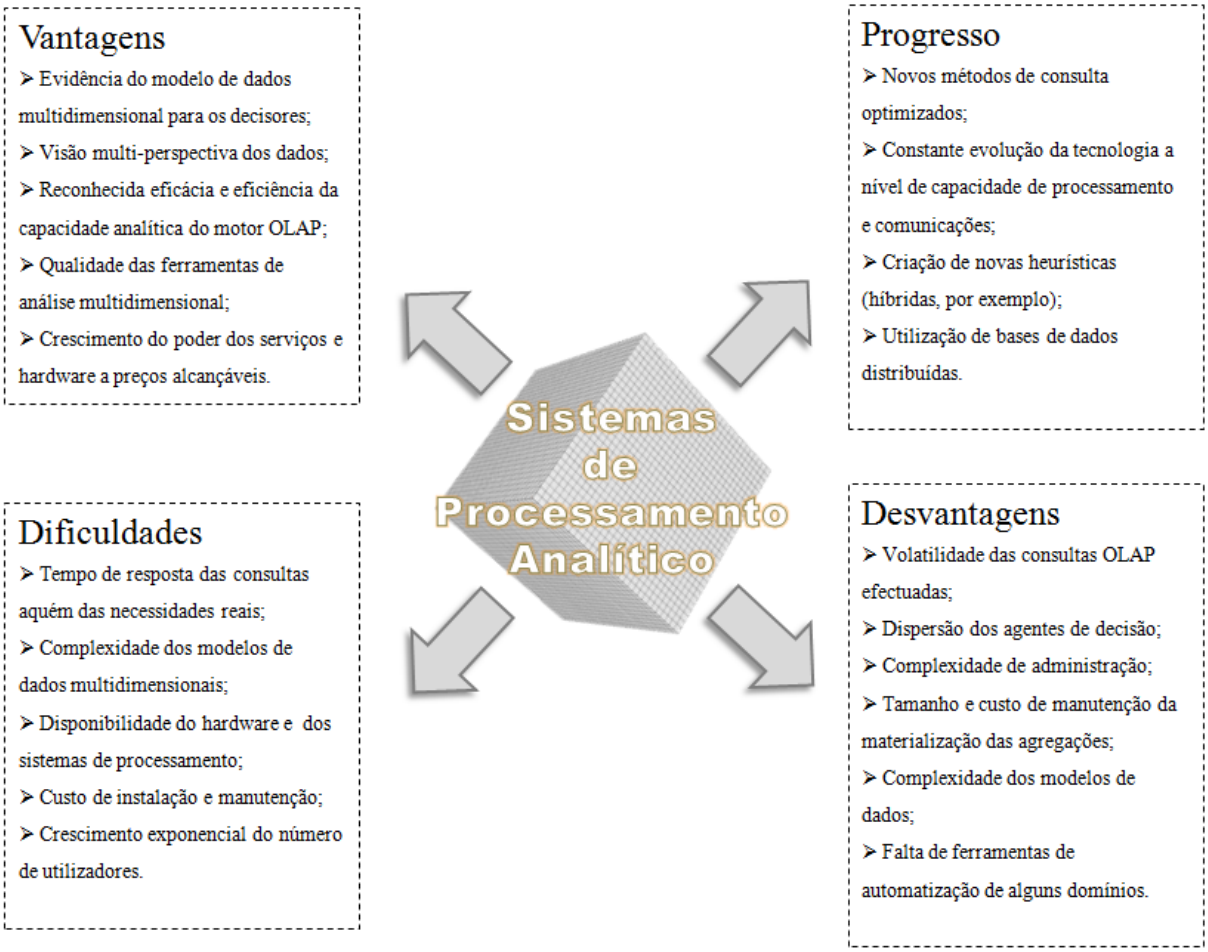


Figura 1. A predominância dos sistemas OLAP

A imagem multi-perspectiva suportada pelo cubo de dados permite dispor a informação em função das dimensões de análise determinadas pelos decisores e armazenar, tanto os dados originais como os resultados do processamento analítico, efectuados sobre um conjunto de métricas seleccionadas. As perspectivas de análise estão normalmente estruturadas em

hierarquias que revertem a granularidade e especificidade das análises. Independentemente do alcance da navegação multidimensional da própria análise da informação, é essencial garantir que os pedidos sejam processados num curto espaço de tempo. Aliás, a qualidade das decisões e a produtividade dos decisores pendem maioritariamente dos tempos de resposta às interrogações que estes lançam no sistema analítico.

Descendo a um nível mais detalhado, contendo informação menos agregada, existem duas soluções típicas que devem ser enquadradas de acordo com as necessidades de agregação, para garantir o melhor desempenho possível: os cálculos *on-the-fly* são realizados sobre os dados base com o objectivo de agrupá-los em função da granularidade conveniente para cada uma das perspectivas de análise. Este método é muito eficiente em ambientes de pequena dimensão mas prova-se inadequado na análise de grandes quantidades de informação devido ao tempo excessivo de processamento e a quantidade de recursos analíticos que consome. Por outro lado, é possível optar por uma solução que calcule previamente as agregações e armazene essa informação no sistema analítico, cuja consulta resulta na disponibilização imediata dos valores processados. Contudo, é natural que um método que apresente tal robustez tenha alguns custos, dos quais se destacam o tempo de cálculo, o pré-processamento das agregações (na forma de vistas materializadas) e o espaço necessário para armazenar cada valor ponderado.

De facto, o tempo de manutenção do cubo materializado é extenso principalmente devido à necessidade de assegurar a consistência de todas as agregações face aos dados originais, sempre que estes são sujeitos a alterações motivadas por novos eventos ou mudanças nas regras de negócio. O número de agregações a reprocessar pode ser enorme na medida em que determinam todas as maneiras possíveis em que os dados originais podem ser ligados hierarquicamente. Por exemplo, ao avaliar-se a dimensão ‘localização’, agrupada por três níveis hierárquicos (constituídos por 2 países, 8 distritos e 50 cidades), verifica-se que esta será composta por 60 membros. Em paralelo, se adicionar-se a dimensão ‘produto’, agrupada por quatro níveis hierárquicos (constituídos por 150 produto, 20 categoria, 5 famílias e 2 indústrias), o nível de membros subirá para os 177. A conjugação de apenas duas dimensões

com os diversos níveis hierárquicos indica 10620[1] agregações possíveis. Isto significa que à medida que os dados base aumentam ao longo do tempo, o número de agregações tende a evoluir exponencialmente, podendo ascender às dezenas de milhões ou mais.

A combinação das métricas agregadas com os dados originais numa perspectiva multidimensional pode conter, potencialmente, todas as respostas para cada pedido efectuado. No entanto, quando se começa a aumentar a complexidade no número de dimensões e hierarquias, o número de agregações fica enorme, tornando-se inconciliável, a sua materialização total. Existe, portanto, necessidade de procurar soluções que optimizem a forma de como um cubo multidimensional é consultado e, sobretudo, utilizado. De facto, é notório que a resposta apropriada a um conjunto de agregações pode significar a solução para muitas outras ou simplesmente, existirem agregações que nunca são usadas, tornando-se inúteis. Note-se que cada célula agregada numa estrutura multidimensional corresponde a um conjunto exclusivo de valores para as diferentes perspectivas, que representam, efectivamente, as métricas de interesse. Ao definir-se um perfil de utilização para manipular as estruturas multidimensionais está-se a seleccionar quais a agregações mais susceptíveis de serem pesquisadas, correspondendo àquelas que são as mais compassivas, tendo em conta um ou mais constrangimentos (Ex. o tempo e espaço para criação e manutenção de cubos de dados).

Face aos constrangimentos das interrogações efectuadas e do espaço necessário para armazenar todas as agregações calculadas, o problema da selecção de dados de um cubo multidimensional num ambiente OLAP traduz-se principalmente na escolha das vistas, subcubos (ou cubóides), de maneira a evidenciar a diminuição dos custos de manutenção e consulta. A proporção da relação tempo/espaço é reconhecida como um problema *NP-hard* [Harinarayan et al., 1996]. De facto, uma pesquisa acertada pode significar ganhos elevados, sendo possível obter, apenas com a utilização de uma percentagem reduzida do espaço de materialização do subcubo, valores semelhantes ao tempo de resposta das interrogações, em comparação com a materialização integral do cubo. Este ganho substancial expõe a relevância do problema, que tem sido, ao longo do tempo, a base de investigação de toda a comunidade

[1] Resultante do produto dos membros das dimensões "localização" e "produto" [60 × 177 = 10620]

científica, para o qual tem vindo a propor um conjunto de diferentes soluções. A selecção de vistas permite aliviar, efectivamente, o problema e proporcionar custos controlados, maior disponibilidade e eliminação de situações mais críticas. No entanto, devem ser enquadradas outras dimensões que têm impactos idênticos - ou ainda maiores - no custo de processamento destas estruturas. Não importa apenas seleccionar os subcubos mais adequados, considerando um determinado perfil de utilização, mas também saber como materializá-los nos nós mais vantajosos, visto que a analogia de vistas materializadas é geradora de novas dependências cujo significado é apreendido pelo *lattice*[2] de correlações ou dependências. Paralelamente, deve ser considerada a dimensão tempo, na medida em que se torna necessário aperfeiçoar e evoluir as soluções consideradas tradicionais e adaptá-las às necessidades presentes e futuras.

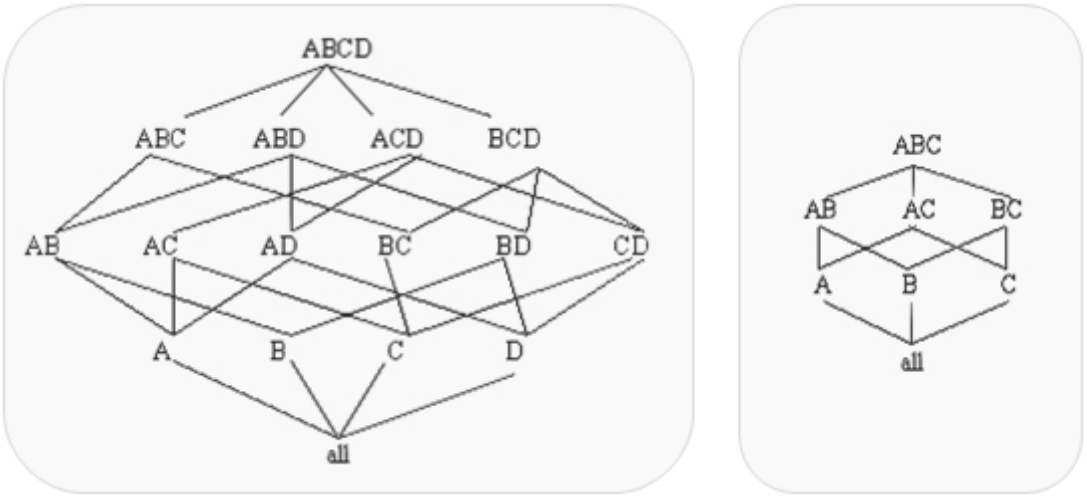


Figura 2. Lattice multidimensional de correlações (cubo com 3 e 4 dimensões)

Na prática, a materialização total ou parcial de um cubo multidimensional é sinónimo de desempenho num sistema de processamento analítico. Por outro lado, se não for possível

[2] Na matemática, um ***lattice*** é um conjunto de pontos ordenado parcialmente no qual dois elementos têm um único supremo e um único ínfimo. Os *lattices* também podem ser caracterizados como estruturas algébricas que respondem a certas identidades axiomáticas. Uma vez que as duas definições são equivalentes, a teoria do *lattice* baseia-se tanto na teoria de ordenação como na álgebra universal. No processamento analítico, o *lattice* de correlações de um cubo permite contextualizar todas as dependências características no que se refere às hierarquias dimensionais e agregações inter-dimensionais. Essencialmente, representa um esquema conceptual de todas as agregações possíveis e respectivas dependências.

concretizar todo o cubo, o perfil de utilização pode ser decisivo na sua optimização. No entanto, existem alguns senãos: o espaço de armazenamento e o tempo de cálculo, materialização e manutenção, na medida em que as alterações das relações base devem ser reflectidas em todas e quaisquer vistas materializadas. Além disso, à que ter em conta a selecção dos subcubos mais adequados. Nessa perspectiva, um dos objectivos primários deste trabalho passa por divulgar algoritmos de selecção optimizados, face aos métodos de análise convencionais de interrogação e selecção de subcubos, focando em particular, o desenvolvimento do conceito de *iceberg cubes* [Beyer & Ramakrishnan, 1999].

Um *iceberg cube* contêm apenas as células agregadas do cubo de dados que satisfazem uma determinada condição, conhecidas como a "ponta do icebergue". Esta estrutura tem como objectivo identificar e calcular apenas os valores que serão os mais adequados e que farão sentido para as análises de suporte à decisão, ou seja, a condição agregada qualifica quais os valores do cubo que têm mais significado e, portanto, os que devem ser armazenados. Essencialmente, as interrogações efectuadas num *iceberg cube* partem da análise de um atributo (ou conjunto de atributos) de forma a obter-se as agregações que estão acima de um determinado limiar, face a determinadas condições (*having count, sum, group by, count, etc*). Estas interrogações são conhecidas como *iceberg querys* [Fang et al., 1997] porque conseguem lidar apenas com os valores que estão acima de um determinado limiar (a "ponta do icebergue") face ao volume total dos dados fonte (o "icebergue"), utilizando muito pouca memória e, significativamente, menos operações sobre os dados.

```
SELECT R.ai1, R.ai2, . . . . , R.aik , count (rest)
FROM R
GROUP BY ai1, ai2,..., aik,
HAVING count (rest) >= n x f
```

Figura 3. Exemplo de uma query iceberg

Existem diferentes algoritmos que permitem calcular *iceberg cubes* de forma mais eficiente e que apresentam soluções distintas para evitar o processamento integral de todas as células possíveis no cubo, optimizando o tempo de cálculo, ignorando aquelas que não apresentam

uma utilidade analítica aceitável. Estudos mencionados na literatura evidenciam duas grandes aproximações no processamento de subcubos: as abordagens *Top-down* e *Bottom-up*.

A representação *Top-down* divulgada pelo algoritmo *MultiWay* [Zhao et al., 1997] utiliza um vector comprimido de elementos dispersos, maioritariamente com o mesmo valor, de maneira a suportar a base do cubo e efectuar o seu processamento, percorrendo de cima para baixo, o *lattice* multidimensional distribuído que constitui todas as combinações dos atributos dos dados fonte e a relação entre elas. Este algoritmo é eficaz quando o produto das cardinalidades das dimensões é moderado. Caso contrário, este método torna-se inviável porque os vectores e os resultados intermédios gerados a partir desses, são muito grandes para serem armazenados em memória (a base do cubo é carregada na totalidade em memória). Note-se que esta perspectiva *Top-down* não pode tirar partido da filtragem (*pruning*) do algoritmo APRIORI [Agrawal & Srikant, 1994] que evidencia a utilização de combinações candidatas para evitar que sejam calculadas todas as combinações possíveis de agregações. Isto porque, o *lattice* distribuído não cumpre a propriedade anti-monótona[3] e a condição da *query iceberg* só será real após o processamento integral do cubo. Para que uma combinação de agregações consiga satisfazer a condição *iceberg*, todos os subgrupos da associação também o devem fazer. As associações candidatas são observadas somente através da combinação das agregações mais frequentes e que já são conhecidas. Todas as outras combinações possíveis são, automaticamente, eliminadas (já que nem todos os seus subgrupos satisfazem a condição da *query* icebergue).

Por outro lado, a perspectiva *Bottom-up* (BUC) [Beyer & Ramakrishnan, 1999] ordena rapidamente a base de dados em função das necessidades, de maneira a permitir particionar e contabilizar convenientemente as combinações possíveis, sem por em causa a memória dedicada para o processamento. O algoritmo particiona a base de dados baseando-se nos

[3] A **propriedade anti-monótona** está intimamente relacionada com o princípio APRIORI, que indica: se um conjunto de combinações é frequente, então todos os seus subconjuntos também o devem ser. Para que as combinações das agregações de um cubo consigam satisfazer a condição *iceberg*, todas elas tem de ser anti-monótonas, ou seja, todos os seus subgrupos tem de verificar a condição *iceberg*.

valores mais frequentes da primeira agregação, de maneira a que apenas o tuplo que contém o valor mais frequente da agregação, seja novamente analisado. Este método irá contar, posteriormente, a combinação de valores para as primeiras duas agregações e efectua o particionamento da base de dados para que apenas esse tuplo (que contém as combinações mais frequentes das duas primeiras agregações) seja novamente analisado e, assim por diante. Esta abordagem beneficia da aplicação da filtragem APRIORI de modo a reduzir o cálculo associado à propriedade anti-monótona, situação que não é possível aplicar numa perspectiva *Top-down*. No entanto, à medida que as bases de dados vão crescendo, o número de combinações candidatas é de tal forma elevado que torna-se incomportável o seu processamento em memória, sendo necessário investigar e evoluir os métodos de optimização mais convencionais.

ID	Local	Cliente
P1	Lisboa	Loja A
P1	Braga	Loja B
P1	Coimbra	Loja C
P2	Coimbra	Loja D
P2	Coimbra	Loja D
P2	Coimbra	Loja E
P2	Porto	Loja D
P3	Santarém	Loja F

Combinações	Count (*)
{P1, ANY, ANY}	3
{P2, ANY, ANY}	4
{ANY, Coimbra, ANY}	4
{ANY, ANY, Loja D}	3
{P2, Coimbra, ANY}	3
{P2, ANY, Loja D}	3
{ANY, Coimbra, Loja D}	2
{P2, Coimbra, Loja D}	2

```
SELECT ID, Local, Cliente, count (*)
FROM TABELAID
GROUP BY ID, Local, Cliente
HAVING count (*) > 1
```

Figura 4. Geração do cubo icebergue com suporte mínimo de 25% dos tuplos de entrada

Nesse sentido, paralelamente a estas perspectivas, surge outro algoritmo que propõe explorar em detalhe a utilização de funções não distributivas nas *iceberg conditions* (por exemplo, utilizando a função *AVG()* em vez do *SUM()*, designados por métodos *Top-k*). O 3) H-CUBING [Han et al., 2001] é uma abordagem do tipo *Bottom-up* que assenta numa estrutura de dados em árvore (*hyper-tree structure*) que privilegia o cálculo partilhado e agiliza o desempenho do processamento. Uma das vantagens deste algoritmo é o facto de ser possível

explorar algumas agregações em simultâneo e de forma partilhada, já que os nós da árvore apartam duplicação de dados. Também, esta perspectiva calcula menos combinações das dimensões, antes de prosseguir no processamento das seguintes, o que leva à filtragem do algoritmo APRIORI no processamento do cubo. Contudo, à semelhança do BUC, este método não pode usar os resultados intermédios para calcular as dimensões mais pequenas, de forma a facilitar o processamento de cubos com elevada dimensionalidade. No entanto, os algoritmos BUC e H-CUBING permitem calcular *iceberg cubes* numa perspectiva *Bottom-up* e facilitam a filtragem APRIORI. O BUC explora as técnicas de ordenação rápida e particionamento das agregações enquanto que o H-CUBING considera uma estrutura hierárquica de dados (*hyper-tree*) para efeitos de cálculo partilhado. No entanto, nenhum destes métodos explora verdadeiramente todas as potencialidades da simultaneidade das agregações multidimensionais. Note-se que nessa altura foi realizada uma investigação por [Wagner & Yin, 2001] que veio contribuir positivamente para a evolução do algoritmo BUC e que apresentou novas propostas de algoritmos destinados ao processamento distribuído.

A solidificação e evolução dos conceitos desenvolvidos neste âmbito permitiu dar origem a um novo método híbrido de selecção de *iceberg cubes*, o STAR-CUBING (SC) [Xin et al., 2003] que integra a força dos três algoritmos anteriores e executa simultaneamente o cálculo de agregações em dimensões múltiplas. Na prática, utiliza uma árvore hierárquica em estrela (s*tar-tree*), amplia os métodos de agregação simultânea e possibilita a filtragem (APRIORI *pruning*) da condição *group-by* que não satisfaz a *iceberg condition*. A análise preliminar efectuada em [Xin et al., 2003] revela que este algoritmo é eficiente e supera todos os seus antecessores em quase todos os tipos de distribuições de dados (robusto em distribuições densas e enviesadas) . No entanto, em distribuições de dados muito dispersos, o desempenho deteriora-se sobretudo devido ao custo do processamento adicional introduzido pela estrutura em árvore. A contribuição dos algoritmos anteriores e o desenvolvimento de novas linhas de investigação sobre a optimização dos métodos de selecção de *iceberg cubes*, permite que seja apresentada uma nova abordagem, diferente de todas as anteriores, que foca a importância da distribuição dos dados, até então, não reconhecida.

Nessa perspectiva, o algoritmo MM-CUBING [Shao et al., 2004] surge como uma alternativa no cálculo de *iceberg cubes*, operando através da factorização do *lattice* distribuído de acordo com a frequência das agregações. A análise preambular efectuada mostra que este método é totalmente eficiente e que se pode adaptar a qualquer distribuição de dados. Apesar da limitação que envolve o consumo elevado de memória devido à invocação recursiva das estruturas utilizadas no processamento, o MM-CUBING revela-se o mais promissor dos algoritmos, já que é o único até agora a conseguir um bom desempenho em todo o tipo de distribuições de dados, sejam elas, densas, dispersas ou enviesadas. Decorrente do STAR-CUBING, surge recentemente um novo algoritmo híbrido vocacionado especialmente para o processamento em paralelo de *iceberg querys* de grande porte. Repare que a maioria soluções propostas até ao momento para o problema do processamento de *iceberg cubes* operam numa perspectiva centralizada, ou seja, aplicadas um único sistema central de processamento analítico, responsável pela gestão de todas as consultas e interrogações geradas e pelo desempenho do cubo. Importa, por isso, evidenciar neste trabalho alguns algoritmos vocacionados para ambientes paralelos e distribuídos, podendo esta última conhecer uma evolução associada à distribuição do processamento e, consequentemente, aumentar o desempenho no cálculo de *iceberg cubes*. Veja-se, então, a seguinte proposta.

O método PnP (*Pipe n'Prune*) [Chen et al., 2005] revela-se inovador na medida em que assenta numa estrutura sólida, constituída pela integração da abordagem *Top-down* com a capacidade de redução de dados, facilitada pela filtragem APRIORI (*pruning*), intrinseca à perspectiva *Bottom-up*. A robustez desta aproximação é comprovada pela eficácia da sua aplicação em cenários que utilizem *iceberg querys* de forma sequencial, distribuída e em paralelo e que recorram à utilização de memória externa para o seu processamento. As análises de desempenho efectuadas a este método revelam que, para o primeiro cenário, o PnP pode ser adoptado, tanto em distribuições densas como em distribuições dispersas, sendo uma alternativa interessante ao BUC e ao STAR-CUBING. No segundo cenário, o PnP mostra-se eficiente em lidar com o I/O da memória externa, cujo tempo de leitura/escrita é duas vezes menor em comparação com o tempo do cálculo total da *iceberg query* na memoria física. Por último, importa referir que este algoritmo é considerado o mais adequado para ser utilizado

em sistemas distribuídos, devido a sua escalabilidade e facilidade no cálculo partilhado. De facto, o PnP revela-se muito eficiente em distribuições reais de dados e adapta-se muito bem a situações cujas agregações são de processamento complexo (por razões de espaço de armazenamento, elevada dimensionalidade e cardinalidade). Outras abordagens envolveram a adaptação de métodos icebergue em função das necessidade especificas de cada universo de utilização. É exemplo disso, o método C-CUBING [Xin et al., 2006] que integra algoritmos de cálculo de *iceberg cubes* (MM e SC) com os benefícios propostos por outro método de redução de dados, denominado por *closed cubes* [Lakshmanan et al., 2002], [Lakshmanan et al., 2003]. Esta lógica de simplificação defende a compressão eficiente das agregações integrais do cubo original. O algoritmo beneficia com a tecnologia de ambos.

Recentemente, foram propostas mais duas soluções para processamento de *iceberg cubes*: o IX-CUBING [Jian et al., 2007], que está dirigido exclusivamente para o processamento optimizado de *iceberg querys* sobre dados no formato XML e o MT-CUBING [Li et al., 2008] que integra uma solução *multi-tree* híbrida, que permite processar estas estruturas através de abordagens *Top-down* e *Bottom-up* de modo a beneficiar das vantagens de cada um deles (inclusive da propriedade APRIORI). Em último lugar, mas não menos importante, refere-se um algoritmo de processamento de *iceberg cubes* que foge às abordagens tradicionais, o CT-CUBING (*cross table cubing*) [Cho et al., 2005]. Esta solução, ao contrário de todas as outras mencionadas, recorre ao cruzamento de várias tabelas disponíveis no *Data Warehouse* em vez de efectuar o cálculo das agregações recorrendo a uma única tabela universal materializada. Um proposta interessante para grandes repositórios de dados. Note-se que não se pretende uma exposição exaustiva de todas as soluções e algoritmos existentes sobre cubos icebergue, mas apenas referir as mais representativas, pelo seu grau de inovação quando foram apresentadas, pela sua importância ao abrirem uma nova linha de investigação ou família de soluções ou, ainda, pela sua eficácia na actualidade. Recorde-se que a concretização do algoritmo apropriado de selecção de cubos multidimensionais de dados deve acompanhar as constantes evoluções do negócio. O surgimento de novos desafios incentiva a que as soluções analíticas existentes sejam adaptadas e estendidas às novas realidades. Nessa perspectiva, aparece a possibilidade de tirar partido do perfil de utilizador, de maneira a

permitir seleccionar as estruturas mais apropriadas. Mas, alterando-se as necessidades dos decisores, será necessária a revalidação das estruturas multidimensionais. Vai haver subcubos materializados que deixarão de ser úteis, e outros, não materializados, que passaram, quiçá, a ser benéficos. Esta reestruturação deve acompanhar as constantes mudanças da realidade envolvente, mas devido à dimensão e complexidade das estruturas multidimensionais, o custo é normalmente, muito elevado. Tendo em conta as abordagens do tipo estática, dinâmica ou pró-activa, será necessário avaliar as formas mais vantajosas, tendo em conta a realidade do sistema e sua dimensão, de maneira a garantir que as vistas materializadas estão coerentes com o negócio e com as necessidades reais dos utilizadores.

1.3. Motivação e Objectivos da Dissertação

Estamos num período em que o crescimento dos sistemas de *Data Warehousing* dão lugar a novos desafios. Cada vez mais, gestores e analistas recorrem a sistemas de processamento analítico para calcular, manipular e analisar um grande volume de dados sob múltiplas perspectivas de modo a facilitar o processo de tomada de decisões. O crescimento do tamanho do *Data Warehouse* e do número de utilizadores impõem fortes barreiras ao processamento analítico em tempo útil. De facto, os problemas decorrentes da selecção de cubos multidimensionais de dados, motivados principalmente pelo tempo de cálculo e espaço de armazenamento, são cada vez mais comuns, consequência da evolução e crescimento dos sistemas de processamento analítico.

A melhoria dos algoritmos de selecção de cubos e utilização de heurísticas são factores que devem ser tomados em consideração para minimizar ou mesmo ultrapassar os problemas decorrentes do processamento analítico complexo. Nessa perspectiva, este trabalho tenciona disponibilizar um estudo comparativo de avaliação de diferentes algoritmos de selecção de cubos, recorrendo a técnicas de *iceberg cubing*. Ao detalhar-se cada um dos algoritmos de *iceberg cubing* aliados a uma politica de selecção e materialização de vistas consolidada é possível seleccionar, com um maior grau de confiança, a solução que mais se adequa a cada tipo de problema, de modo a minimizar o tempo de resposta às interrogações efectuadas sobre

as agregações do cubo e, assim, disponibilizar atempadamente a informação requerida pelos decisores. Paralelamente, serão apresentadas análises comparativas de desempenho entre as várias propostas de solução que contribuem positivamente para as conclusões e trabalho futuro a desempenhar. Os objectivos determinados para o âmbito desta investigação encontram-se resumidos na tabela 1.

Objectivos	Tarefas
Desenvolver o conceito de estruturas multidimensionais de dados / cubo de dados	Avaliar, analisar, investigar e evoluir os conceitos associados a estruturas multidimensionais de dados. Desenvolver o conceito de cubo de dados e explorar a sua evolução e características. Exploração de dados em diferentes perspectivas. Justificar a utilidade e poder deste tipo de estruturas.
Investigar soluções de optimização para materialização de vistas sobre cubos de dados. Selecção de vistas materializadas	Obter um elevado desempenho na visualização do cubo, o que significa uma selecção mais eficaz dos subcubos de dados, com o objectivo de melhorar a satisfação dos utilizadores e qualidade das decisões.
Explorar eficazmente o conceito de *iceberg cubing* e desenvolver os métodos de aplicação	A selecção de vistas materializadas, através da aplicação de técnicas de *iceberg cubing*, tem com objectivo mostrar os níveis de ganhos elevados apenas com 10-20% de utilização do espaço de materialização do cubo de dados. Isto significa diminuir o tempo de resposta das agregações, tendo igualmente em consideração o perfil de utilização.
Elaboração de um estudo comparativo sobre os vários métodos de *iceberg cubing*	Efectuar uma avaliação prática, através do comparativo de diferentes algoritmos de *iceberg cubing* e optimização de *iceberg querys*.

Tabela 1. Síntese do âmbito do trabalho de investigação proposto

1.4. Metodologia Adoptada

A metodologia adoptada na realização deste trabalho foi um pouco simplista e convencional de modo a enquadrar facilmente o leitor no foco da problemática estudada, evidenciando os seus aspectos, com especial ênfase no cubo de dados e processamento analítico, seus desafios e soluções. Na realidade, iniciou-se por 1) fazer uma abordagem geral ao tema, gerando uma contextualização e um "*estado da arte*" de modo a posicionar este trabalho no terreno e efectuar um levantamento da literatura existente, aprofundando os conceitos necessários que servem de base para a dissertação. A isto costuma-se chamar a fase de identificação e reconhecimento. De seguida, 2) avançou-se para o estudo das estruturas multidimensionais relativamente à sua orgânica, materialização e exploração, como forma de abrir caminho para as técnicas de *iceberg cubing*. Tendo-se desenvolvido os conceitos de *iceberg cubing* e interiorizado os métodos e algoritmos, ficaram concluídos os fundamentos necessários para investigar 3) a forma de como materializar "melhor" os icebergues, optimizando o seu processamento e a sua aplicação, em função da capacidade de satisfazer as *queries* a que se destina. Posteriormente, 4) foi apresentado um estudo comparativo das abordagens *iceberg*, com o objectivo de criar um suporte de investigação que possa auxiliar na escolha do algoritmo mais adequado na resolução de um determinado problema. Em suma, foi adaptada tipicamente uma abordagem *Top-down*, desenvolvendo as temáticas genéricas no domínio das estruturas dimensionais em ambientes OLAP, particularizando posteriormente, as áreas mais proeminentes da optimização e selecção de cubos multidimensionais.

1.5. Estrutura da Dissertação

Este trabalho tem como objecto de estudo as estruturas multidimensionais de dados, vulgarmente designadas por cubo de dados. Importa, por isso, sensibilizar sobre o problema a ser analisado, mostrando as suas vertentes, com especial ênfase na optimização de métodos de selecção de vistas materializadas em ambientes de processamento tipicamente analítico, evidenciando igualmente, os seus desafios e evoluções. Nessa perspectiva e, de acordo com âmbito proposto na secção 1.3, além do presente capítulo, este documento encontra-se

estruturado em mais 3 capítulos, segmentados pela natureza dos assuntos abordados, nomeadamente:

- **Capítulo 2** – O referenciado "estado da arte" no que diz respeito ao estudo de estruturas multidimensionais, onde estão desenvolvidos os conceitos em volta do cubo de dados (o que é, para que serve, como se desenvolve, qual a sua evolução, etc.). São ainda investigadas as técnicas de selecção de dados e criação de sub-cubos, introduzindo os assuntos sobre selecção de estruturas multidimensionais e optimização de vistas materializadas.

- **Capítulo 3** – Exploração do conceito de *iceberg cubing*, como método optimizado de selecção de cubos, incluindo o desenvolvimento dos conceitos de *iceberg querys*, *iceberg conditions* e investigação dos diversos algoritmos envolvidos nesta tecnologia. Elaboração de um estudo de avaliação que compara diferentes algoritmos de optimização baseados nos métodos de *iceberg cubing*. Avaliação da eficiência dos algoritmos em função da natureza de um determinado cenário analítico (exposição duma análise comparativa das diferentes lógicas de cálculo de *iceberg cubes* de modo a medir a mais adequada face a um determinado problema analítico).

- **Capítulo 4** – Apreciação crítica do trabalho desenvolvido. Concretização do levantamento dos contributos que esta dissertação pode outorgar na evolução dos métodos de selecção, por intermédio da optimização envolvida na escolha das agregações mais adequadas de um cubo. Em último lugar, são referenciadas algumas áreas de aplicação para trabalhos futuros.

Capítulo 2

A Preponderância das Estruturas Multidimensionais de Dados

2.1. A Origem das Estruturas Multidimensionais de Dados

No inicio dos anos 70, Edgar F. Codd propôs o modelo de dados relacional. Tal como é conhecido, este modelo assenta no pressuposto de que os dados (que cumprem certas restrições) podem ser tratados da mesma maneira que as relações matemáticas. No esquema relacional, uma relação é vulgarmente designada por tabela, caracterizada por ser uma estrutura bidimensional que respeita um determinado esquema conceptual, com zero ou mais instâncias. O corpo de uma relação é constituído por um ou mais atributos que dão significado aos dados a armazenar. A cada instância do esquema conceptual de uma relação designa-se por tuplo (registo). Para cada atributo de uma relação, é possível definir o seu domínio – este determina o intervalo de valores possíveis que podem ser assumidos por esse mesmo atributo. O número de atributos que compõem o esquema conceptual de uma relação são designados por grau da relação. Por sua vez, ao número de tuplos de uma relação dá-se o nome de cardinalidade da relação. Além de conceber o modelo relacional, Codd propôs ainda dois métodos para a sua manipulação: a álgebra e o cálculo relacional, que actuam e desenvolvem relações sobre conjuntos, retornando o resultado sob a forma de novos conjuntos. Na álgebra relacional, dado que uma relação é um conjunto, então todas as operações citadas pela teoria dos conjuntos podem ser aplicadas às relações. Adicionalmente, definem-se outros operadores do modelo relacional para selecção, divisão e junção de relações.

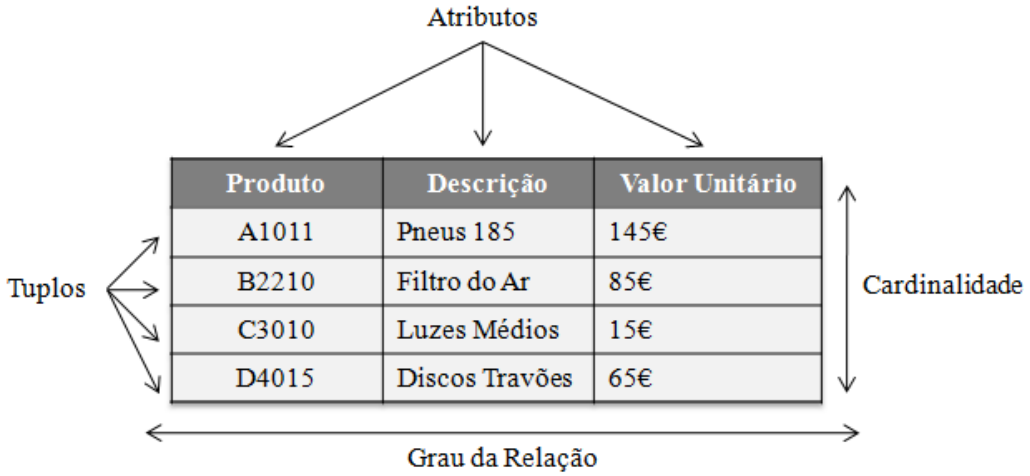


Figura 5. Representação de uma relação (tabela) de atributos

Cada registo de uma relação deve ser identificado univocamente por um atributo ou conjunto de atributos designados por chave. De uma forma simplificada, sendo a chave o identificador único de uma instância, é possível estabelecer uma ligação entre diversas relações de modo a representar mais fielmente o mundo real. Inerente a este conceito e considerando o principio da integridade dos dados, o modelo relacional propõe regras para normalizar o agrupamento dos atributos de modo a minimizar o número de dependências funcionais existente com o intuito de eliminar fontes de redundância nos dados, contribuindo para diminuir os problemas decorrentes da manutenção entre relações, custos de armazenamento de dados e desempenho. De facto, é visível que o modelo relacional proposto por Codd reúne um conjunto de características agradáveis, vocacionadas para a modelação e processamento dos dados de forma transaccional (OLTP[4]). Na prática, a normalização do modelo certifica a consistência e integridade dos dados, possível através das transacções que decorrem. Cada transacção é responsável por um número pequeno de tuplos, num ambiente em constante actualização. Esta realidade mostra-se um pouco difícil quando se pretende efectuar uma análise de dados útil, que resulte num conjunto de tuplos de interesse para o utilizador. Uma consulta irá solicitar,

[4] **On-Line Transaction Processing**, referindo-se a uma classe de sistemas orientados para a gestão de aplicações transaccionais. As transacções realizadas são criadas para manter a consistência de um sistema (normalmente, uma base de dados ou sistema de ficheiros) assegurando que todas as operações são registadas e executadas, do principio ao fim, de uma forma conhecida e interdependente.

usualmente, operações que envolvem a junção de várias tabelas, motivadas muitas vezes, pela dispersão dos dados ou pelo nível de normalização alcançado. As operações que envolvem a união de muitas tabelas são conhecidas por terem custos elevados a nível de desempenho, mesmo quando são usados os melhores procedimentos. Estas acções, como se sabe, tornam as análises efectuadas mais demoradas.

As dificuldades sentidas pelas organizações na análise dos dados em tempo útil, num ambiente de crescente competitividade e em constante mudança, obriga-as a dispor de meios propensos à diminuição da incerteza. Um conhecimento mais consolidado e organizado permite que a gestão do processo de tomada de decisão seja mais exacto, de modo a possibilitar um melhor posicionamento das organizações no mercado concorrente. Mas, de facto, esta necessidade colidia com as restrições de uma simples transacção na base de dados, em que o resultado de uma interrogação é apresentado sob a forma de uma lista, longa e de difícil leitura. Os agentes de decisão começaram, então, a sentir necessidade de manipular livremente os dados e analisá-los de acordo com as suas precisões de negócio. Acompanhando esta necessidade com a evolução dos sistemas de informação e do mercado empresarial, introduz-se, na década de 90, uma nova classe de ferramentas analíticas, baptizadas de OLAP pelo próprio Codd [Codd. et al., 1993].

O termo foi citado pela primeira vez quando Codd definiu doze regras[5] que estas aplicações deveriam atender, notabilizando-se a capacidade de visualizar conceptualmente os dados de negócio numa perspectiva multidimensional. Esta capacidade dos modelos OLAP advém da visão que cada agente de decisão tem do negócio e da forma como é permitida a interacção com o utilizador (pode "dialogar" directamente com o modelo lógico multidimensional do sistema OLAP de modo a formular livremente as suas consultas). A visão multi-perspectiva do negócio, em que um dado facto pode ser analisado em função de diferentes significados. A visão multidimensional é, portanto, constituída por um conjunto de consultas que fornecem dados a respeito de medidas de desempenho sobre uma determinada perspectiva de negócio, decomposta por uma ou mais dimensões. Deste modo, as características de um sistema OLAP

[5] Note-se que as regras definidas por Codd não foram consideradas muito consensuais.

assentam num modelo de dados multidimensional [Kimball, 1996] e a visualização multi-perspectiva dos dados processados nesse modelo são facilitados pela representação lógica, figurada, de um cubo de dados.

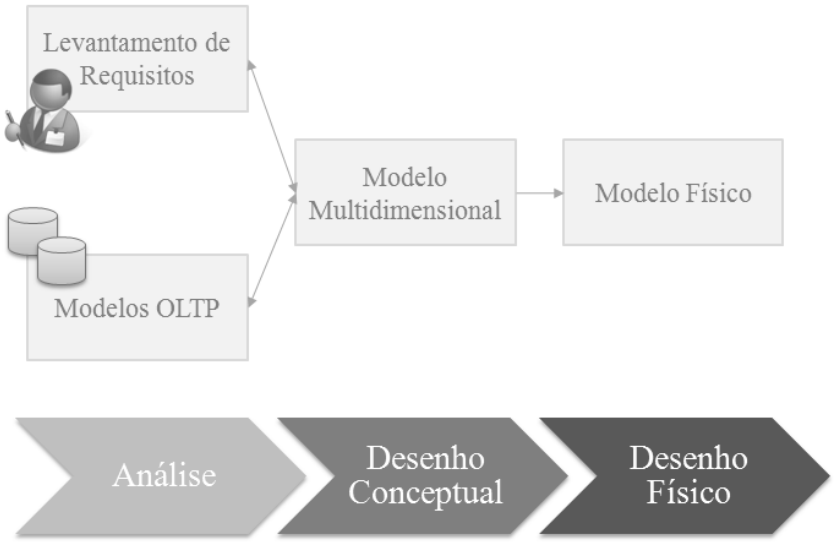


Figura 6. Processo conceptual genérico de um ambiente OLAP

De facto, os modelos de dados multidimensionais são pensados expressamente para suportar o processamento analítico OLAP: navegar nos dados, em vários níveis de agregação, seleccionando algumas perspectivas e ocultando outras, usar funções de agregação, manipular dimensões, etc. A visualização multidimensional de três dimensões de análise é vulgarmente designada por um cubo, onde cada dimensão corresponde a cada caracterização do facto em análise. Na realidade, o número de dimensões a analisar costuma ser superior, devendo-se então falar em hipercubo (generalizado para *n* dimensões).

2.2. Cubos de Dados

O cubo não é mais do que uma reprodução multidimensional dos dados modelados, tendo sido proposto inicialmente em [Gray et al., 1996] como uma difusão do operador SQL[6]

[6] **Structured Query Language**, define-se por ser uma linguagem de pesquisa estruturada para efectuar e gerir consultas sobre os dados armazenados em bases de dados relacionais.

group-by. Na prática, o cubo representa uma grelha multidimensional construída a partir dos valores de cada uma das dimensões envolvidas. Cada célula da grelha guarda a respectiva agregação calculada (métricas, valores numéricos acondicionados dentro de cada célula), todas elas com o mesmo significado a nível de combinação de coordenadas, instância das dimensões e nível de granularidade. O significado de dimensão é uma noção fundamental em dados multidimensionais. Essencialmente, estas usam-se para seleccionar os dados do cubo e efectuam o seu agrupamento até a um determinado nível (granularidade). Cada dimensão encontra-se organizada sob a forma de uma ou mais níveis, designados por hierarquias. As hierarquias contêm um conjunto distinto de níveis que correspondem a diferentes graus de detalhe e formas de classificação.

Ao navegar nos diferentes graus de detalhe, diz-se que o nível A *roll-up*[7] para o nível B se uma classificação dos elementos de A, de acordo com os elementos de B, tiver significado semântico para a análise OLAP (ex. o nível "distrito" *roll-up* para o nível "país"). Um nível pode fazer *roll-up* para qualquer número de níveis, permitindo assim a existência de múltiplas hierarquias numa dada dimensão. Isto será possível se existirem vários critérios de classificação para os membros de uma dimensão (ex. uma oficina pode ser classificada pela sua localização geográfica ou pelo seu tipo). Outra situação que pode ocorrer é quando vários caminhos de *roll-up* são possíveis entre dois níveis, chamados caminhos alternativos ou dimensão múltipla. Veja-se de seguida, na figura 7, o exemplo da representação da dimensão "oficina". A estruturação das dimensões do ponto de vista do analista é completada com a utilização de atributos de nível de dimensão: descrevem os membro de um dado nível de dimensão, mas não definem hierarquias (ex. o nome e morada de uma oficina ou o nome de uma dada região). Uma representação multidimensional pode conter mais que um cubo de diferentes granularidades (chamados hipercubos). Esta situação é usual e necessária sempre

[7] ***Roll-up*** é uma das operações comuns efectuadas sobre estruturas multidimensionais de dados em ambientes analíticos tipicamente OLAP, que permitem visualizar os dados multidimensionais em diferentes níveis de agregação (detalhe). Outras operações comuns são: *drill-down/up, slice, dice* e *pivot*. Concretamente, o *roll-up* refere-se ao processo de analisar os dados com menos granularidade.

que existe a necessidade de uma aplicação OLAP requerer a análise de diferentes factos (ex. vendas de veículos e reparações de veículos).

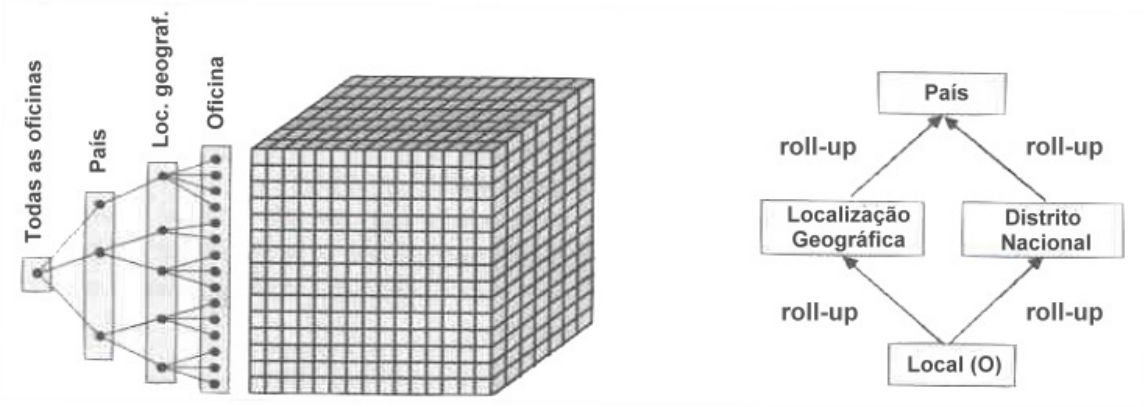


Figura 7. Hierarquia de níveis da dimensão "oficina" e caminhos alternativos da dimensão

É representado, num cubo de dados, o assunto em análise pelo agente de decisão, designado dimensionalmente, por facto (ex. reparações de veículos). Para melhor apreender esta representação, vejamos o seguinte exemplo: um fabricante de automóveis pode querer analisar as reparações de veículos de modo a melhorar o seu produto, definir novas políticas de garantias e obter informação acerca da certificação de oficinas de manutenção. Assim, o construtor vai analisar as reparações de veículos (facto de negócio) segundo um conjunto de perspectivas (anteriormente caracterizadas por dimensões). O nível de dimensões ao qual ocorre a combinação dos membros irá determinar a granularidade do facto. Na figura 8, é efectuada a representação multidimensional desta situação. Repare-se que a granularidade do cubo é "fabricante por país e por mês", aqui representado por "país *x* fabricante *x* mês" ou simplesmente pela combinação de letras correspondentes a cada dimensão e hierarquia (pfm). A granularidade será maior à medida que o nível de combinações dos valores das dimensões aumenta, por exemplo, "especialidade fabricante *x* zona *x* mês" (conforme a hierarquia dimensional evidenciada na figura 7) ou menor, por exemplo, "fabricante *x* país *x* mês". Um cubo de maior granularidade pode, normalmente, ser obtido a partir de cubos de granularidade mais baixa, seguindo exclusivamente as hierarquias dimensionais que, no limite, podem implicar mesmo a sua total compressão (a dimensão é agregada até ao seu nível máximo,

"todas as oficinas", na figura 7). Esta designação ficará certamente mais clara seguindo o exemplo de representado na figura 9.

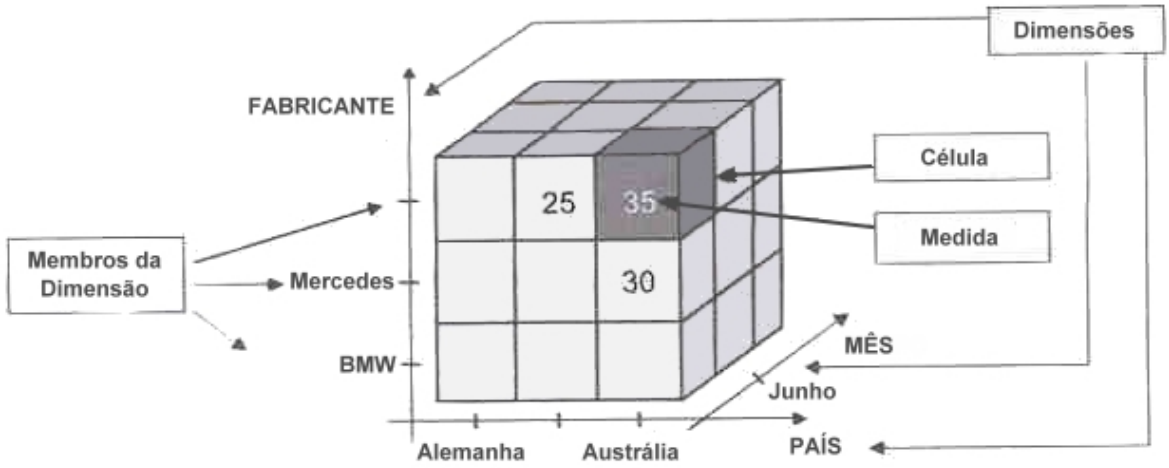


Figura 8. Esquema multidimensional para o facto de negócio de reparações de veículos

De facto, a dimensão fabricante é comprimida, sendo os agregados, os valores das células correspondentes a cada "país *x* fabricante *x* mês", gerando-se o subcubo "país *x* mês". Pode agora clarificar-se o significado de "todos", muitas vezes representado também como "nenhum", que significa o topo de cada hierarquia. É estranha esta dupla designação oposta, mas compreensível: tudo depende de que perspectiva se está a olhar para o processo de agregação, sendo por isso possível, visualizar os níveis de ambas as formas. A designação "nenhum" de representação (-) deriva do facto de resultar do processo de colapso da dimensão, onde a dimensão é omitida. A nomeação "todos" significa que todos os membros da dimensão encontram-se agregados. Do exemplo evidenciado na figura 9, verifica-se que, para o fabricante de automóveis BMW, no mês de Fevereiro, por exemplo, existe uma compressão da dimensão país até chegar ao nível máximo de agregação, ou seja, (Alemanha, BWM, Fevereiro) + (Austrália, BMW, Fevereiro) + (Portugal, BMW, Fevereiro). Esta situação caracteriza uma acumulação agregada ao longo da dimensão país, que partiu do significado "nenhum" em direcção a "todos". Esta possibilidade de gerar um cubo utilizando outros, destaca as relações de dependência existentes que estão na base do chamado *lattice* de correlações ou dependências, indicado em [Harinarayan et al., 1996], que conceptualiza todas as agregações possíveis e respectivas dependências. Seguindo os exemplos apresentados, são

supostas três dimensões (fabricante, país e tempo) e também, a existência de níveis hierárquicos intermédios (por exemplo, o nível país agrupa instantaneamente no nível "todos"). Como existem três dimensões, vão ter-se $\mathbf{2^3}$ cubos possíveis.

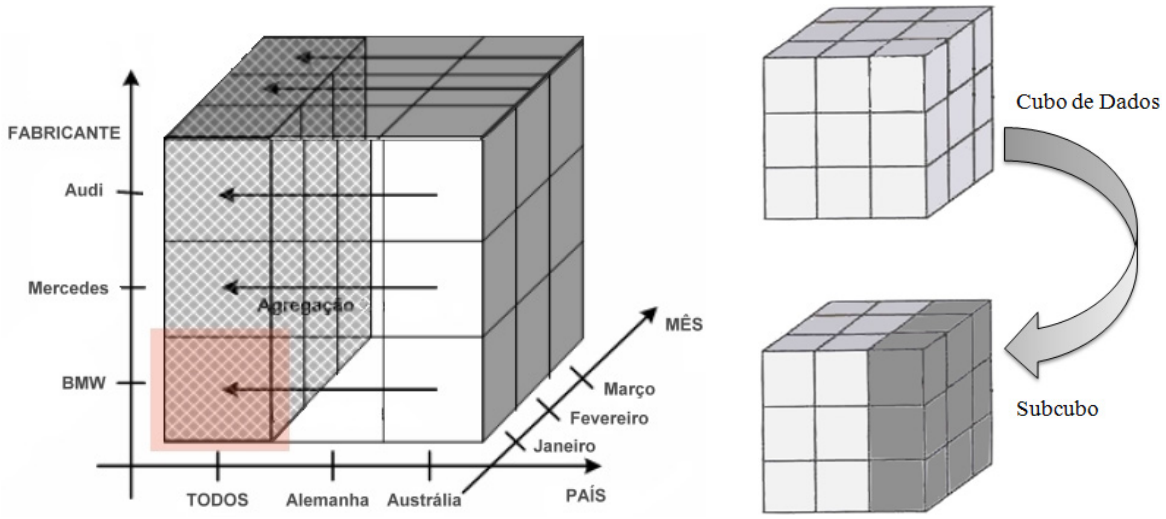


Figura 9. Representação do subcubo de nível máximo de agregação, através do cubo de dados

O número de cubos é calculado, de modo genérico, pela multiplicação do número de dimensões e o número de níveis hierárquicos da dimensão. Sendo o resultado um produto, o número de cubos relativo a um dado modelo dimensional pode assumir um crescimento exponencial repentino, podendo facilmente ficar muito grande. Na realidade, basta o modelo ter cinco dimensões *i*, cada uma com quatro níveis hierárquicos *L*, para se ter um total de $\prod_{i=1}^{n}(Li + 1)$ = 1024 subcubos ou ($\mathbf{4^5}$). O cubo de dados não é mais do que uma agregação dos dados a um determinado nível de granularidade, entendido por um maior ou menor detalhe. Vendo o *lattice* apresentado na figura 10, a agregação (pfm) terá uma menor granularidade do que a agregação (p--). Importa ressalvar que, de facto, o cubo designa muitas vezes o próprio espaço multidimensional e, especialmente, representa o conjunto de todas as agregações possíveis e seus subconjuntos. Este crescimento combinatório acentuado dá origem a uma nova problemática, extremamente importante, em sistemas de processamento analítico: a grande dimensão do espaço físico dos dados agregados quando materializados, a necessidade inevitável de limitar o seu tamanho e o custo da sua actualização, sempre que tiverem de ser refrescados.

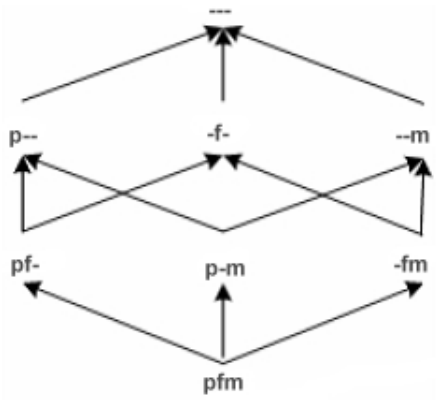


Figura 10. Lattice de correlações associado às dimensões país, fabricante e mês (tempo)

2.3. Benefícios do Cubo de Dados

Nem todas as consultas agregadas efectuadas no ambiente de processamento analítico necessitam de estarem calculadas e armazenadas num cubo de dados. Por exemplo, agregações consideradas simples, cujo cálculo é reduzido, podem ser obtidas sem haver necessidade de interrogar o cubo. Porém, implicam a sua leitura e posterior agregação, antes de entregarem o *output* desejado. Operações desta natureza podem ser muito demoradas, já que podem envolver muitos registos no seu cálculo, mesmo se a base de dados possuir uma politica de índices consolidada. Esta realidade é, de todo, antagónica com o *modus operandi* de um sistema de processamento analítico OLAP, já que dele depende a qualidade do processo de tomada de decisões e a respectiva produtividade dos decisores. No entanto, se uma determinada agregação estiver já calculada e guardada no sistema, a resposta à interrogação efectuada pela consulta, pode ser imediata. É, de facto, uma solução que prima-se pela sua classe. Veja-se agora o seguinte exemplo: uma consulta que solicite o total de vendas mensais, relativas a um determinado ano, será respondida de imediato pelas agregações correspondentes ao subcubo (--m) do *lattice* de correlações representado na figura 10, que precisará de poucas operações (ou nenhumas, se de facto o mês for a granularidade considerada para a data), evitando-se assim a selecção e cálculo posterior dos próprios dados base (possivelmente, serão algumas centenas de milhares de registos).

De facto, os benefícios provenientes da utilização de cubos OLAP manifestam-se em várias disposições. Mesmo se o subcubo idealmente mais adaptado à interrogação efectuada não estiver disponível, as correlações que existem entre subcubos podem ser utilizadas para encontrar outro que, de uma forma alternativa, possa ser usado para retornar uma resposta mais exacta daquilo que se pretende. Possivelmente, envolverá a leitura de um maior número de células (e posterior agregação), mas ainda assim, o custo será tremendamente menor em comparação com os custos decorrentes da utilização dos dados base. Ao conhecerem-se os subcubos a materializar e os seus custos, será muito fácil escolher qual é que se vai usar para dar resposta a uma determinada interrogação. O exemplo da figura 11, cujo *lattice* de subcubos encontra-se materializado parcialmente, permite ilustrar esta situação. Veja-se o seguinte: um analista de negócio pretende saber o número total de vendas agregadas por país (entre parêntesis, está indicado o custo de utilização de cada subcubo). Uma vez que o subcubo (p--) não se encontra materializado (que implicaria apenas um custo de resolução de 50), é necessário encontrar alternativas: os subcubos (p-m), (pfm) e ainda considerando as relações base, a custos 600, 6000 e 18000, respectivamente.

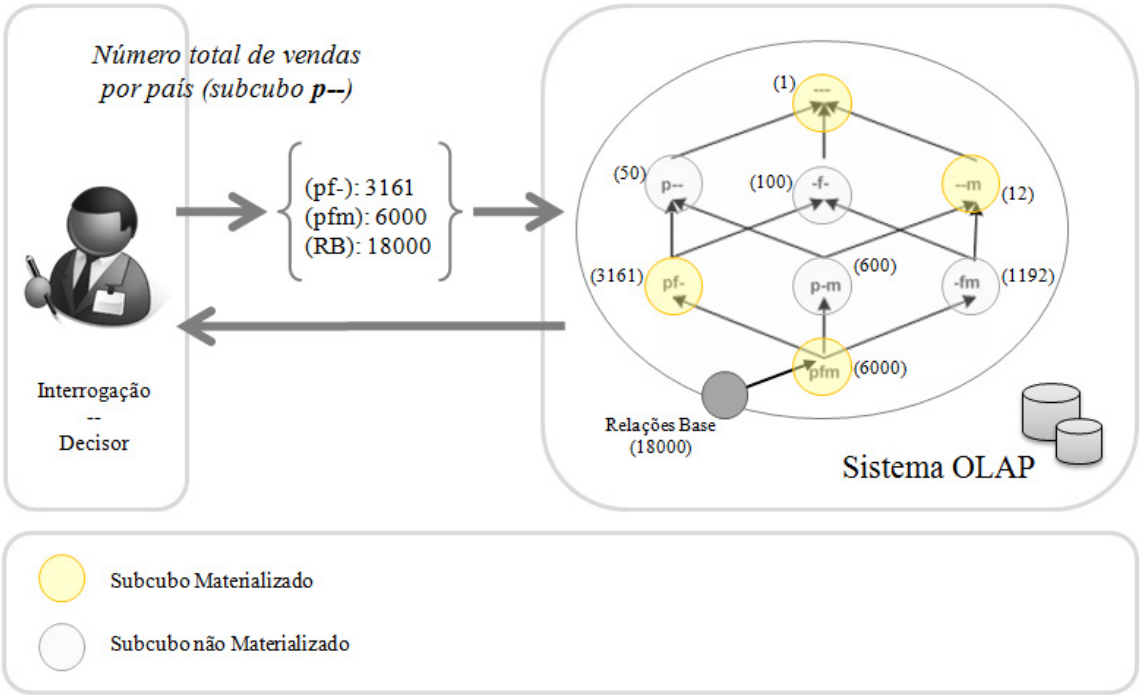


Figura 11. Selecção de um subcubo para responder a uma interrogação

Nesta situação, seria escolhido o subcubo (p-m) para responder o mais exacto possível à interrogação solicitada ao sistema OLAP, já que seria aquele que envolveria o menor custo (dos subcubos disponíveis). Claro que a resposta ideal a esta interrogação seria obtida através do subcubo (p--) que, estando materializado, implicaria um custo muito reduzido (50), sendo a resposta quase imediata. Mesmo assim, esta afirmação só seria verdadeira para esta interrogação. Diferentes interrogações implicariam certamente, para uma resposta ideal, combinações de outros subcubos (materializados ou não). Cada vez se torna mais claro que a materialização do cubo (total ou parcial) é uma condição fundamental de desempenho num sistema tipicamente OLAP. Aqui inclui-se também a importância da definição da natureza de utilização, de modo a optimizar a selecção dos subcubos mais requeridos pelos agentes de decisão com um determinado perfil, na impossibilidade de materializar todo o cubo. Estas questões levantam outras tão ou mais importantes, já que condicionam o desempenho e a fiabilidade de todo o sistema de processamento analítico: o espaço de armazenamento, o tempo de cálculo, a materialização dos dados e o tempo de actualização (as alterações efectuadas nas relações base deverão ser reflectidas em todos e quaisquer subcubos materializados). Adicionalmente, ocorre ainda um problema delicado: quais os cubos mais adequados para serem seleccionados. Esta problemática irá ser discutida com mais detalhe, mais à frente, neste trabalho.

2.4. Reconhecimento de um Cubo de Dados

Para se entender de que forma os sistemas de processamento analítico determinam a esquematização de um cubo de dados, utilizemos um exemplo de uma consulta típica a este tipo de estruturas, relativo a uma empresa que comercializa e efectua reparações de automóveis. Para o fazer, propõe-se a utilização de um esquema extensivo ao modelo entidade/relação (E/R) [Chen, 1976] que permita usá-lo para modelar os dados de forma multidimensional, ao qual dá-se o nome de modelo multidimensional E/R (ME/R) [Sapia et al., 1999]. Assim, na figura 12, são representadas três dimensões: veículo (com a hierarquia modelo » marca » fabricante), tempo (com a hierarquia dia » mês » ano) e oficina (dimensão múltipla, com hierarquia paralela nos níveis: tipo de oficina e região. Este último ainda

depende do nível país). As métricas de interesse para a análise do facto encontram-se também evidenciadas (irá usar-se, a titulo ilustrativo, apenas a métrica # de veículos reparados).

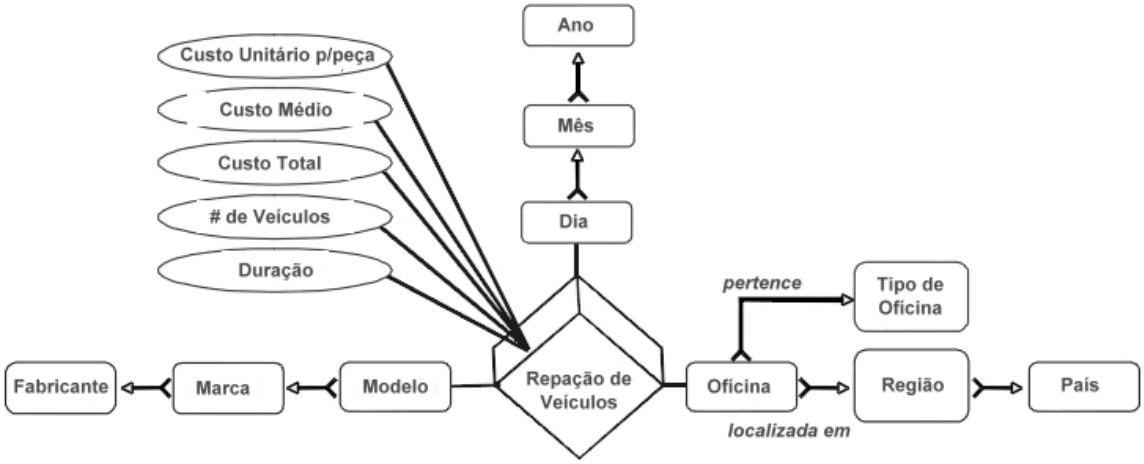
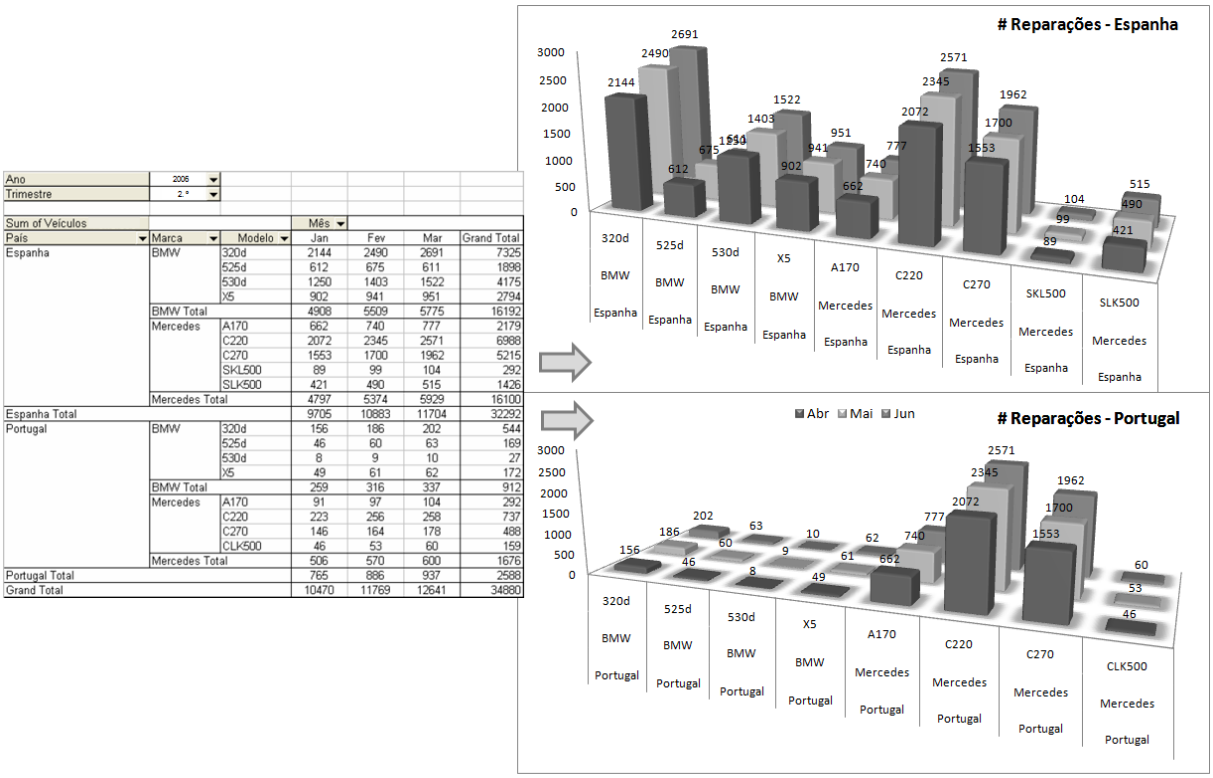


Figura 12. Representação multidimensional do exemplo, usando a notação ME/R [Sapia et al., 1999]

Usualmente, um analista inicia a consulta através de um relatório de negócio padrão com o número total de reparações, por exemplo, relativos ao segundo semestre de 2006, por país, dos veículos de marca BMW e Mercedes (agrupados por modelo). O formato desta interrogação está representado na figura 13. Nela estão representados os resultados das reparações de veículos através de uma perspectiva multidimensional, sob a forma de uma tabela pivot construída no Excel. Algumas dimensões são restritas a um único valor (através da execução da operação OLAP de *slicing*[8]), neste caso, a dimensão tempo: ano e trimestre. Estas dimensões designam-se também por "dimensões de selecção", pois todo o resultado depende das condições por elas designadas (dados apenas do 2º semestre de 2006). Outros membros de dimensões (país, marca, modelo e mês, por exemplo) vão ser mostrados distribuídos por cada um dos eixos (na tabela da figura 13, linha para os primeiros três e coluna para o último), chamados de dados qualificadores ou "dimensões resultado", já que o facto vai ser analisado segundo os valores agregados para cada um dos membros associados

[8] ***Slicing*** ou ***Slice*** é uma das operações mais comuns efectuadas sobre estruturas multidimensionais de dados em ambientes OLAP, que permitem visualizar os dados multidimensionais em diferentes níveis de agregação (detalhe). Concretamente, refere-se a um subconjunto de uma matriz multidimensional e corresponde a um valor único para um ou mais membros de uma dimensão, fora desse grupo.

nestas dimensões, mostrando o resultado nas respectivas células multidimensionais, gerando as chamadas "medidas resultado" ou métricas agregadas, da interrogação.



Ano	2006					
Trimestre	2.º					
Sum of Veículos			Mês			
País	Marca	Modelo	Jan	Fev	Mar	Grand Total
Espanha	BMW	320d	2144	2490	2691	7325
		525d	612	675	611	1898
		530d	1250	1403	1522	4175
		X5	902	941	951	2794
	BMW Total		4908	5509	5775	16192
	Mercedes	A170	662	740	777	2179
		C220	2072	2345	2571	6988
		C270	1553	1700	1962	5215
		SKL500	89	99	104	292
		SLK500	421	490	515	1426
	Mercedes Total		4797	5374	5929	16100
Espanha Total			9705	10883	11704	32292
Portugal	BMW	320d	156	186	202	544
		525d	46	60	63	169
		530d	8	9	10	27
		X5	49	61	62	172
	BMW Total		259	316	337	912
	Mercedes	A170	91	97	104	292
		C220	223	256	258	737
		C270	146	164	178	488
		CLK500	46	53	60	159
	Mercedes Total		506	570	600	1676
Portugal Total			765	886	937	2588
Grand Total			10470	11769	12641	34880

Figura 13. Resultado de uma consulta multidimensional do exemplo

Prosseguindo com a exploração do cubo, a consulta será formulada através da manipulação da estrutura numa perspectiva multidimensional, por exemplo, trocando as medidas resultado pelas dimensões resultado ou mesmo modificando a selecção de uma "dimensão selecção". Veja-se, por exemplo, o seguinte caso: o analista, depois de conhecer o número de reparações num determinado país, marca e mês, pode querer aumentar o detalhe do local onde as reparações foram efectuadas. Na prática, isto é possível através de uma operação OLAP do tipo *drill-down*[9] sobre a dimensão país, de modo a detalhar a agregação ao nível da região ou

[9] ***Drill-down*** refere-se a uma das operações mais frequentes efectuadas sobre estruturas multidimensionais de dados em ambientes OLAP, que permitem visualizar os dados multidimensionais em diferentes níveis de agregação (detalhe). Concretamente, indica que a navegação nas agregações possíveis do cubo está a ser conduzida do mais genérico (*up*) para o mais detalhado (*down*).

mesmo do local da oficina. De forma oposta, a operação que corresponde à procura de dados mais genéricos denomina-se por *roll-up*, como já foi mencionado anteriormente. Todas estas operações OLAP devem ser de rápida execução, de modo a responder eficazmente ao utilizador que está a interagir com o sistema de processamento analítico. Recorde-se que todas estas operações actuam sobre o *lattice* multidimensional, em que cada subcubo corresponde a um grau diferente de sumarização, em função de uma determinada análise multidimensional. Saltam à vista os conceitos de generalização e detalhe dos dados no *lattice*. Sendo o primeiro o oposto do segundo, refere-se ao processo que abstrai um grande número de dados significativos de uma base de dados multidimensional, movendo-os de um nível conceptual baixo para um nível conceptual alto. Na realidade, os analistas gostam de ter a flexibilidade e a facilidade de dispor grandes conjuntos de dados sumarizados, de forma concisa e sucinta, em diferentes níveis de granularidade e a partir de diferentes ângulos. Esses dados ajudam a fornecer uma visão global sobre o domínio em análise. Os sistemas de processamento analítico permitem efectuar a generalização dos dados, resumindo-os em diferentes níveis de abstracção. Apresentam-se de seguida algumas das operações OLAP mais usuais realizadas para explorar os diferentes níveis de granularidade, representativos dos subcubos das dimensões país, fabricante e mês. Na figura 14, exibem-se as operações de *roll-up* e *drill-down*:

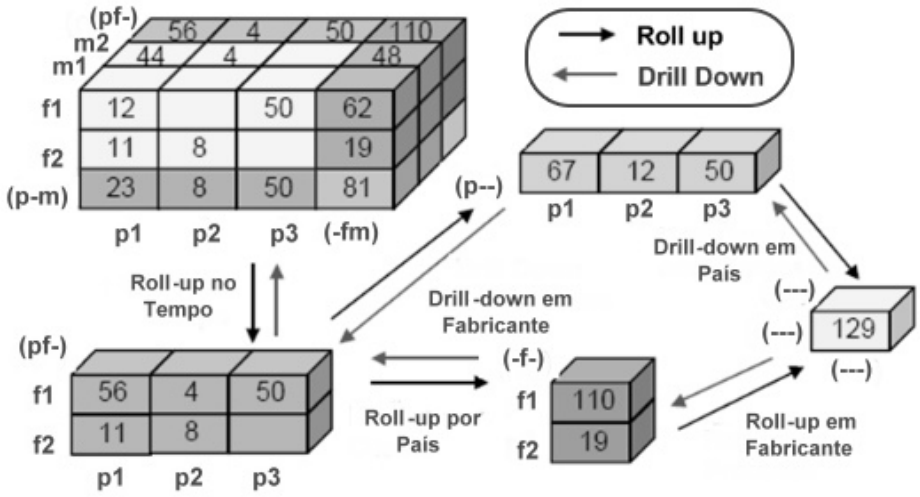


Figura 14. Representação das operações de roll-up e drill-down sobre o cubo (pfm)

De facto, o diálogo entre o agente de decisão e o sistema OLAP decorre de uma forma muito interactiva, e pode ser descrita como uma sequencia iterativa do tipo "uma resposta leva a uma nova questão". Esta situação provoca que o decisor seja prejudicado a nível da sua produtividade, pela demora na obtenção da resposta desejada. No limite, para além de se assistir a um decréscimo da produtividade (prejudicial ao negócio), podem ainda ocorrer dois efeitos paralelos: a intercalação de outras tarefas com as operações de consulta e a consequente diminuição da centralização e capacidade de gerar novas hipóteses, o que pode causar, eventualmente, o desuso do sistema de processamento analítico. Na figura 15, mostram-se as operações de *dice*[10]*, slice* e *pivot*[11]:

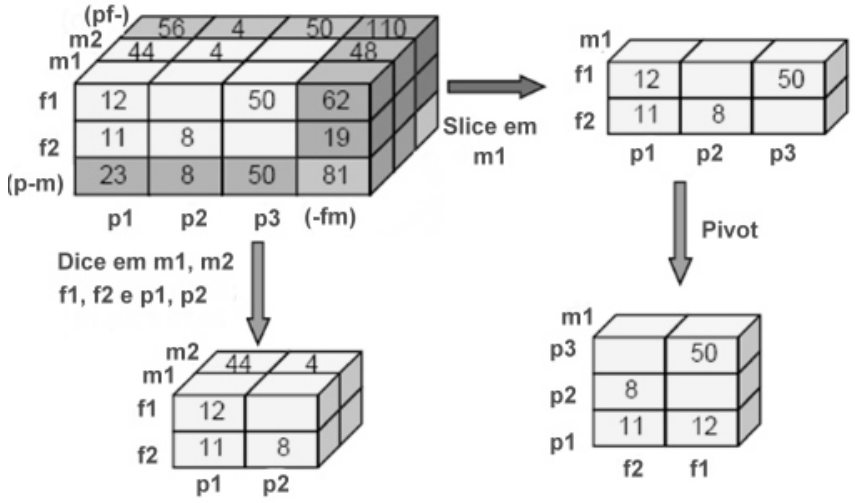


Figura 15. Representação das operações dice, slice e pivot sobre o cubo (pfm)

2.5. Posicionamento do Processamento Analítico no Mundo Real

A capacidade em explorar os dados correspondentes ao negócio possibilita aos decisores a capacidade de saberem o que os dados na realidade significam. Os exemplos que têm sido

[10] ***Dice*** refere-se a uma das operações comuns efectuadas sobre estruturas multidimensionais de dados em ambientes OLAP. Essencialmente, uma operação de *dice* corresponde a um *slice* aplicado a duas ou mais dimensões do cubo (ou mais de dois *slices* consecutivos).

[11] ***Pivot*** refere-se a uma operação comum, tipicamente OLAP, que permite alterar a orientação dimensional das perspectivas em análise.

demonstrados nas secções anteriores reflectem um desses casos, mas não são de maneira nenhuma, os únicos em que os sistemas OLAP podem ser utilizados. Os sistemas de informação, geralmente, podem ter um grande impacto em todo o sistema de valor e criação de negócio de uma organização. Podem actuar, por exemplo, na melhoria dos processos de negócio a todos os níveis, relacionando-os com todos os outros intervenientes envolvidos na cadeia de valor da organização (fala-se, neste caso, em sistemas operacionais extensíveis aos níveis de gestão e estratégicos). É aqui que poderão actuar os sistema de processamento analítico. Veja-se, a análise de comportamento de um conjunto de requisitos com métricas associadas, cuja satisfação assegure o sucesso do negócio, têm uma intervenção proeminente de um sistema OLAP. De facto, a maioria das actividades no seio de uma organização podem beneficiar do conhecimento extraído dos sistemas de processamento analítico. A verdade é que este tipo de tecnologia permite tirar partido de uma visão multidimensional de agregação dos dados de negócio de modo a facilitar o rápido acesso à informação estratégica e de gestão para análise, agilizando todo o processo de tomada de decisão numa organização.

Os sistemas OLAP permitem aos analistas, gestores e executivos de extrair conhecimento dos dados através do acesso rápido, consistente e interactivo a uma vasta variedade de possíveis vistas sobre os mesmos. Transformam os dados base de modo a que estes possam reflectir a dimensionalidade real dos negócios e como estes são entendidos pelos agentes de decisão. Enquanto que os sistemas de processamento analítico têm a capacidade de responder "quem?" e "o quê?", é a sua habilidade para responder "e se?" e "porquê?" que os distingue indiscutivelmente dos sistemas de *Data Warehouse* (apesar da sua complementaridade[12]). Os sistemas OLAP permitem tomar decisões projectadas para o futuro, com base em cálculos mais complexos, do que simples operações aritméticas, com por exemplo, a resposta a esta interrogação de negócio: "Qual será o efeito do custo dos refrigerantes para os distribuidores se o preço do xarope de fruta subir 0,30€/litro e os custos de transporte diminuírem 0,55€ por

[12] Os sistemas de *Data Warehousing* gerem e armazenam dados numa perspectiva multidimensional. Os sistemas OLAP transformam esses dados em informação estratégica. À medida que os agentes de decisão executam consultas mais complexas, avançam no processo de gestão de conhecimento (dados » informação » conhecimento).

quilómetro?". Na prática, os sistemas de processamento analítico abrangem um vasto domínio de funções organizacionais. Por exemplo, o departamento financeiro usa OLAP em acções de orçamentação, atribuição e custo de actividades, análise e modelação do desempenho financeiro, etc. No departamento de vendas, as actividades de previsão e análise de vendas são também asseguradas por este tipo de sistemas. Entre outras aplicações, no domínio do Marketing, os sistemas de processamento analítico são usados para análises de mercado, novas oportunidades de negócio, previsão de vendas, análise de promoções e campanhas, análise e segmentação de clientes, etc. A nível industrial, os benefícios de um sistema OLAP são visíveis, por exemplo, no planeamento da produção e na análise de defeitos.

De facto, são inúmeras as vantagens que advêm do uso de um sistemas de processamento analítico, mas a sua característica mais distintiva é o facto de permitirem dar aos gestores as informações que necessitam para tomar decisões eficazes, conforme necessário e em tempo útil, enquadradas no panorama de evolução estratégica da organização. Nas secções anteriores deste trabalho, já foi possível discutir que a entrega *just-in-time* da informação levanta outras questões que envolvem a complexidade do cálculo das agregações, o tempo de processamento e o espaço de armazenamento, para alem de ser necessário mais do que um nível base dos dados detalhados. Adicionalmente e, porque a natureza das relações das agregações pode não ser conhecida, o modelo de dados deve ser flexível, de modo a assegurar que o sistema de processamento analítico responda eficazmente as constantes alterações dos requisitos de negócio. Estas questões levam a outras relacionadas com o processamento eficiente das interrogações OLAP, que envolvem a determinação das operações mais comuns efectuadas sobre os subcubos e selecção materializada dos mais relevantes para responder a essas operações. Este e outros assuntos serão detalhados nas secções seguintes deste trabalho.

Muitos outros exemplos poderiam ser acrescentados, já que o processamento analítico é parte integrante, fundamental, da designação mais abrangente de *Business Intelligence*. Esta classe de sistemas orientados para a gestão define-se como uma categoria ampla de aplicações e tecnologias capazes de permitir a obtenção do entendimento acerca do negócio ou organização, com vista à compreensão dos dados gerados nas suas actividades e dai extrair

conhecimento útil que permita aos decisores tomar melhores decisões. A designação implica um conhecimento de todos os factores que afectam o negócio, sendo imperativo que o decisor tenha um profundo saber sobre as actividades de clientes, cadeia de fornecedores, competidores, parceiros de negócio, circunstâncias económicas e operações internas. Atente-se de que forma é que os sistemas de processamento analítico podem auxiliar em cada um destes domínios. Sobre os clientes, as interrogações analíticas permite efectuar análises sobre os seus gostos, de modo a adaptar o negócio a cada um deles, para que a organização possa oferecer um serviço único, de qualidade e personalizado. Isso permite, antecipadamente, conhecer também as motivações de cada cliente a um nível detalhado, tornando-se útil para desenvolver produtos ou serviços inovadores e à imagem de cada cliente, com um factor de sucesso de vendas muito alto. Quanto aos competidores, há que ter em consideração que os objectivos serão muito semelhantes aos da organização: satisfação dos clientes, com consequente maximização do lucro. Para que a empresa possa estar um passo sempre à frente da concorrência, é necessário conhecer de forma antecipada os movimentos e acções que os competidores estão a preparar, a fim de preparar uma tomada de decisão mais informada e inovadora, que permita ganhar vantagem. A nível de parceiros de negócio, estes devem ter também acesso a determinada informação estratégica, para minimizar erros de comunicação com as organizações e colaborar na melhoria da sua gestão.

Sobre o ambiente organizacional, é importante saber qual o impacto que os sistemas de processamento analítico podem ter. Note-se que o estado da economia e dos seus factores chave são reflexões importantes a ter em conta no processo de tomada de decisão. Como já foi mencionado no decorrer deste trabalho, o processamento analítico pode ajudar, de facto, nestas situações. Sendo o tempo uma dimensão de utilização constante, a visualização da evolução temporal de muitos factores chave de negócio (e sua inspecção em diferentes níveis de abstracção) mostram aspectos que podem ser fulcrais na definição estratégica de uma organização. Por último, mas não menos importante, existe uma situação em que os sistemas de processamento analítico têm uma maior aplicação: na análise das operações internas de uma organização. Estas referem-se a tudo o que são dados gerados pelas actividades diárias

realizadas, nos quais os sistemas de processamento analítico poderão ser aplicados e adequados a uma multiplicidade de processos existentes.

2.6. Materialização de Vistas em Cubos de Dados

Nas secções anteriores, já foi discutido, por diversas vezes, o problema da necessidade de materialização do cubo, sendo irrealista o processamento de todos os cubóides que podem ser gerados dentro de um cubo de dados (ou através das relações base) em situações reais. Nesta secção será aprofundado este tema. A materialização de vistas é uma técnica de optimização que visa melhorar o desempenho do processamento das interrogação multidimensionais, solicitadas ao sistema OLAP, pelos agentes de decisão. Baseia-se no conceito de previsão do cálculo (total ou parcial) das interrogações, de modo a diminuir o impacto que os seus tempos de resposta poderão ter no desempenho do sistema de processamento analítico. A optimização das interrogações efectuadas utilizando a técnica de materialização de subcubos divide-se, essencialmente, em três áreas de actuação: a selecção de vistas a materializar; a utilização das vistas materializadas e a sua manutenção (actualização) [Gupta, 1995], [Chirkova et al., 2001]. Antes de analisar cada uma das áreas, veja-se as opções a ter em consideração para materialização de vistas, dado um cubo de dados, tendo em consideração que a escolha deve incidir sempre sobre o *trade-off* espaço e tempo:

- **Sem materialização** – Não existe o pré-processamento de nenhum dos subcubos de dados. Esta situação conduz a um elevado custo no processamento das agregações multidimensionais, e pode tornar-se extremamente lento.

- **Materialização integral** – É efectuado o pré-processamento de todos os subcubos. O *lattice* de dependências de cada subcubo calculado traduz-se no processamento global do cubo de dados. Esta opção tipicamente conduz à necessidade de grandes quantidades de espaço de armazenamento, já que é indispensável guardar todos os dados materializados.

- **Materialização parcial** – É efectuado o pré-processamento apenas de um conjunto seleccionado de subcubos considerados mais adequados, do universo de cubóides disponíveis. Em alternativa, é possível também calcular um subconjunto do cubo que contenha apenas as células que satisfazem um determinado critério especificado pelo utilizador, tal como definir um limite mínimo para se considerar a contagem dos tuplos do cubóide (designados por *iceberg cubes* [Fang et al., 1997], o tema central deste trabalho, a ver em detalhe no capítulo 3)**.** A materialização parcial representa uma harmonização interessante entre o tempo de resposta e o espaço ocupado em memória, pelo que os tópicos seguintes serão baseados nesta selecção.

Materializar uma vista sobre uma estrutura de dados multidimensional significa armazenar os tuplos resultantes do processamento da sua geração, numa tabela. A esta tabela dá-se o nome de vista materializada, ou seja, tornar reais as agregações de um determinado cubóide resultantes das interrogações efectuadas ao *lattice* de dependências de um cubo de dados. Em princípio, se o subcubo de resposta a uma determinada interrogação for previamente armazenado numa tabela, à partida, a disponibilização do resultado ficará somente condicionado ao período de leitura dos tuplos da mesma. Por isso, qualquer cenário cuja janela de execução seja limitada, pode ter ganhos consideráveis ao utilizarem-se estas estruturas. O mesmo acontece quando são processadas interrogações complexas que envolvem grandes quantidades de dados. Isto leva a querer que o ideal seria mesmo guardar as respostas para todas as interrogações possíveis, contudo, essa acção levanta dois problemas fundamentais, já falados anteriormente: o espaço em disco e o tempo de processamento. Neste contexto, surge um dos desafios que se coloca à utilização de vistas materializadas: Quais as vistas mais indicadas para serem materializadas tendo em conta a minimização do tempo de processamento das interrogações, dadas determinadas restrições de espaço e tempo? (referindo-se ao problema de selecção de vistas materializadas, que será objecto de discussão mais detalhada na próxima secção deste trabalho).

A utilização de vistas materializadas é outra área de investigação que importa referir. Os decisores ao analisarem os dados, lançam interrogações escritas em termos de base de dados,

isto porque desconhecem a existência de vistas materializadas ou porque o dinamismo do sistema pode obrigar a uma constante actualização do conhecimento relativamente ao conjunto de vistas em utilização. O principio fundamental é manter a utilização de vistas materializadas totalmente transparente para o utilizador, de modo a agilizar o seu processo de actualização e adaptação. Ou seja, as interrogações que são lançadas na base de dados têm de ser reformuladas em termos de vistas disponíveis sempre que tal seja possível de modo a contribuir para a optimização do desempenho do sistema de processamento analítico. À parte da sua utilização, outro domínio de extrema importância tem de ser considerado: a sua manutenção. Veja-se o seguinte: as tabelas base, sobre as quais se definem as vistas, sofrem várias alterações (inserções, actualizações, remoções, etc.) ao longo do tempo. Qualquer alteração nos dados base deve ser reflectida nas vistas materializadas de maneira a manter a sua consistência com as tabelas base. A incoerência dos dados entre estas estruturas resultam num custo elevado no processamento das interrogações. Isto porque a resposta a uma determina consulta tem de ser a mesma (sejam as interrogações efectuadas no esquema da base de dados ou sobre vistas materializadas).

Na verdade, a optimização das interrogações lançadas no sistema de processamento analítico recorrendo à técnica de materialização de vistas (ou subcubos) deve ser considerada como uma mais-valia, na medida em que pode originar ganhos consideráveis nos tempos de resposta das interrogações e, com isso, melhorar consideravelmente o desempenho global do sistema analítico. Contudo, é uma prática que consome certos recursos em termos de espaço de armazenamento e tempo de cálculo. Por isso, torna-se necessário encontrar um ponto de equilíbrio que coadune os ganhos e os custos de utilização, não esquecendo que é fundamental considerar as actualizações que vão ocorrendo ao longo do tempo, já que as relações base são alteradas em resultado das transacções e eventos ocorridos no mundo real, sendo necessário que essas modificações sejam reflectidas por toda a hierarquia das agregações, de forma a manter-se a coerência dos dados e evitar-se a utilização de informação desactualizada. Técnicas de paralelismo e actualizações incrementais devem ser exploradas na reconciliação de dados [Gupta, A. et al., 1993], [Griffin & Likkin, 1995], [Zhuge et al., 1995].

Veja-se agora a natureza dos custos envolvidos na materialização de vistas, já mencionados nesta secção: o espaço para a sua efectivação; o tempo de materialização (inicial ou reconstrução); o tempo de actualização (refrescamento). O primeiro é de natureza monótona: mais agregações implicam sempre um maior espaço de armazenamento. Cada nova alteração implica não só novos tuplos nas relações base, mas também muitos outros nos subcubos materializados, já que a dimensão tempo tem um carácter eminente, excepto se a granularidade não for considerada muito pequena (ex. mês). Os restantes custos são não monótonos [Bauer & Lehner, 2003], ou seja, a adição de uma nova agregação a um conjunto X prévio pode significar uma diminuição dos custos. Suponha-se a situação mostrada na figura 16. Para simplificar, considere-se que a manutenção é efectuada por geração integral dos subcubos a materializar. Atente no cálculo do custo de cada distribuição X e X' (que correspondem aos cenários 1 e 2, respectivamente). O custo de materialização de X é de: 18000 (subcubo 1) + 3 x 6000 (subcubos 2, 3 e 4) + 12 (subcubo 5) = 36012; enquanto que o custo de materialização X' é de: 18000 (subcubo 1) + 2 x 6000 (subcubos 2 e 4) + 2 x 600 (subcubos 3 e 5) + 12 (subcubo 6) = 31212 é menor, apesar de X'=X+(p-m).

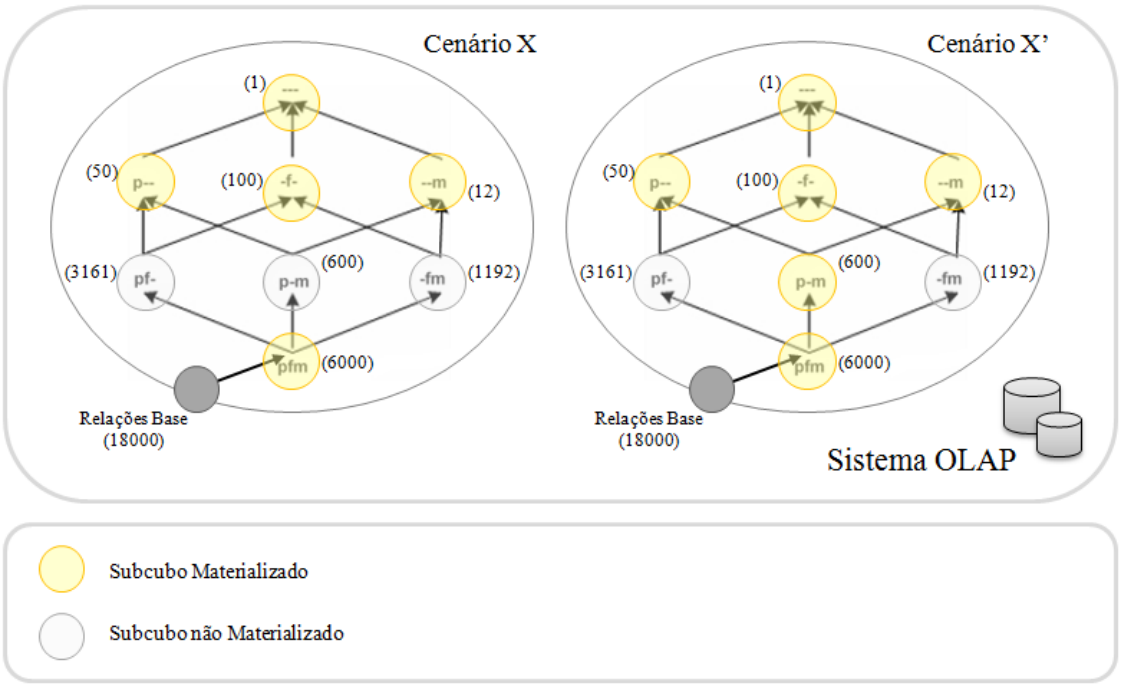


Figura 16. Cálculo dos custos de manutenção das duas distribuições X de subcubos

É visível neste exemplo que o custo de materialização é nitidamente não monótono. Essa situação é equivalente ao custo de manutenção por reconstrução dos subcubos a materializar, denominada manutenção integral, onde se podem aplicar as técnicas de actualização ou manutenção incremental (através da identificação dos deltas [Mumick et al., 1997]) para alteração das vistas materializadas, sempre que as relações base são modificadas, fruto da actualização dos sistemas transaccionais ou outras fontes externas, decorrentes dos eventos ocorridos no ambiente de negócio. Em regra, os custos de manutenção incremental são menores do que os custos de actualização integral dos subcubos. A razão é simples: as alterações têm um impacto limitado em muito dos subcubos materializados. Esta análise de custo benefício sobre os vários factores intervenientes, mostra que, de facto, o problema de materialização de vista é de natureza complexa. Resumindo: a natureza de *I* determina a utilidade das agregações do subcubo *M*; sendo C o custo, *Ci(I, M)* é monótono em *M*; *Cm(M)* é não monótono com *M*. Dessa forma, dados os custos e o seu comportamento, interessa na natureza de *I*, seleccionar as agregações *M* cuja existência se revelar ser mas benéfica, o denominado problema de selecção de vistas ou subcubos a materializar.

Para além da materialização de vistas, a existência de uma politica consolidada de utilização de estruturas auxiliares de indexação contribui também para um acesso eficiente aos dados armazenados nas estruturas multidimensionais [Roussopoulos, 1982], [Harinarayan et al., 1996], [Gupta, H. et al., 1997]. O processamento de *querys* OLAP pode, de facto, ser optimizado com a utilização de técnicas de indexação, das quais se distinguem: *bitmap indexing*, onde cada atributo tem a sua própria tabela de indexação e auxiliam na tradução e redução das operações do tipo *join*, comparação e agregação em lógica binária (0,1); *join indexing*, que permite registar as linhas de junção de duas ou mais relações de uma base de dados relacional, reduzindo o custo das operações de agregação OLAP; *bitmapped join indexing*, que combina os dois métodos anteriores de modo a acelerar, em alguns casos, ainda mais o processamento de *querys* OLAP [Valduriez, 1987], [O'Neil & Graefe, 1995]. Reconciliando a materialização de vistas com uma politica consistente de indexação, é possível aumentar o processamento das interrogações que são lançadas no cubo de dados. Efectuada a materialização dos subcubos, o processamento das interrogações deve ser sempre

efectuado tendo em consideração as operações OLAP que vão ser efectuadas nos subcubos disponíveis (envolve a transformação das condições especificadas numa interrogação de selecção, projecção ou *group by* em operações do tipo OLAP) e a determinação de quais os subcubos materializados que são importantes para a concretização dessas operações (envolve a identificação dos todos os cubóides materializados, potenciais para responder às interrogações efectuadas, aplicando um filtro - *pruning* - a esse universo, utilizando o conhecimento de todas as relações dominantes do *lattice* de dependências). Desta forma, é possível estimar custos tendo em conta os restantes cubóides materializados e seleccionar aqueles cujo preço de cálculo seja menor.

2.7. Selecção Optimizada de Vistas Materializadas

A busca de soluções para o problema de selecção de subcubos a materializar deverá assumir um carácter necessariamente aproximado e simulado. Senão, veja-se o seguinte: este problema, já referenciado como sendo *NP-hard*, é de natureza combinatória, já que qualquer subcubo pode ser candidato ou não para ser materializado em *M*. Se o número de cubóides possíveis for muito elevado, a quantidade de agregações ainda é maior. A procura de soluções que envolvam a pesquisa exaustiva do espaço de soluções provocaria um número igual de iterações, cada uma sujeitada a um cálculo do custo. Por isso, qualquer processo de optimização deverá centrar-se na procura das zonas mais promissoras do espaço de soluções possíveis, para que o tempo de execução seja razoável. No entanto, essa situação necessita, geralmente, dum número elevado de iterações, o que introduz a questão do carácter simulado do cálculo de custos, a ver mais à frente. A materialização de vistas obriga ainda a ter em consideração os problemas decorrentes de incoerência de dados (quando se refere ao problema de manutenção de vistas) e a capacidade em tornar esta técnica, invisível para os utilizadores e aplicações informáticas, aquando a sua utilização.

A maior parte das abordagens ao problema de selecção de vistas a materializar incorrem através de algoritmos de pesquisa, cujo objectivo é procurar os subcubos mais adequados sobre um espaço de soluções possíveis. Na prática, procuram o conjunto optimizado de

soluções, tendo em consideração as interrogações mais frequentes e o *trade-off* entre espaço e tempo. Em [Zhang et al., 1999] os algoritmos (optimizados) de pesquisa podem ser de quatro tipos: 1) determinísticos, que constroem uma solução de um modo determinístico, seja por aplicação de heurísticas ou pesquisa exaustiva no espaço de soluções; 2) aleatórios, regidos por uma abordagem diferente: no inicio, é definido o conjunto de hipóteses de movimento no espaço de soluções do problema, de modo a criar arestas entre diferentes soluções no espaço disponível. Duas soluções relacionam-se por arestas somente se, poderem ser transformadas numa outra, apenas com uma hipótese de movimento. Na prática, este tipo de algoritmos percorrem aleatoriamente as soluções em função de certas regras e terminam quando não for possível mais movimentos ou se esgotar o tempo limite. O resultado será a melhor solução encontrada até esse ponto. A escolha das regras que determinam os movimentos têm um impacto significativo nestes algoritmos; 3) genéticos, algoritmos de pesquisa aleatória com comportamento semelhante à evolução biológica na procura da melhor solução. Parte-se de uma população inicial aleatória de modo a originar gerações através de cruzamentos e transformações. Somente os membros mais adaptados da população passam para próxima geração de acordo com uma função de custo. Estes algoritmos terminam quando deixam de haver melhorias significativas nas novas gerações, ou após um certo número de gerações. O melhor indivíduo encontrado na população final é a solução; 4) híbridos, que combinam a lógica dos algoritmos determinísticos e aleatórios puros: as soluções obtidas pelo primeiro são usadas como ponto de partida para o segundo ou como membros da população inicial para os algoritmos genéticos.

Antes de se enunciar o problema de selecção das vistas a materializar, a noção de AND-OR DAG introduzida por [Gupta, H., 1997] deve ser discutida de modo a clarificar os conceitos base que assentam num possível modelo de cálculo de custos, fundamental para essa escolha. Veja-se, em primeiro lugar, os conceitos de optimização de *querys* considerando uma operação de *join*. Considerando que uma base de dados D contém um conjunto de relações R_1, R_2, ..., R_m é definido um plano de processamento local para uma determinada interrogação Q. Um plano de execução é representado por um grafo que, neste caso, representa a árvore binária da operação e que evidência a relação de todos os seus ramos e nós. As arestas

assumem o predicado (mapeamento dos tuplos do produto cartesiano dos nós adjacentes para *{verdadeiro, falso}*, consoante se o tuplo for para ser considerado ou não no resultado) e selectividade da operação de junção (que retorna o rácio dos tuplos incluídos em função do número total). O espaço de pesquisa é o conjunto de todos os planos de processamento local. Porque a estrutura do processamento pode ser arbitrária, a cardinalidade do conjunto é muito maior. Um ponto no espaço de pesquisa representa um plano de execução específico. Cada ponto no espaço de pesquisa tem um custo associado. A função de custo mapeia a árvore de processamento e o seu custo. Já que há um conjunto alargado de métodos para efectuar uma operação de *join* (*nested loop, sort-merge, hash loop, etc.*) existem diferentes funções de custo relacionadas com a árvore de processamento respectiva. Na figura 17 é possível observar o custo de duas árvores que representam a operação *join* obtidas através de diferentes métodos (1). O custo de duas árvores em (2) são igualmente diferentes, mas partilham a mesma estrutura de operação:

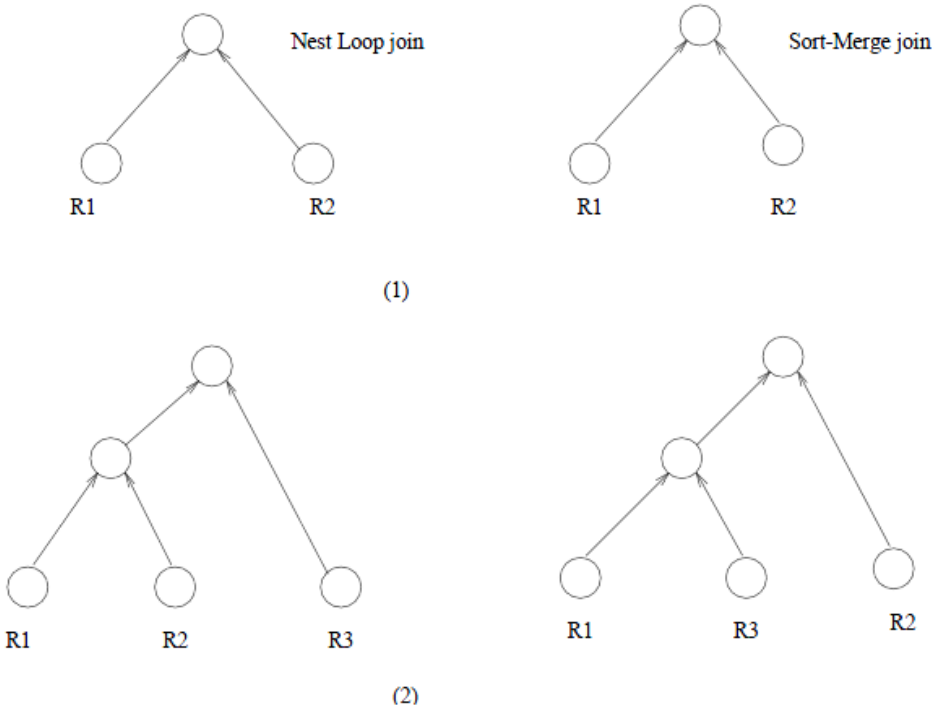


Figura 17. Ilustração dos planos de processamento de uma operação de junção

Dado um conjunto de planos de processamento para uma *query*, o objectivo para optimizar a sua execução passa por encontrar o plano, do conjunto disponível, com menor custo. Mas não é assim tão simples. Num sistema tipicamente analítico, um consulta envolve constantemente múltiplas interrogações, sendo por isso necessário considerar também a optimização do processamento de *querys* múltiplas [Timos, 1988], [Timos & Kyuseok, 1994]. Considere-se um conjunto de *querys* $Q = Q_1, Q_2, ..., Q_n$. Para cada interrogação Q_i existe, pelo menos, um plano de processamento local. Um plano múltiplo (ou global) pode ser construído através da selecção de um plano local de cada interrogação, para posteriormente, serem agregados num único. O plano local optimizado é referido como o plano de custo de processamento de cada *query* Q_i individualmente, enquanto que o plano global optimizado refere-se ao plano global de processamento, fruto da união das partes comuns dos planos locais individuais. O problema de optimização de querys múltiplas (MQO) é definido por: Dado N conjuntos de planos de processamento ($P_1, P_2, ..., P_n$) com $P_i = \{P_{i1}, P_{i2}, ...P_{iki}\}$ sendo o conjunto de planos possíveis para Q_i, $1 \leq i \leq n$, K_i é o número do plano de processamento local para Q_i. Deverá encontrar-se um plano de processamento múltiplo, através da selecção de um plano de cada P_i, de modo a que o custo da interrogação global seja minimizado. Geralmente, a união do plano local óptimo difere do plano global óptimo, portanto, não é exequível encontrar o plano global optimizado simplesmente por combinar os planos locais seleccionados. A combinação de planos de processamento múltiplos de interrogações (ex. várias árvores de uma instrução *join*) produz o plano de execução global da consulta, representado sob a forma de grafo acíclico dirigido (DAG).

Todos os planos de processamento possíveis para todas as interrogações formam um AND-DAG. Todos os planos de processamento juntos formam um AND-OR DAG. A junção de todos os AND-OR DAG individuais de cada interrogação permite criar o grafo dos possíveis caminhos para responder a essa mesma *query*, representados por um conjunto de nós equivalentes e operacionais. O grafo AND-OR global, por sua vez, representa todos os caminhos possíveis para responder a múltiplas *querys*. Um nó equivalente num DAG (*Eq node*) representa as classes de equivalência da expressão lógica que gera um resultado semelhante para responder à interrogação, cada um deles definido como um nó operacional

dependente do nó de equivalência e dos dados de *input*. Um nó de operação (*Op node*) corresponde a uma operação relacional algébrica (*join*, *select*, etc.). Representa a expressão definida pelo operando e pelos dados de *input*. Os nós dependentes de um *Eq node* são um ou mais *Op nodes* e os filhos de um *Op node* podem ser um ou dois *Eq nodes*. A existência de mais do que um filho em qualquer nó de equivalência indica um nó de operação. Cada nó de operação representa um AND. Um grafo AND-OR global é constituído pela união dos grafos AND-OR de cada *query*.

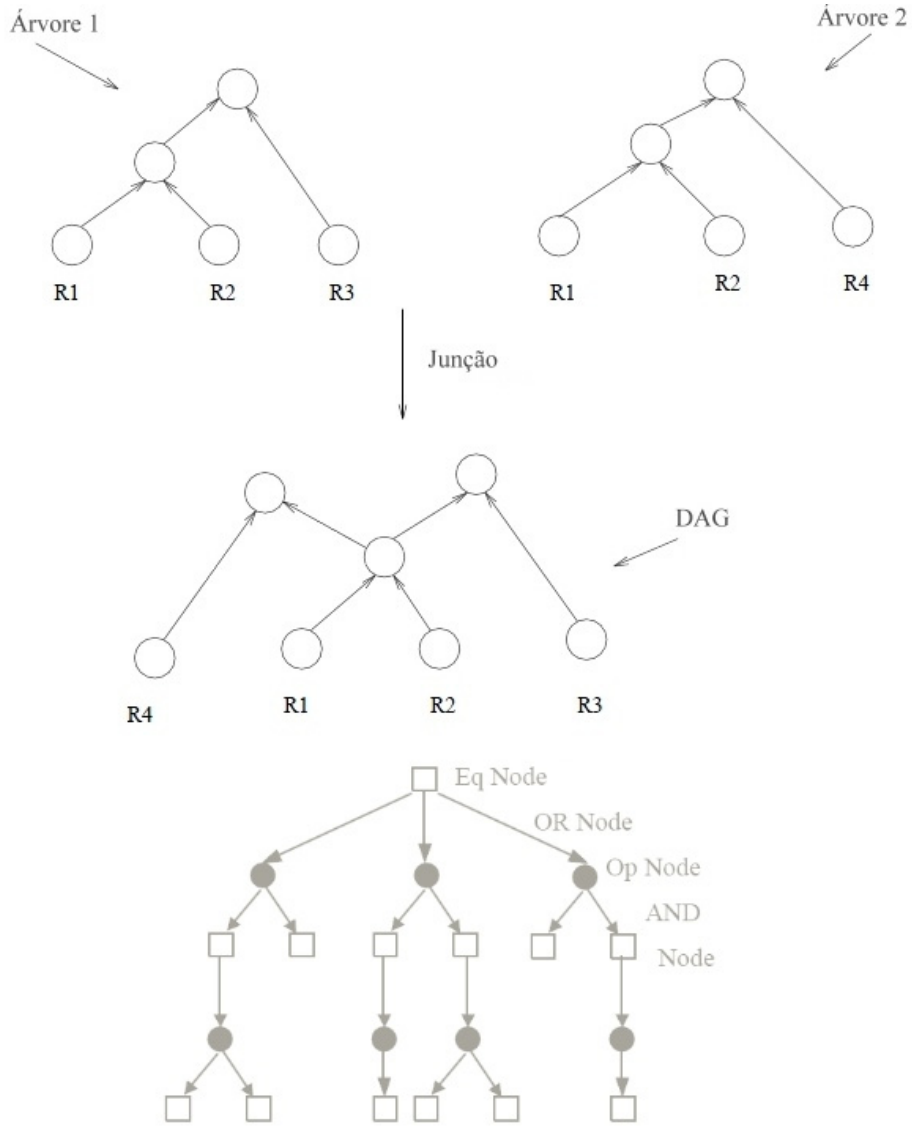


Figura 18. Um plano de processamento global (representação DAG)

Determinados os conceitos de optimização de *querys*, importa validar as variáveis a ter em consideração no problema de selecção de vistas (tendo em conta a minimização do custo do plano de execução global de uma consulta e manutenção dos subcubos em todos os nós). Genericamente, reflectem no seguinte: considere-se que *q(u, v)* refere-se ao custo de processamento da resposta a uma interrogação *u* utilizando uma vista materializada; v_x *q(u, v)* representa o somatório dos custos de processamento da *query* relacionados com as arestas do caminho mais curto de *v* a *u* mais o custo de selecção inicial do nó *v*, v_r. Se a vista *v* não consegue responder à interrogação *u* em *q(u, v)*, as tabelas base/relações base serão usadas em vez de *v*. De modo semelhante, *m(u,v)* refere-se ao custo de manutenção de uma vista materializada e reflecte o somatório dos custos de manutenção relacionados com as arestas do caminho mais curto de *v* a *u*. Através da relação destas dimensões é possível estabelecer um modelo de custos linear [Harinarayan et al., 1996] cujo algoritmo genérico deve ser adaptado de modo a seleccionar convenientemente as vistas a materializar em diferentes cenários. De facto, face à inexequibilidade de efectuar o cálculo experimental dos custos de materialização de forma real, a simulação torna-se a única forma viável, desde que exista um modelo adequado que permita emular o comportamento e as varáveis existentes no sistema real. Dessa forma, é possível deduzir fórmulas de cálculo de custos e, com elas, estimar os custos de simulações de *M* (soluções) propostas por qualquer algoritmos de optimização.

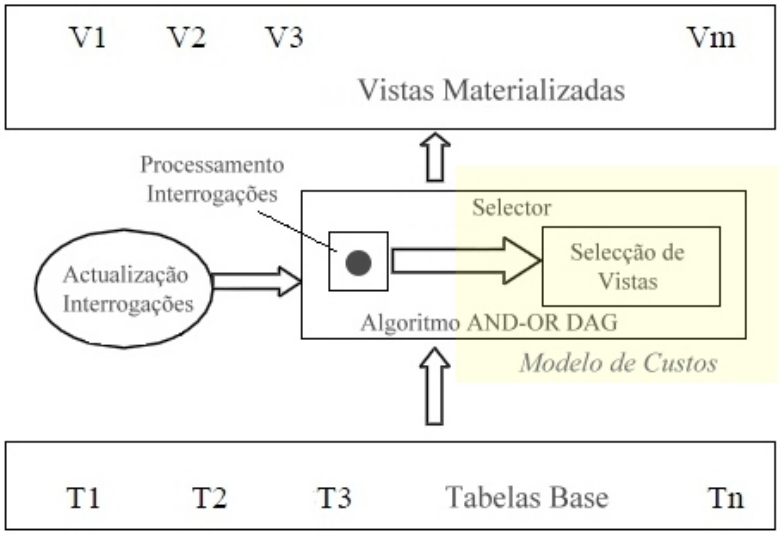


Figura 19. Processo genérico de selecção de vistas materializadas

Veja-se agora a seguinte definição do modelo de custos para o problema: Seja *L* (um DAG, por exemplo) o espaço de soluções e *M* ($M \subseteq L$) um conjunto de vistas a materializar, cada vista $v \in L$ têm três variáveis associadas: a frequência da interrogação f_v, a frequência de actualização g_v e o custo de leitura r_v. O custo de responder a uma interrogação *Q* (pertencente a uma vista *v*) corresponde ao tamanho de *v*, ou seja, o número de tuplos (r_v) [Kalnis & Papadias, 2001], [Theodoratos, 2004]. Pressupondo a existência de *L*, o custo de *M* é dado por $S(M) = \sum_{v \in M} rv$. Seja *u (v, M)* o custo de actualização da vista *v* quando *M* é o conjunto de vistas materializadas, então o custo de actualização de *M* é dado por $U(M) = \sum_{v \in M} gv \,.\, u(v, M)$. Seja *q(v, M)* o custo de responder a uma interrogação *v* quando *M* é o conjunto de vistas materializadas, então o custo total de responder a interrogações é dado por $Q(M) = \sum_{v \in L} fv \,.\, q(v, M)$. O problema de selecção de vistas a materializar consiste em escolher um conjunto *M* que minimize *Q(M)* e que respeite as restrições $S(M) \leq S_{max}$ e $U(M) \leq U_{max}$, sendo S_{max} o espaço de pesquisa disponível para materialização e U_{max} a janela temporal disponível para manutenção do conjunto de subcubos materializados.

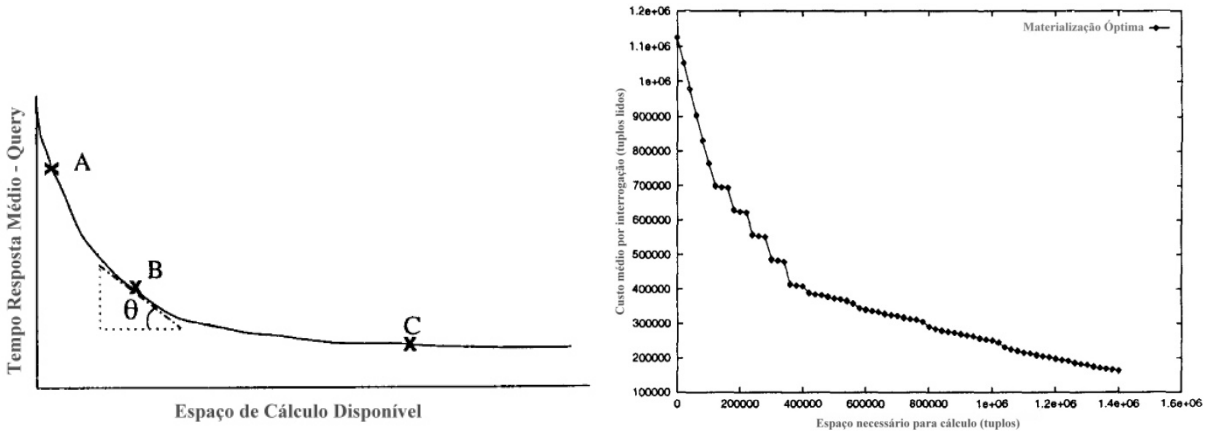


Figura 20. O custo de processamento de uma query em função do espaço [Shukla et al., 1998]

Repare-se que estas considerações têm um efeito imediato na optimização do processo de selecção de vistas materializadas, mas há que considerar também os constrangimentos decorrentes, normalmente, de determinadas imposições físicas, de carácter restritivo, face à solução proposta, como é o caso do espaço disponível e das restrições de processamento. Estas limitações incitam, geralmente: a recusa da solução faltosa (ou a sua inadmissibilidade),

a sua correcção e/ou imposição de uma penalização. A primeira refere-se à obrigatoriedade de validar cada solução escolhida (e consequente recusa ou substituição, no caso de ser uma não conformidade) ou então de garantir que o processo de pesquisa de soluções contenha um determinado algoritmo para evitar a geração de soluções inválidas. A segunda situação verifica-se caso seja detectada uma condição de não conformidade pela primeira. Se assim for, importa disponibilizar uma heurística que permita a eliminação de subcubos, até que a conformidade desejável seja atingida. Sobre a imposição de uma penalização, apenas é utilizada em técnicas de optimização onde se efectue a avaliação de uma função da elegância que determine esse mesmo processo (é o caso, por exemplo, dos algoritmos genéticos, uma abordagem para o problema de selecção de vistas materializadas, em que o processo de escolha é guiado pela adaptação ou elegância de cada solução: o custo). Se uma dada solução violar o constrangimento imposto, é-lhe simplesmente aplicada uma sanção (que representa um valor de custo extra) mas que irá prejudicar a sua probabilidade de selecção e consequentemente transmitir algumas das suas características para a próxima geração.

Como já foi discutido, o custo de responder a uma *query* (tempo de execução) é assumido ser igual ao número de tuplos das agregações usados para responder à interrogação. Portanto, a optimização do processo de selecção de uma vista agregada *v* é calculada em função da optimização do custo das interrogações para cada vista *w* (incluindo *v*) em vez de utilizar as relações base. Ou seja, em vez de aceder-se a uma tabela base, é possível reduzir o custo de processamento acedendo a uma vista materializada *v* de tamanho menor para cada *query w* que possa ser respondida por *v*. Em muitas aplicações reais, o custo de manutenção é o constrangimento que provavelmente mais dificulta o processo de reconciliação e garantia de consistência de dados entre as vistas materializadas e o *Data Warehouse* (em vez de restrições associadas ao espaço de armazenamento). O problema de manutenção de vistas materializadas parece, no entanto, ser muito parecido ao problema do espaço em disco. Contudo, o primeiro é de natureza mais complexa que o segundo. O custo de manutenção de vistas depende do relacionamento entre elas (a selecção de um subcubo irá afectar a relação com os subcubos materializados previamente). O custo total de manutenção de uma vista pode diminuir quando

mais vistas são materializadas, no entanto, o espaço ocupado pelo conjunto irá aumentar quando um novo subcubo é materializado (constrangimento do espaço).

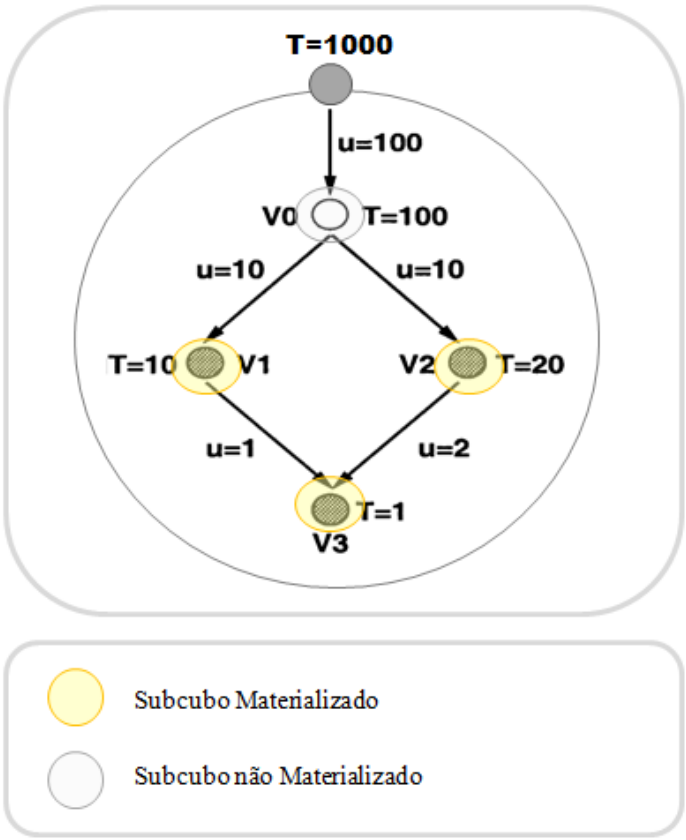


Figura 21. Representação do custo de manutenção de vistas

A figura 21 ilustra as diferenças entre os problemas de manutenção de vistas e espaço de armazenamento. Veja-se a seguinte notação: para um determinado *lattice* de correlações, em que v_i corresponde a um determinado subcubo, t e u representam o tamanho de uma tabela em que (r_{vi}) representa o custo de processamento de uma interrogação e ($wq_{u,v}$) o custo de manutenção respectivo. De modo a simplificar, assume-se que a frequência de uma determinada interrogação (f_v) e a frequência da sua actualização (g_v) são as mesmas para todos os subcubos evidenciados no exemplo. Então, suponha-se que $M = \{v_1, v_2, v_3\}$ são cubos materializados em ordem a v_3 e v_1 seguidos por v_2. O espaço total em disco usado é de $\sum_{v=1}^{+} t$, ou seja, $1 + 10 + 20 = 31$ e o custo total de manutenção é de $\sum_{v=1}^{+} u$, ou seja, 1 + 100 + 100 = 201. Repare que v_1 e v_2 necessitam de ser calculados a partir do subcubo virtual (não é nada

mais que as relações base do cubo, ou em última instancia, as tabelas base existentes no *Data Warehouse*) e v_3 é respondido por v_1 (daí o +1 no custo de manutenção). Considere-se agora materializar v_0. O espaço total em disco usado aumenta para 1 + 10 + 20 + 100 = 131 e o custo total de manutenção diminui para 1 + 10 + 10 + 100 = 121, porque v_1 e v_2 podem agora ser actualizados por v_0. A propriedade anti-monótona torna a manutenção de vistas complexa.

Para solucionar o problema do custo de manutenção de vistas materializadas, são propostos na literatura um número diversificado de algoritmos e soluções de optimização, que serão avaliados com mais detalhe na próxima secção deste trabalho. A título de exemplo, veja-se agora, o exemplo da aplicação de dois algoritmos *greedy*[13]: BPUS (*Benefit per unit of space*) proposto por [Gupta, H. et al., 1997], [Gupta, H., 1997] e PBS (*Pick by size*) sugerido em [Shukla et al., 1998]. O primeiro é dos mais discutidos e usados. Utiliza o conceito de DAG para representar as vistas e interrogações, sendo cada nó um gafo *G* que representa uma operação de agregação, selecção, projecção ou junção. A cada nó está associado um tamanho (número de tuplos) e uma frequência de interrogação (número de vezes que o nó foi acedido). O algoritmo constrói um conjunto de vistas *M* para serem materializadas num espaço limitado *L*, onde *A* representa todo o conjunto de vistas do *lattice* de correlações. Iniciada num conjunto vazio (inicialmente, *M* está vazio), a vista agregada *w* com maior benefício por unidade de espaço é seleccionada.

O algoritmo é executado de modo a escolher a vista com maior beneficio por unidade de espaço em cada iteração até que a condição de espaço seja atingida. A noção de benefícios de uma vista *w* em relação a um conjunto de vistas materializadas *M* foi uma das novidades apresentadas pelo autor e é calculado através da equação $B(w, M) = \tau(G, M) - \tau(G, M \cup \{w\})$.

[13] Os **Algoritmos *greedy*** referem-se a um modelo para resolver problemas de optimização, realizando sempre a escolha que parece ser a melhor no momento. Através da melhor escolha local efectuada, esperam alcançar a solução global considerara óptima para o problema em análise. Beneficiam por serem simples e de fácil implementação, contudo, nem sempre conduzem à solução óptima global, devido sobretudo à possibilidade de ocorrer processamento de duplicado nos cálculos e de não operarem extensivamente sobre todos os dados disponíveis.

O cálculo do benefício de uma vista consiste na diferença dos custos de processamento do conjunto de vistas (materializadas ou não). O custo total de um conjunto de vistas materializadas é dado por $\tau(G, M) = \sum_{wi \in G}^{k} fwi \,.\, q(wi, M)$. O benefício por unidade de espaço é calculado através do quociente entre o benefício apresentado pela vista *w* e o seu espaço *L(w)*: *BPUS(w, M) = B (w, M) / L(w)*. O benefício deste algoritmo é de pelo menos 63% (aproximado do óptimo). O tempo de execução é de $O(kn^2)$ onde o *k* é o número de vistas seleccionadas e o *n* o número total de vistas do *lattice* de correlações. Pelo facto do lattice de dependências de *D* dimensões ter 2^d vistas e $2^{(2\hat{}d)}$ conjuntos de vistas, *n* cresce exponencialmente com a dimensionalidade, tornando o BPUS um algoritmo extremamente lento para processar um grande número de dimensões - os *Data Warehouses* típicos têm, normalmente, mais de 10 dimensões. Grande parte do tempo gasto por este algoritmo está relacionado com a actualização do beneficio por unidade de espaço de todas as vistas, em cada iteração. O processo demora $O(n^2)$ para actualizar os BPUS das outras vistas, quando é seleccionada uma nova vista a materializar.

```
Algoritmo BPUS
ENQUANTO (ESPAÇO > 0) FAZ
        w = agregado com o máximo benefício
            por unidade de espaço em A
        SE (ESPAÇO - |w| > 0) ENTAO
                ESPAÇO = ESPAÇO - |w|
                S = S ∪ w
                A = A - w
        SE NÃO
                ESPAÇO = 0
```

```
Algoritmo PBS
ENQUANTO (ESPAÇO > 0) FAZ
        w = vista com menor dimensão em A
        SE (ESPAÇO - |w| > 0) ENTÃO
                ESPAÇO = ESPAÇO - |w|
                S = S ∪ w
                A = A - w
        SE NAO
                ESPAÇO = 0

(S) conjunto de agregados escolhidos
```

Figura 22. Algoritmos greedy de optimização de selecção de vistas, com restrição de espaço

O segundo algoritmo *greedy* enunciado para esta problemática denomina-se por PBS. Ao contrário do BPUS que considera o beneficio por unidade de espaço, o PBS apenas examina o tamanho das vistas que irão ser seleccionadas. O algoritmo selecciona as vistas, em tamanho crescente, até que seja atingido o limite de espaço. O PBS utiliza o mesmo conjunto de vistas que o BPUS, já que o *lattice* de correlações está dependente do espaço disponível. Nestas circunstâncias, uma vista *w* é, no mínimo, *k+1* vezes maior em relação ao maior subcubo dependente, onde *k* representa o número total de dependências de *w*. Esta regra garante que o

nó base de uma vista é sempre maior do que qualquer subcubo dependente (de outra forma, não era possível alcançar uma solução satisfatória). A complexidade temporal deste algoritmo é dada por *O(nlog)* onde *n* representa o número de cubóides existente no *lattice* de correlações. O tempo de execução é muito mais rápido em relação ao BPUS na medida em que não é necessário actualizar as vistas em todas as iterações. Necessita-se apenas de tempo para ordenar os cubóides do *lattice* por tamanho, sendo muito mais "barato" em relação ao tempo gasto pelo BPUS. Contudo, pressupõe que o *lattice* seja limitado a nível de espaço, o que pode não ser o mais comum na prática.

2.8. Descrição de Soluções Optimizadas para Exploração de Cubos

A optimização de estruturas multidimensionais conforme descritas nas secções anteriores permite traçar o objectivo, o modelo, a lógica de selecção e os constrangimentos. Mas ao efectuar-se uma caracterização plena das propostas de optimização para o problema de selecção de cubos, falta discutir também sobre duas dimensões ubíquas neste tipo de problemática: o tempo e o espaço. Reflicta-se sobre a primeira. Actualmente, o perfil das necessidades dos utilizadores altera-se constantemente, originando uma progressiva desactualização dos subcubos materializados e uma carga crescente das interrogações efectuadas. Por outro lado, já se verificou que a criação e actualização destas estruturas conduz a custos significativos, principalmente se for necessário reprocessar o cubo de dados extensivamente. Logo, importa perceber os impactos do tempo no processamento das estruturas multidimensionais. Se, por ventura, a adaptação do cubo for efectuada em intervalos crescentes, há que considerar que os custos de voltar a materializar os cubóides irão suceder em cada ajuste que se efectuar. Por outro lado, se utilizarem-se adaptações ditas instantâneas, pode-se estar a sobrecarregar o sistema com actualizações desnecessárias e que não correspondem às necessidades em longo prazo. Adicionalmente, uma terceira abordagem deve ser discutida, relacionada com soluções pró-activas ou preditivas, que pressupõem a existência de um componente especulativo, que vai procurar antever as necessidades futuras, adequando atempadamente os cubos a esse perfil de interrogações previsto.

Assim, na descrição das soluções de adaptação das estruturas multidimensionais de dados em função do carácter temporal do perfil de interrogações dos utilizadores, há que considerar uma abordagem tripla:

- As designadas **soluções estáticas** referindo-se àquelas cujo intervalo de readaptação do cubo de dados irá ocorrer de forma alargada. Neste trabalho, foram já demonstradas algumas destas soluções [Harinarayan et al., 1996], [Gupta & Mumick, 1999], [Shukla et al., 1998] cuja designação "estática" revela uma determinada estabilidade das estruturas nesta abordagem. Esta é a solução mais comum para tratar estruturas multidimensionais de maior dimensão, já que implicam, como foi referido, custos muito elevados. Podem também designar-se por soluções reactivas já que favorecem a reacção desfasada temporalmente em conformidade com às alterações dos perfil de utilização;

- As **soluções dinâmicas** com carácter quase instantâneo, que permitem acompanhar de perto as constantes alterações nos perfis de interrogação dos utilizadores. As soluções enunciadas em [Scheuermann et al., 1996], [Kotidis & Roussopoulos, 1999] são exemplos que reflectem esse comportamento. Podem ser designadas também por soluções activas já que se assiste a uma acção quase imediata face às necessidades. Este tipo de soluções só são exequíveis quando os custos de manutenção e readaptação do cubo são baixos;

- As **soluções pró-activas** enunciadas por exemplo na seguinte literatura [Belo, 2000], [Sapia, 2000] desempenham uma acção dupla: para além da sua instantaneidade, procuram posicionarem-se à frente no tempo (carácter preditivo) de modo a readaptar as estruturas multidimensionais de dados, tendo em conta as necessidades futuras dos utilizadores.

Sobre a segunda dimensão espaço, referido no inicio desta secção, é talvez a abordagem mais simples. As lógicas de selecção de subcubos ou vistas a materializar assume apenas duas

variantes: lógicas de selecção centralizadas ou lógicas de selecção distribuídas. A maioria das propostas enunciadas na literatura sobre o problema de selecção de vistas materializadas endereça uma abordagem centralizada num repositório de dados - o formato mais tradicional - um *lattice* cujo espaço de distribuição *S* é armazenado e mantido num único servidor de processamento analítico. As abordagens estáticas operam maioritariamente neste quadrante. Já a distribuição de estruturas multidimensionais pode ter algumas variantes, consequência das suas características. A abordagem activa e algumas pró-activas utilizam uma memória intermediária muito rápida (*cache*) para optimizar as operações de readaptação de cubóides, que podem residir em um ou mais servidores. Para todos os efeitos, sendo o *cache* um espaço de armazenamento, significa que irá verificar-se uma distribuição (ainda que limitada) das estruturas multidimensionais de dados. Contudo, esta situação pode ser estendida, considerando não uma, mas várias *caches* [Kalnis & Papadias, 2001]. Naturalmente, que a distribuição do processamento entre vários pontos implica considerar os custos de comunicação e a partilha das interrogações no modelo de custos [Stonebraker et al., 1994], [Kalnis & Papadias, 2001].

Contrariando a abordagem clássica, a materialização de vistas pode ser efectuada, portanto, não numa, mas em várias máquinas de processamento analítico (arquitectura M-OLAP multi-nó, a discutir na próxima secção), fazendo uso de soluções de bases de dados distribuídas e beneficiando do processamento sustentado, maior espaço de armazenamento, maior disponibilidade e custos controlados. Mas existe sempre um senão: se, de facto, uma arquitectura M-OLAP beneficia das vantagens da distribuição do processamento analítico, a complexidade da sua administração aumenta também. Existe, por isso, a necessidade de lidar com uma nova variante: o espaço. Se até agora, o clássico problema de selecção de vistas materializadas incidia sobre a escolha dos subcubos mais vantajosos, é preciso saber também materializá-los nos nós mais apropriados. No entanto, se existirem mecanismos e heurísticas para lidar com esta situação e forem considerados os custos de comunicação relativos às várias redes que interligam os diversos servidores analíticos e os valores da capacidade de processamento e armazenamento de cada nó, as vantagens poderão ser muitas e a distribuição poderá ser extensiva. Mencionadas as dimensões essenciais: tempo e espaço, importa agora

posicionar, em função de cada uma delas, as propostas de selecção de vistas materializadas segundo as lógicas de selecção (algoritmos) correspondentes. Naturalmente, que não se pretende mostrar todas as soluções existentes, mas sim efectuar um *survey* das mais representativas, quer pela sua importância ou pelo grau de inovação que trouxeram para a comunidade cientifica.

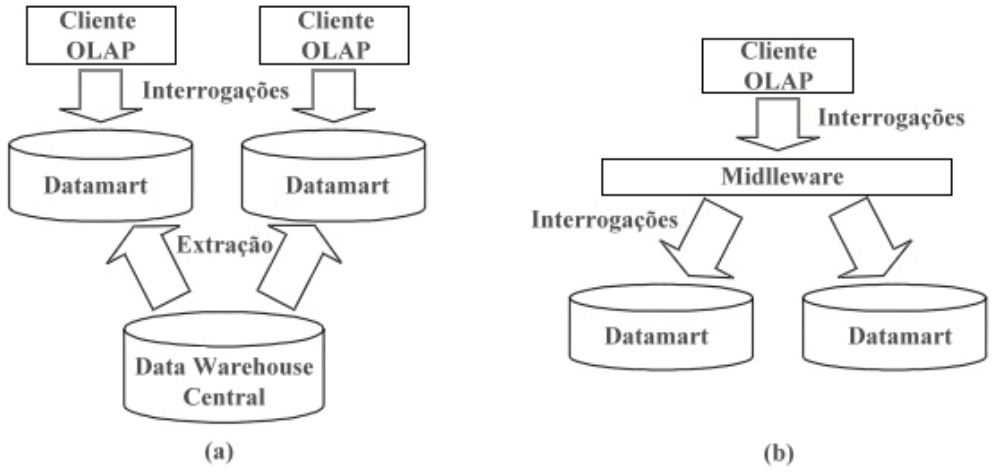


Figura 23. Arquitecturas de processamento analítico, sendo a) centralizada e b) distribuída

Repare-se que, quanto ao espaço, existe na literatura um grande número de abordagens centralizadas. Contudo, devem ainda ser consideradas aquelas cujo processamento analítico é efectuado de forma distribuída (ou descentralizada) e dispersa (ou fortemente distribuída). Sobre as lógicas de selecção de vistas materializadas, para além dos algoritmos *greedy*, há que considerar as propostas baseadas em heurísticas específicas, os algoritmos evolucionários, de pesquisa aleatória e por *simulated annealing*[14]. Quanto ao tempo, apesar das soluções ditas estáticas terem um predomínio consistente na literatura, as abordagens dinâmicas são também proeminentes no problema de selecção de vistas materializadas. Veja-se, então em primeiro lugar, a relação entre a dimensão espaço e os algoritmos de selecção de vistas materializadas

[14] ***Simulated annealing*** é uma meta-heurística probabilista genérica vocacionada para a resolução de problemas de optimização. Esta heurística procura localizar uma boa aproximação do mínimo global de uma dada função, num grande espaço de pesquisa de soluções. Em certos problemas, esta técnica mostra-se mais eficaz do que a enumeração exaustiva: o seu objectivo é apenas o de encontrar uma boa solução aceitável, num intervalo de tempo fixo, em vez de tentar obter a melhor solução possível.

existentes na literatura. O primeiro estudo sobre a problemática de selecção de vistas para suportar o processamento analítico dos dados foi anunciado em [Harinarayan et al., 1996] com a proposta de um algoritmo *greedy*. Posteriormente, propõe uma extensão a esse algoritmo, considerando também o processo de optimização e selecção de índices OLAP [Gupta et al., 1997]. Considerando as dificuldades dos algoritmos *greedy* face à evolução do crescimento dos sistemas de processamento analítico, [Baralis et al., 1997] considera o perfil de utilização na selecção de vistas materializadas, onde são escolhidas apenas as vistas consideradas relevantes para um determinado grupo de utilizadores, diminuindo o espaço de pesquisa de soluções e, consequentemente, o processamento integral do cubo. Uma outra melhoria é proposta por [Shukla et al., 1998] ao algoritmo *greedy* denominada por PBS (visto na secção anterior) na medida em que as vistas são seleccionadas em função do seu tamanho (não considerando o espaço de armazenamento). Esta mostra-se bastante mais eficiente, mas mantendo a qualidade das anteriores.

Paralelamente, é indicado um novo algoritmo em [Soutyna & Fotouhi, 1997] baseado em programação dinâmica que retorna o conjunto óptimo de vistas a materializar, em função do espaço de pesquisa disponível e da frequência das interrogações. Mais tarde, em [Shukla et al., 2000] é apontado um algoritmo que pressupõe que o cálculo das agregações não é efectuado a partir de um único cubo, mas sim através de modelos de dados multi-cubo. Como já se viu, [Gupta, H., 1997] propõe também uma metodologia iterativa para representar os planos de processamento e execução de interrogações, tendo em conta o problema de selecção de vistas a materializar (demonstrados por um gafo AND-OR). Adicionalmente, são considerados os constrangimentos dos custos de manutenção e do espaço de armazenamento das vistas materializadas [Gupta & Mumick, 1999]. Outras abordagens que exploram indirectamente estas questões, se seguiram: Veja-se, por exemplo, o modelo proposto por [Yang et al., 1997] denominado por *Multiple View Processing Plan* (MVPP) que percorre as interrogações à procura de expressões comuns e reutilizáveis no problema de selecção de vistas materializadas ou a solução proposta em [Theodoratos & Selis, 1997] que indica que todas as interrogações colocadas pelos utilizadores devem ser respondidas somente por vistas materializadas. O problema do espaço não é considerado nesta abordagem e os algoritmos

propostos são de natureza extensiva ou heurística. Posteriormente [Theodoratos & Bouzeghoub, 2000] sugerem uma abordagem que agrega as questões levantadas até então. Os autores pretendem catalogar todos os aspectos no processo de selecção de vistas materializadas, desde a definição do modelo de custo, optimização de interrogações, selecção e manutenção de subcubos e indicação dos constrangimentos decorrentes. Mais tarde, é também apresentada uma solução para selecção de vistas materializadas assentes em base de dados multidimensionais [Jamil & Modica, 2001]. No campo dos algoritmos evolucionários, existe também abordagens centralizadas que fazem uso de algoritmos genéticos. Veja-se, por exemplo, os seguintes métodos: o caso proposto por [Horng et al., 1999] visa seleccionar um grupo de vistas a materializar, atendendo aos aspectos relacionados com custos das interrogações, constrangimentos tempo e espaço e manutenção de vistas. Adicionalmente, [Lin & Kuo, 2004] apresentam uma solução semelhante à anterior, mas com uma forma diferente de tratar as soluções inexequíveis. Ainda no domínio dos algoritmos evolucionários, [Zhang et al., 2001] propõem uma abordagem híbrida na medida em que usam uma combinação dos algoritmos genéticos com heurísticas, conseguindo-se obter assim uma diminuição da complexidade e do tempo de execução dos mesmos.

No domínio dos algoritmos aleatórios, destacam-se os algoritmos cujos resultados são graduais (pesquisa iterativa, por *simulated annealing* ou uma junção de ambos). Na literatura pode verificar-se que [Kalnis et al., 2002a], [Derakhshan et al., 2006] são bastante eficientes, especialmente indicados para utilização em *Data Warehouses* com elevada dimensionalidade, conseguindo obter uma qualidade próxima dos algoritmos *greedy*, num tempo de resposta muito inferior. Por último, importa discutir a caracterização espacial das soluções descentralizadas e dispersas. Veja-se os seguintes exemplos: em [Bauer & Lehner, 2003] é proposta a distribuição do cubo por vários nós, com possibilidade de replicação. É divulgado um *lattice* de correlações distribuído e uma extensão do modelo de custos linear. Esta solução enquadra um factor de relacionamento dos custos de comunicação e processamento e propõe a evolução do algoritmo *greedy* centralizado para um outro de carácter distribuído. [Zharkov, 2008] propõe outra solução distribuída, desta vez para materialização de vistas em sistema distribuídos de bases de dados em tempo real, tendo em conta os custos de processamento e

manutenção de vistas materializadas. Contempla também o problema do espaço de armazenamento e as operações de manipulação de cubóides. Ao efectuar-se o enquadramento genérico dos algoritmos de selecção de cubóides em função das suas características espaciais, importa agora analisar as principais abordagens existentes na literatura de acordo com a sua temporalidade, ou seja, em função do tempo que consomem para readaptar e manter as vistas materializadas. Visto que as abordagens estáticas conferem maioritariamente um espaço centralizado de operação, acabaram já por ser descritas nos parágrafos anteriores. Por isso, as descrições seguintes irão centrar-se sobretudo nas propostas dinâmicas e pró-activas. Uma das primeiras soluções dinâmicas a surgir foi anunciada por [Scheuermann et al., 1996] propondo um gestor de *cache* (*Watchman*) que guarda em memória o resultado das consultas e as interrogações geradas por essas. As consultas seguintes, se forem iguais, podem ser respondidas de imediato pela *cache*. Adicionalmente, podem ser usados um conjunto de algoritmos [Chen & Roussopoulos, 1994], [Gupta et al., 1995] para testar a paridade das consultas, de modo a alargar o número de soluções.

Em [Kotidis & Roussopoulos, 1999] é proposto outro gestor de *cache* (*Dynamat*) onde são guardados pedaços do cubo, resultantes das consultas agregadas, de granularidade menor em comparação com as vistas. O gestor funciona através de duas etapas: em primeiro lugar, verifica se existe um pedaço do cubo disponível em *cache* para responder a uma determinada interrogação e, assim, efectuar a melhor selecção de vistas materializadas em função do espaço disponível. Posteriormente, na fase de actualização, o gestor efectua uma triagem sobre o conjunto de vistas materializadas que melhor respondem ao problema, numa janela temporal disponível. [Karayannidis & Sellis, 2001] propõem um outro gestor (*Sisyphus*) para armazenar cubos de dados em *cache*, desta vez baseado na divisão do espaço em *chunks* [Desphand et al., 1998], [Zhao et al., 1997]. Ao entrar uma nova interrogação, o gestor calcula o conjunto de *chunks* precisos para responder a essa consulta. Esta sistema opera mais como um gestor de espaço para o problema de selecção de vistas, mas pode funcionar também como uma *cache*. Um terceiro gestor dinâmico de *cache* é indicado em [Shim et al., 1999] baseado em políticas de transferência de *caches* e gestão dinâmica do seu conteúdo, de modo a dar resposta não só às interrogações (cujo conteúdo exacto encontra-se armazenado em *cache*)

mas também fazendo uso de pedaços do cubo cuja correspondência não seja exacta. Mais tarde, [Kalnis & Papadias, 2001] sugerem uma arquitectura dinâmica, fortemente distribuída, denominada OCS (*OLAP Cache Server*) onde os servidores OLAP estão geograficamente dispersos e interligados através de uma rede *WAN*. Os utilizadores não acedem ao *Data Warehouse* directamente, mas sim a um OCS, que responde às interrogações se tiver a informação disponível em *cache* ou remete a consulta para os seus "vizinhos" remotos. Esta solução propicia uma maior disponibilidade dos dados OLAP e beneficia de uma escalabilidade sustentada, na medida em que novos OCS aumentam a capacidade de processamento e armazenamento. Esta arquitectura contribui para o domínio dos sistemas de processamento analítico a nível de sistemas de *cache* activos em redes Web, o que permite armazenar páginas OLAP dinâmicas com capacidades de manipulação de dados.

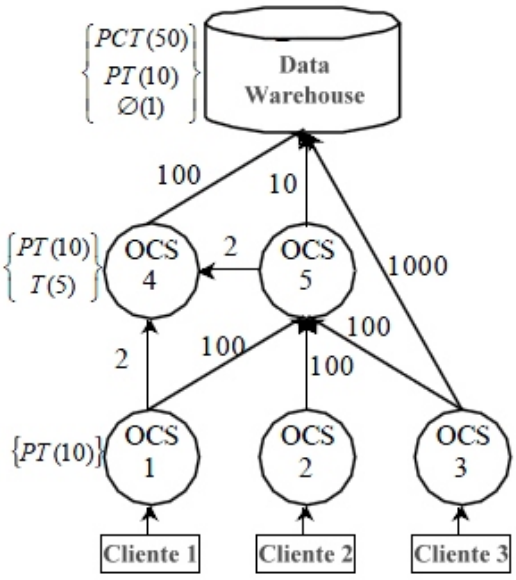


Figura 24. Uma rede de OLAP Cache Servers [Kalnis & Papadias, 2001]

Ainda no domínio das soluções dinâmicas em ambientes dispersos, importa referir a solução proposta em [Kalnis et al., 2002b] acerca de uma outra abordagem OLAP distribuída: *PeerOLAP*, cuja arquitectura é mostrada na figura 25, é constituída por diversos clientes usuais (PC's), cada um contento *cache* útil, interligados por uma *WAN*. Quando é lançada uma interrogação, se esta não puder ser respondida pelo terminal de origem (onde é lançada a consulta), é enviada para a rede até que seja encontrado um computador remoto (*peer*) que

tenha armazenado na sua *cache* a resposta pretendida. Veja-se agora algumas soluções pró-activas, de modo a finalizar as propostas de selecção e readaptação de vistas materializadas, em função da dimensão tempo, baseadas sobretudo na reestruturação dinâmica e preditiva de *caches*. A solução proposta em [Belo, 2000] descreve um sistema distribuído (assistente pessoal) com capacidade de adaptar dinamicamente as vistas materializadas mais utilizadas por um grupo de utilizadores que partilha os mesmos pedaços de um cubo. No essencial, a arquitectura contêm um motor de aprendizagem e inferência, responsável pela obtenção de conhecimento e selecção das regras que mais se adaptam às necessidades dos utilizadores, de modo a readaptar eficazmente as vistas requeridas e aceder aos dados em tempo útil. Para além de prever e propor aos utilizadores quais os próximos passos a realizar, este método beneficia sobretudo do conhecimento extraído do perfil de necessidades do grupo de utilização.

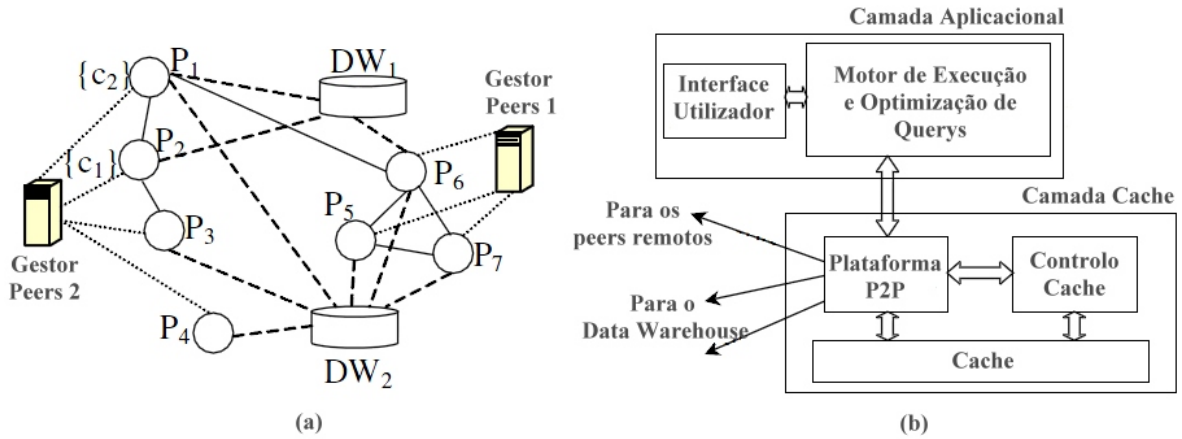


Figura 25. Arquitectura (a) PeerOLAP genérica e (b) detalhe de um peer [Kalnis et al., 2002b]

Outro sistema pró-activo foi abordado por [Sapia, 2000] denominado *Promise* (*Predicting User Behavior in Multidimensional Information Systems Environments*) que utiliza um mecanismo de previsão do comportamento dos utilizadores baseado num modelo *Markov*[15] de

[15] Em matemática, as **cadeias de Markov** representam um caso particular do processo estocástico (correspondente a uma família de variáveis aleatórias). O modelo define a seguinte propriedade: cada variável aleatória depende da precedente e, ao mesmo tempo, é condicionalmente independente das restantes, ou seja, os valores das v. a. anteriores são irrelevantes para a previsão dos valores seguintes, desde que o estado da v. a. actual seja conhecido. Os valores assumidos pelas variáveis denotam o estado do processo no tempo.

primeira ordem. Estabelece a viabilidade da solução efectuando, em primeiro lugar, a demonstração de que uma consulta tem grande probabilidade de estar próxima da anterior (ao analisar a carga das interrogações lançada no sistema OLAP) de modo a aplicar, posteriormente, o chamado "tempo de consideração", que possibilita a análise da resposta anterior até ser formulada nova interrogação (reestruturação). Um outro modelo pró-activo semelhante foi indicado por [Park et al., 2003] e que propõe a utilização de um gestor de *caches* que opera sobre os resultados das *querys* para a readaptação e cálculo de outras. Considera o beneficio não só das interrogações actuais, mas também dos pedidos que estão previstos para o futuro. O mecanismo preditivo proposto (*lowest usability first future*) baseia-se na exploração das correlações semânticas entre as diversas interrogações de uma instância OLAP, através de um modelo de probabilidades.

Efectuado o levantamento de algumas soluções citadas na literatura para o problema de selecção e manutenção de vistas materializadas, tendo em conta o seu posicionamento e relacionamento com as dimensões - espaço e tempo - importa agora, detalhar cada uma das lógicas de selecção que têm vindo a ser discutidas ao longo desta secção, de modo a completar o circulo: tempo » algoritmos de selecção « espaço. Veja-se, de imediato, o posicionamento do algoritmo de optimização *greedy*. Como se tem vindo a verificar, esta lógica de selecção tem sido usada amplamente num grande número de propostas. Logo em [Harinarayan et al., 1996] foi proposto o algoritmo GSC (*Greedy under Space Constraint*) que expunha as propriedades elementares da sua definição: ao iniciar-se um espaço de subcubos vazios, em cada iteração, será seleccionado aquele que for considerado mais benéfico face aos custos de processamento e tempo de resposta das interrogações. Outras propostas se seguiram, extensivas as propriedades elementares dos algoritmos *greedy*, alargando a sua aplicação e introduzindo novos constrangimentos (por exemplo, o tempo de manutenção de vistas explicado em [Gupta & Mumick, 1999]). Actualmente, destacam-se neste domínio, as seguintes abordagens: os algoritmos *greedy* por *inverted-tree* e a heurística A* [Gupta & Mumick, 1999], os algoritmos *greedy* por duas fases e integrados [Liang et al., 2001], [Yu et al., 2004] e os algoritmos *greedy* polinomiais [Nadeau & Teorey, 2004].

Os algoritmos genéticos ocupam também uma posição valiosa na problemática de selecção de vistas a materializar [Holland, 1992]. A sua génese está relacionada directamente com a forma de como se processa a evolução da "vida" de cada cromossoma num espaço de soluções complexo. Ao iniciarem com uma população aleatória (de cromossomas), geram em cada iteração uma árvore genealógica onde são seleccionados os "pais" e respectivos descendentes, utilizando processos idênticos aos biológicos (cruzamento e mutação). Posteriormente, todos os cromossomas são analisados utilizando uma função de adequabilidade para determinar a qualidade de cada solução, de modo a decidir quais aqueles que serão evoluídos ou eliminados do processo. Quanto mais adequados forem os cromossomas, mais probabilidades existem de estes serem seleccionados para gerar uma nova população. Este processo é repetido até ser encontrado o cromossoma de maior adequabilidade, em todas as populações geradas. Estes algoritmos têm vindo a demonstrar na literatura que, com um número reduzido de gerações, conseguem obter resultados tão bons ou melhores em comparação com os algoritmos *greedy* [Horng et al., 1999], [Zhang et al., 2001], [Lin & Kuo, 2004].

No domínio dos algoritmos aleatórios, as propostas evidenciadas na literatura procuram dar resposta ao tempo de resposta longo dos algoritmos *greedy*, cuja exequibilidade está dependente da dimensionalidade dos *Data Warehouses*. Como discutido anteriormente, em [Kalnis et al., 2002a] é proposta uma heurística que actua sobre um espaço de soluções, no qual são adicionadas vistas materializadas até que exista janela temporal disponível para o processamento. A escolha e omissão de vistas materializadas é efectuada recorrendo a três tipos de análise: pesquisa iterativa (desdobrando-se nos algoritmos de optimização *hill climber*[16] e *simulated annealing*) e pesquisa em duas fases, uma combinação das duas lógicas de selecção anteriores. Todas elas assentam num espaço centralizado, sendo a última, aquela

[16] Em ciências e tecnologias de informação, ***hill climber*** ou *hill climbing* refere-se a algoritmos de optimização de simples implementação, cujo modo de funcionamento inicia, essencialmente, através da selecção de uma solução aleatória (potencialmente pobre) de modo a incluir iterativamente pequenas modificações que vão melhorando aos poucos a sua confiabilidade. Quando o algoritmo não consegue detectar mais melhorias a incluir na solução, termina. Idealmente, neste ponto, a solução encontra-se próxima da óptima, mas não existem garantias que isso seja verdade, pelo que o seu recurso é popular apenas como "primeira escolha" na resolução de problemas de optimização.

que apresenta resultados semelhantes aos algoritmos *greedy* num tempo inferior de processamento. Outras soluções indirectas ao problema de selecção de vistas são também propostas, baseadas na optimização de algoritmos evolucionários. É o caso da solução indicada em [Kennedy & Eberhart, 1995] e [Eberhart & Kennedy 1995] que introduz o algoritmo PSO (*Particle Swarm Optimization*) ou enxame de partículas. Na sua essência, este algoritmo simula a capacidade de captar e processar conhecimento no reino animal das espécies mais evoluídas. Na pratica, dado um conjunto simplificado de agentes, as partículas, que andam dispersas pelo espaço multidimensional de pesquisa, correspondem a uma posição que representa uma solução para o problema. Cada partícula tem, em cada momento, uma colocação e uma velocidade, que é modificada pelo aceleramento causado por dois tipos de informações: a melhor colocação que conseguiu atingir e a melhor colocação atingida por uma qualquer partícula do enxame. Esta acção permite deslocar a partícula para locais onde se encontrem as melhores soluções para o problema. Existem duas versões do algoritmo PSO: a versão base, contínua (valores reais, em espaço contínuo) e a versão discreta (valores binários, em espaço discreto) proposta em [Kennedy & Eberhart, 1997]. As evoluções propostas para este algoritmo podem ser encontradas na literatura em [Parsopoulos & Vrahatis, 2005] e [Diosan & Oltean, 2006].

Uma outra solução evolucionária é indicada por [Maniezzo et al., 2001] que aborda o problema de optimização através da analogia com o comportamento de uma colónia de formigas, denominada ACO (*Ant Colony Optimization*) [Dorigo et al., 1996]. As formigas colocam uma matéria química nos locais onde passam. Esta acção incute a escolha que efectua do caminho a seguir (o caminho com maior probabilidade de ser escolhido é aquele que tiver maior quantidade de matéria química). A proposta reflecte sobre um problema mais pequeno na selecção de pedaços de vistas a materializar (fragmentação vertical) [Munneke et al., 1999] através da implementação de ANTS [Maniezzo, 1999]. Na sua essência, os algoritmos ACO operam sobre a pesquisa paralela efectuada por várias instâncias de processamento, baseadas num sistema dinâmico de memória com informações de todos os resultados obtidos de passos anteriores. Uma "formiga" é constituída como agente de processamento, que constrói iterativamente uma solução para resolver o problema. Os

algoritmos ACO inspirados na evolução da vida, tem, de facto atraído bastante atenção ao longo do tempo, na medida em que representam agentes biológicos que são "forçados" a desenvolver mecanismos sofisticados de modo a ultrapassar os vários problemas que vão encontrando. Várias melhorias têm sido propostas na literatura actual para aplicação de algoritmos ACO nos problemas de optimização complexos, com resultados interessantes [Maniezzo et al., 2004], [Blum & Dorigo, 2004] e [Rozin & Margaliot, 2007].

2.9. Evolução dos Sistemas de Processamento Analítico

O sucesso das soluções de processamento analítico têm levado a uma procura cada vez mais acentuada dos seus serviços: o volume de dados cresce a um passo acelerado, o número de utilizadores aumenta também, à medida que mais áreas de negócio são englobadas, o esforço reclamado a um sistema de processamento analítico é cada vez maior já que existem cada vez mais dimensões a processar e as interrogações são cada vez mais frequentes, complexas e variáveis. Torna-se necessário haver máquinas de processamento mais poderosas, o que significa um aumento exponencial dos custos associado a uma administração da informação quase impraticável. Por isso, é necessário considerar um sistema de processamento analítico que seja escalável nas suas várias vertentes, de modo a dar resposta aos constrangimentos indicados. Diversas soluções arquitecturais e conceptuais têm sido propostas na literatura, para as quais as propostas de optimização descritas na secção anterior foram endereçadas. A optimização do processo de materialização de vistas é um primeiro grande passo para assegurar o crescimento sustentado de um sistema OLAP, independentemente do seu enquadramento espacial e temporal. Uma politica consistente de materialização de vistas, capaz de responder a um imenso número de interrogações em tempo útil confirma o sucesso de um sistema analítico. Veja-se o exemplo da figura 26.

Contudo, esta situação é apenas uma parte do problema de optimização. Importa considerar também outros factores como a própria evolução do conceito e arquitecturas do *Data Warehouse* e como estes "abriram o caminho" para dinamizar as configurações dos sistemas de processamento analítico. Na realidade, a própria origem do conceito de *Data Warehouse*

resulta da convergência entre a evolução dos sistemas de informação e a sua aparente impossibilidade de adaptação face as necessidades das organizações. De facto, a necessidade de um sistema especializado de suporte à tomada de decisões baseado em dados históricos, isolado dos sistemas operacionais, que permitisse acompanhar o negócio de forma consistente impôs a emergência no aparecimento dos sistemas de *Data Warehouse*: procurar respostas rápidas para os requisitos de negócio com o mínimo de intercessão dos processos próprios dos sistemas operacionais. Por outro lado, a maior ou menor insistência na obtenção atempada da informação impôs também a sua rápida introdução no mundo real, consubstanciando a necessidade inevitável de evoluir a suas arquitecturas de suporte.

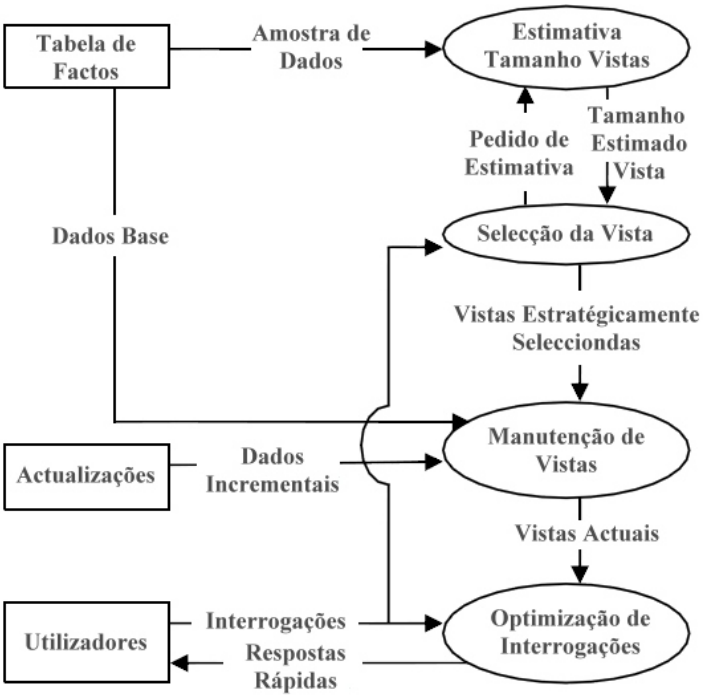


Figura 26. Exemplo de uma política consolidada de selecção de vistas

Note-se que todos os factores discutidos anteriormente nascem da procura crescente de informação analítica. Por isso, uma arquitectura de um *Data Warehouse* pode assumir várias configurações possíveis, não só por questões de administração e desempenho, mas também pelas imposições que lhe são efectuadas devido sobretudo ao desenvolvimento e operação das organizações. Aqui, podem ser considerados os *Data Warehouses* empresariais centralizados, os empresariais distribuídos e uma perspectiva híbrida, com conceitos das duas propostas

anteriores. Veja-se de imediato, o primeira. A abordagem centralizada é a mais tradicional, aquela em que a maioria das pessoas pensa quando se fala nestes assuntos. Os dados são armazenados numa única base de dados corporativa, com um modelo de dados único, de modo a permitir a relação dos dados com quaisquer sectores de actividade de uma organização, proporcionando aos agentes de decisão uma visão sólida e integrada dos dados face às áreas mais relevantes de negócio. É uma abordagem interessante do ponto de vista da disponibilização dos dados, mas que pode trazer problemas de futuro: quando o número de fontes é grande e diversificado face a um número de utilizadores crescente, torna-se difícil a escalabilidade do sistema, o que pode resultar no falhanço total do cumprimentos dos objectivos propostos para um *Data Warehouse*, já que o aumento da sua dimensão é inevitável. Apesar disso, muitas das proposta evidenciadas na secção anterior para o problema de selecção de vistas a materializar, foram descritas como estáticas centralizadas, dirigidas a este tipo de arquitectura. Posteriormente, surgem os problemas de manutenção, optimização e sobretudo, questões relacionadas com a janela de processamento disponível para extrair, preparar e integrar novos dados no *Data Warehouse*. Consequentemente, o tempo de readaptação das estruturas multidimensionais de dados torna-se progressivamente maior à medida que o *Data Warehouse* vai aumentando, tornando a sua disponibilidade cada vez mais reduzida para consultas.

Face a esses constrangimentos, novas abordagens surgiram para readaptarem as configurações tradicionais. A criação de *Data Marts* foi uma das primeiras, potenciada pelo facto dos departamentos de uma organização estarem apenas interessados numa parte dos dados de um *Data Warehouse*. Esta situação permitiu a criação de estruturas mais leves, que atenuassem a janela temporal necessária para as consultas e o tempo de refrescamento, proporcionando consultas mais rápidas e agrupadas por sectores de actividade. Estes factores conduziram ao aparecimento de um novo conceito, justificado por duas realidades: uma abordagem do tipo centralizada, em harmonia com um *Data Warehouse* central e abordagem descentralizada, que remete para a criação de um *Data Mart* departamental. Nesta ultima abordagem, deverá existir uma camada que assegure a obtenção de uma visão corporativa dos dados, de modo a não criar ilhas de informação em cada departamento e não tornar limitada a visão integral da

informação empresarial (figura 23). Em qualquer uma destas abordagens, as proposta de selecção de vistas a materializar (em especial, as centralizadas) continuam a ser as mais apropriadas. A abordagem distribuída dos *Data Marts* num arquitectura de *Data Warehousing* aliada com a materialização de vistas correspondentes no ambiente de processamento analítico motivou o aparecimento de uma nova acção interna: a distribuição de estruturas multidimensionais de dados por vários nós, aproveitando os benefícios de utilização de plataformas de redes e hardware comuns (M-OLAP). Como discutido na secção anterior, a arquitectura M-OLAP é constituída por um conjunto de nós que correspondem a diversos servidores OLAP distribuídos e interligados entre si, por uma rede de comunicações. Conceptualmente, cada servidor pode responder a interrogações colocadas por uma grupo de utilizadores, com espaço de armazenamento e capacidade de processamento próprios.

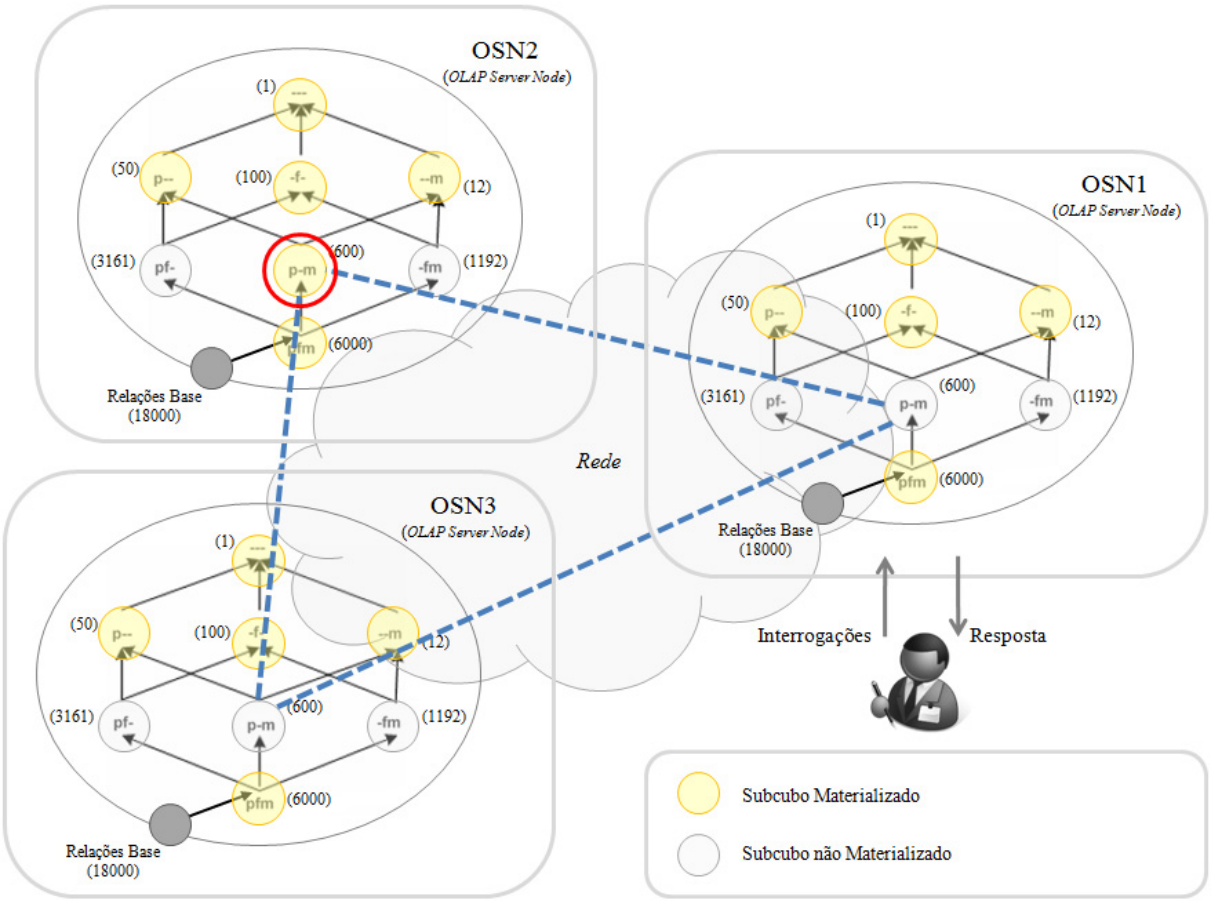


Figura 27. Exemplo de uma arquitectura M-OLAP

Assim, o problema de selecção e manutenção de vistas a materializar está condicionado ao espaço disponível, à capacidade de processamento e ao perfil das interrogações distribuídas de cada nó. As possibilidades de comunicação alargam as correlações entre os subcubos do *lattice* de pesquisa para fora de cada nó: por exemplo, se no servidor OLAP onde é lançada a consulta não existir vista materializada para dar uma resposta imediata ao pedido, as interrogações são lançadas para os nós vizinhos (com outras vistas materializadas) de modo a alargar as hipóteses de encontrar a solução requerida num tempo de resposta reduzido (veja-se o exemplo da figura 27). Na literatura, podem ser vistos diversos algoritmos de selecção de vistas vocacionados para arquitecturas M-OLAP [Loureiro & Belo, 2006], [Loureiro & Belo, 2007a], [Loureiro & Belo, 2007b]. Sobre os *Data Warehouses* empresariais distribuídos, veja-se a arquitectura descentralizada indicada na figura 23 (b). Ao alargar a sua configuração de modo a ter em conta a localização geográfica de diferentes *Data Marts*, está-se a evoluir para um arquitectura designada por *Data Warehouse* Federado [Informática, 1997]. As vantagens desta solução passam pela criação e gestão flexível dos Data Marts de forma individual, mas sempre permitindo uma visão global e integrada da informação empresarial, através de uma camada intermédia de ligação. Contudo, este tipo de arquitectura torna-se mais difícil de gerir no global e o nível de complexidade é elevado face ao processamento das interrogações, distribuídas pelos diversos *Data Marts*.

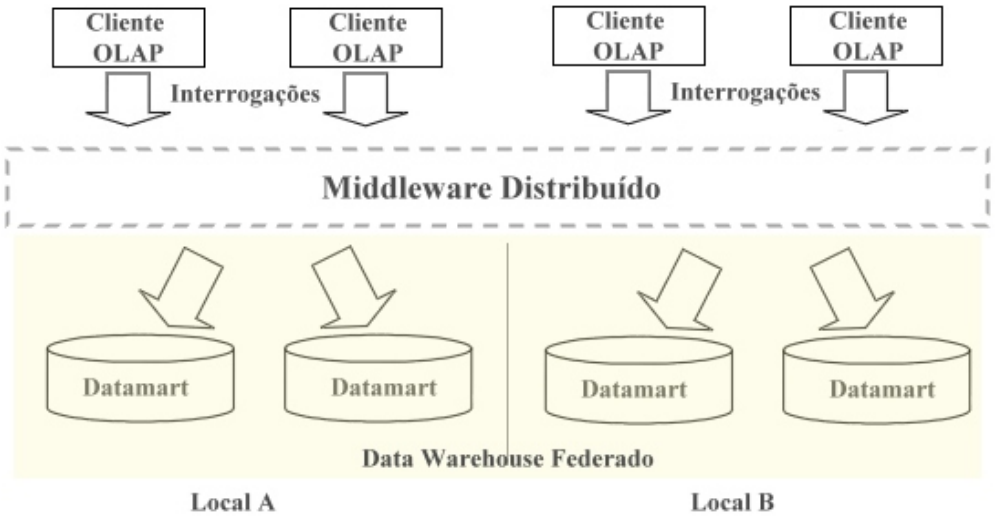


Figura 28. Arquitectura genérica de um Data Warehouse Federado

Nas arquitecturas híbridas, encontram-se incorporadas algumas das abordagens já mencionadas, onde são acrescentadas novas camadas de armazenamento ou processamento a nível mais local. Estas camadas são adoptadas de modo a diminuir os custos de comunicação decorrentes das abordagens distribuídas. São camadas onde se pode encontrar a replicação de algumas estruturas, procurando aproximar os dados da sua fonte de utilização. Isso pode ser efectuado através de servidores intermédios de processamento analítico ou mesmo colocando algumas dessas camadas nas máquinas dos próprios clientes. A vantagem da inclusão de novas classes OLAP intermédias possibilita a readaptação activa (dinâmica) ou pró-activa das estruturas multidimensionais de dados residentes em cada nó, em função do perfil de necessidades dos utilizadores. Os servidores de processamento analítico, geradores de informação multidimensional a partir de um *Data Warehouse*, conheceram também diversas implementações ao longo do tempo, cujas abordagens devem ser seleccionadas tendo em consideração os constrangimentos (espaço e tempo, principalmente) de cada realidade. A implementação de servidores de processamento analítico podem ser catalogados em três grandes categorias:

- Os servidores **Relacionais OLAP** (ROLAP) são considerados intermediários entre um servidor de *back-end* relacional e as aplicações cliente. Utilizam um gestor de base de dados relacional para gerir o repositório de dados e uma camada intermédia OLAP para gerir os conteúdos agregados. Este tipo de servidor inclui, geralmente, um processo de optimização para o gestor de base de dados relacional, a implementação de uma lógica de navegação agregada e ferramentas e serviços adicionais para a manipulação de dados. Na literatura, [Agarwal et al., 1996] propõe um conjunto de métodos para o processamento eficiente de agregações multidimensionais em servidores ROLAP.

- Os servidores **Multidimensionais OLAP** (MOLAP) permitem visualizar os dados numa perspectiva multidimensional, através de motores que agregam os dados e os colocam em estruturas multidimensionais. Efectuam o mapeamento das vistas multidimensionais directamente para as células do cubo. A vantagem de utilizar um

cubo é que este permite indexação dos dados agregados. Muitos servidores MOLAP adaptam uma representação do armazenamento para tratar separadamente *lattices* dispersos e densos: os subcubos mais densos são identificados e guardados em estruturas multidimensionais, enquanto que os subcubos mais dispersos são submetidos primeiramente a um algoritmos de compressão para racionalizar o espaço disponível. Contudo, a tecnologia MOLAP tende a ter menos escalabilidade que as implementações ROLAP. Sobre esta tecnologia, encontram-se referências em [Zhao et al., 1997] onde é proposto o método *MultiWay* para processamento de cubos em ambientes MOLAP. [Ross & Srivastava, 1997] apontam o problema da dimensionalidade na materialização de vistas e desenvolvem um método para processar estruturas de dados dispersos em ambientes multidimensionais.

- Os servidores **Híbridos OLAP** (HOLAP) combinam a tecnologia ROLAP e MOLAP, beneficiando da escalabilidade da abordagem relacional e do rápido processamento das abordagens multidimensionais. Por exemplo, um servidor híbrido tem a capacidade de armazenar um grande número de dados detalhados numa base de dados relacional, enquanto as agregações são mantidas num repositório multidimensional separado. Na literatura, [Kaser & Lemire, 2005] propõem melhorias aos mecanismos de armazenamento neste tipo de tecnologia.

Esta secção prova que, de facto, as exigências decorrentes do mundo empresarial moderno são cada vez mais complexas, num sistema em constante mudança. Os sistemas de processamento analítico, ao acompanharem essa evolução, são forçados a evoluir as suas arquitecturas e modos de funcionamento. Na prática, o crescimento contínuo dos sistemas OLAP em dimensão e utilização impôs a necessidade de alterações constantes na gestão da informação e na infra-estrutura física de suporte. A visão multidimensional dita o seu sucesso, contudo, o aumento da complexidade destas estruturas implicou novas abordagens para a sua optimização (em resposta as perspectivas de selecção de estruturas multidimensionais de dados tradicionais). Mas não basta ter uma política consolidada de selecção de vistas materializadas. É necessário também haver máquinas de processamento mais poderosas, com

capacidade de resposta para processar interrogações cada vez mais frequentes, complexas e variáveis. A escalabilidade dos sistemas OLAP deve, por isso, ser assegurada em todos os domínios de modo a garantir a sua sobrevivência. A sua adequação ao meio será tanto melhor se forem implementados algoritmos optimizados no processo de selecção de cubos, aliados a uma infra-estrutura analítica de última geração (seja ela centralizada ou distribuída).

Capitulo 3

Cubos Icebergue: Materialização Optimizada de Vistas

3.1. Processamento Eficiente de Estruturas Multidimensionais

O processamento eficiente de cubos de dados é um tarefa fundamental em qualquer ambiente analítico. O pré-processamento integral de um cubo pode reduzir drasticamente o tempo de resposta das interrogações e melhorar a qualidade do sistema de processamento analítico. Essa acção corresponde, na sua essência, ao cálculo e armazenamento das agregações de dados multidimensionais de maneira a que seja possível analisá-los imediatamente. No entanto, esta tarefa consome bastantes recursos a nível de tempo e espaço, sendo o desempenho uma questão fundamental no que se refere ao processamento do cubo, amplamente investigado na literatura actual. Note-se que análise de dados permite agregar valores e extrair informação estatística útil que serve para relacionar diversos domínios. Nesse sentido, [Gray et al., 1996] propõe a seguinte classificação para as funções de agregação:

- **Funções distributivas –** A função F*()* é considerada distributiva se existir uma função *G()* tal que $F(\{X_{i,j}\}) = G(\{F(\{X_{i,j} \mid i = 1, ..., I\}) \mid j = 1, ..., J\})$; como exemplos, apontam-se as funções *COUNT()*, *MIN()*, *MAX()* e *SUM()*;

- **Funções algébricas –** A função *F()* é considerada algébrica se existir um tuplo de ordem *M* de valores de uma função *G()* e uma função *H()* tal que $F(\{X_{i,j}\}) = H(\{G(\{X_{i,j} \mid i = 1, ..., I\}) \mid j = 1, ..., J\})$; como exemplo, aponta-se a função *AVG()*;

- **Funções holísticas –** Uma função *F()* é considerada holística se não existir uma constante M tal que caracterize o cálculo $F(\{X_{i,j} \mid i = 1, ..., I\})$; como exemplos, apontam-se os valores da mediana ou da moda.

As agregações provenientes de funções distributivas são relativamente fáceis de processar na medida em que a natureza da própria função de agregação permite que, sendo o núcleo representado como um vector *n-dimensional* em memória e sabendo que cada dimensão tem uma cardinalidade *Li +1*, os agregados podem ser calculados através da projecção de cada uma delas. As agregações provenientes de funções algébricas são mais difíceis de calcular que as distributivas, pois este tipo de funções pressupõem o cálculo de valores intermédios para poderem apresentar o resultado final. No que se refere às funções holísticas, a forma mais eficiente de processar as agregações é recorrendo ao algoritmo 2^n. Esta solução começa por reservar um apontador para cada célula do cubo e, sempre que é recebido um novo tuplo $(x_1, x_2, ..., x_n, v)$ em que x_i representa as coordenadas dessa célula e *v* o valor nela contido, a função de agregação é invocada 2^n vezes, ou seja, uma vez para cada apontador de cada célula do cubo cujo valor equivale a *v*. Quando todos os tuplos tiverem processados, a função final é invocada para cada um dos *(Li + 1)* nós do cubo, em que *Li* corresponde à cardinalidade da dimensão *i*.

Se o cubo não puder ser mantido em memória devido à sua dimensão, não pode ser representado através de vectores. Por isso, devem ser usadas técnicas que permitam fazer a sua partição, tais como ordenação ou recurso a funções de dispersão. Neste contexto, apontam-se três abordagens centrais no processamento eficiente de cubos: 1) os algoritmos baseados em ordenação e dispersão (*sorting* e *hashing*), 2) os algoritmos baseados em partições ou fragmentação de agregações e 3) os algoritmos de agregação recorrendo a vectores multidimensionais (MOLAP), detalhados em [Agarwal et al., 1996], [Ross &

Srivastava, 1997], [Zhao et al., 1997]. A primeira referência foi das primeiras a evidenciar a questão da optimização do processamento de agregados, salientando a necessidade de generalizar os operadores de agregação para poderem ser aplicados no cálculo de cubos. Note-se que qualquer um dos algoritmos indicados processam vários agregados segundo uma lógica sequencial. Assim, de modo a estabelecer quais os nós que podem ser processados a partir de outros e qual a sequência pela qual os atributos devem ser tratados, recorre-se à conceptualização do *lattice* de correlações [Harinarayan et al., 1996] demonstrado no seguinte exemplo: ao assumirem-se para um determinado cubo as dimensões (produto, cidade, ano) e (vendas totais) como métrica de análise, o número total de subcubos é calculado por $\prod_{i=1}^{n}(Li + 1)$ onde L_i representa o número de níveis possíveis para as dimensões representadas em *i*. Neste caso, o número de subcubos (ou *group-by*) a processar é calculado por $2^3 = 8$. As combinações possíveis são {(produto, cidade, ano), (produto, cidade), (produto, ano), (cidade, ano), (produto), (cidade), (ano), ()} onde () representa o subcubo vazio (nó onde as dimensões não são agrupadas). Em termos SQL, as agregações de um cubo são referenciadas com instruções de *group-by*. Cada instrução pode ser associada a um subcubo; o conjunto de todos os subcubos (*group-by*) forma o *lattice* de correlações, que está na base da definição espacial das estruturas multidimensionais.

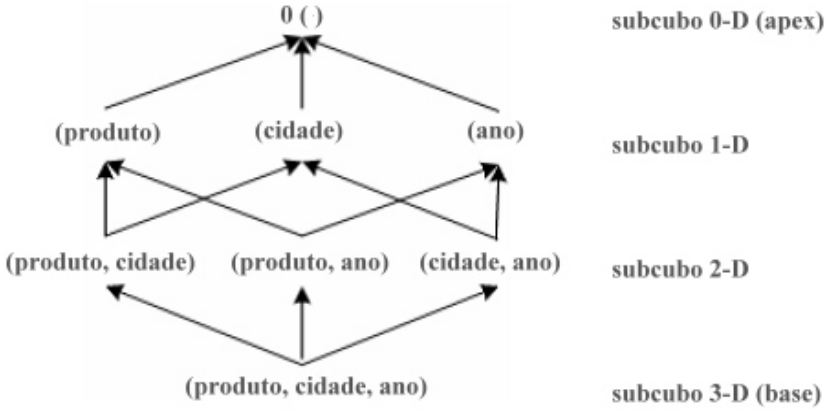


Figura 29. Representação do lattice de subcubos do exemplo (cada um deles um group-by)

Ao considerar as "vendas totais", o subcubo base (que contém todas as dimensões) pode retorquir essa informação combinando qualquer uma das dimensões (produto, cidade, ano); já o subcubo *apex* refere-se ao caso onde o *group-by* está vazio (ou seja, contém somente o

somatório total das vendas). Isto significa que o subcubo base é o menos generalizado (mais especifico) do *lattice* de correlações, enquanto que o subcubo *apex* é o mais generalizado (portanto, o menos especifico). Estes dois pontos são considerados os limites da representação do espaço de soluções (os restantes subcubos permanecem entre eles). Se começar no subcubo *apex* e navegar em direcção a um nível menos generalizado no *lattice*, será o mesmo que efectuar a operação OLAP de *drill-down*; *roll-up* será, previsivelmente, o inverso. Uma *query* SQL que não tenha a condição *group-by* (processamento do total de vendas) é um operador de zero dimensões; por outro lado, uma *query* que contenha uma condição de agrupamento (processamento do total de vendas, por produto) é considerada um operador de uma dimensão. Um operador de um cubo em *n* dimensões é equivalente ao conjunto de instruções *group-by*, uma para cada subcubo das *n* dimensões. Portanto, o operador do cubo é a generalização de *n* dimensões da instrução *group-by* (ou de outras instruções SQL de agregação básicas) como ilustrado na figura 30 [Gray et al., 1996].

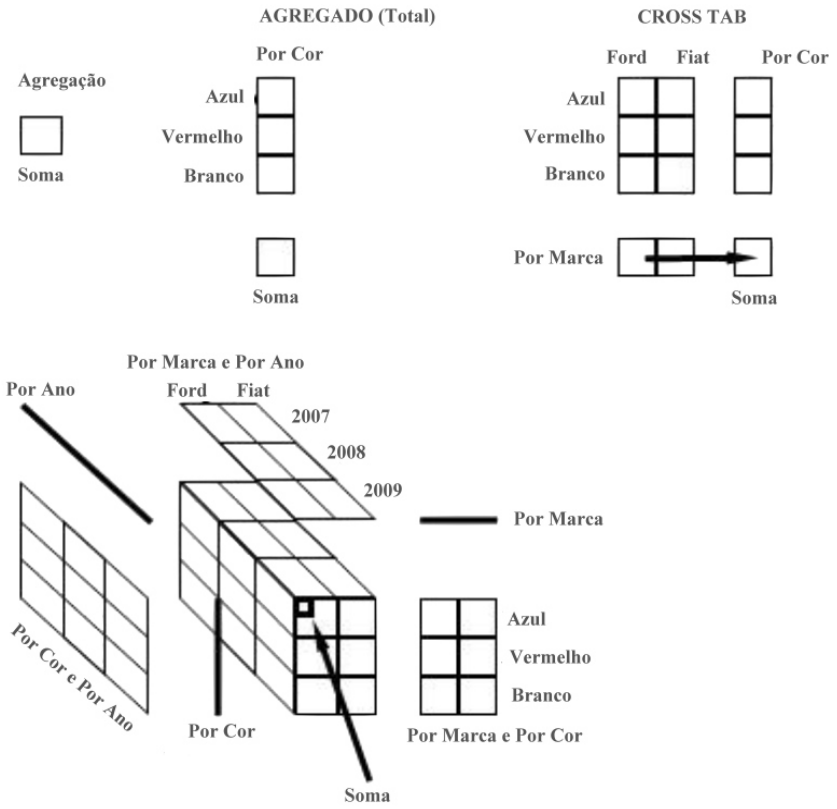


Figura 30. Operador Cubo de 1-3 dimensões para uma operação básica de agregação

No geral, existem duas estruturas de dados essenciais utilizadas para armazenamento de cubos de dados: as tabelas relacionais são utilizadas como estruturas base para implementação de ROLAP e os vectores multidimensionais, para a implementação de MOLAP. Nessa perspectiva, os algoritmos de pré-processamento de cubos, apesar de operarem em diferentes ambientes, partilham alguns "truques" de optimização comuns que devem ser utilizados no processamento eficiente dessas estruturas [Agarwal et al., 1996]:

- **Ordenação, agrupamento e *hashing*[17] –** Este tipo de operações podem ser aplicadas aos atributos das dimensões a fim de reorganizar a agrupar os tuplos relacionados. No processamento de cubos, a agregação é efectuada nos tuplos que partilham o mesmo conjunto de valores duma dimensão. Estas operações contribuem para optimizar o acesso e agregação desses dados de modo a facilitar o processamento de agregações.

- **Agregações em paralelo e *caches* intermédios –** No processamento de cubos de dados é mais eficiente processar as agregações de alto nível através das de baixo nível, processadas anteriormente, em vez de utilizar as tabelas base (de factos). Adicionalmente, as agregações paralelas resultantes do processamento dos *caches* intermédios pode levar a uma redução acentuada nos custos de armazenamento em disco (leitura e escrita).

- **Agregações a partir dos subcubos mais pequenos –** Quando existem múltiplos subcubos, normalmente é mais eficiente processar o subcubo mais generalizado a partir do subcubo mais pequeno, processado anteriormente. Outro truque que pode ajudar a optimizar o processamento de cubóides é o mapeamento de atributos alfanuméricos de dimensões em números inteiros, compreendidos entre 0 e o valor da cardinalidade do atributo (utilizado normalmente nos cubos icebergue).

[17] ***Hashing*** ou ***hash*** é um procedimento bem definido ou uma função matemática que converte grandes quantidades de dados variáveis num único identificador, usualmente representado por um número inteiro, para cada parcela de dados. Esta acção permite indexar cada porção dos dados convertidos e manter a sua integridade.

- **Filtragem APRIORI na optimização de cubos icebergue** – A propriedade APRIORI [Agrawal & Srikant, 1994] no contexto do processamento *iceberg cubes*, define o seguinte: se uma determinada célula de um subcubo não satisfazer o patamar de suporte definido, então nenhum dos seus descendentes (os nós por baixo) o fará também. Esta propriedade pode ser usada para reduzir substancialmente o tempo de processamento de *iceberg cubes*. Recorde-se que este tipo de estruturas contêm uma condição, que é um constrangimento para as células que vão ser materializadas. Uma condição icebergue comum indica que as células têm de satisfazer um patamar mínimo (por exemplo, uma contagem ou um somatório) para serem seleccionadas. Nesta situação, a propriedade APRIORI pode ser usada para filtrar e retirar (*pruning*) os descendentes de uma célula que não satisfaça a condição. Veja-se o seguinte exemplo: Se a contagem de uma célula *c* de um subcubo é menor do que a condição icebergue *v*, então a contagem de quaisquer descendentes da célula *c* nos subcubos por baixo nunca pode ser ≥ que *v*, pelo que podem ser filtrados e retirados. Por outras palavras, se uma condição icebergue (especificada em SQL por uma cláusula *having*) é violada por algumas células *c*, então qualquer dos seus descendentes irão violar também a condição. As medidas que obedecem a esta propriedade são designadas por anti-monótonas e permitem ganhar tempo e espaço no processamento de cubos.

Os trabalhos mais recentes na área de OLAP e geração de hipercubos incidem sobre uma variante do problema tradicional de processamento de cubos. Como se sabe, o problema tradicional consiste, essencialmente, em calcular todos os agregados da forma mais eficiente possível, tendo em conta que o problema é exponencial em relação ao número de dimensões consideradas. Além disso, há que ter em atenção que o tamanho de cada agregado depende da cardinalidade das dimensões que o compõem. Note-se que até ao momento, foram discutidos métodos que permitem calcular as agregações das estruturas multidimensionais de dados recorrendo a algoritmos que executam o pré-processamento integral do cubo. Contudo, em ambientes reais, esta acção é praticamente impossível. Na prática, assiste-se a verdadeiros desafios contra o tempo de processamento e espaço de armazenamento disponíveis decorrentes do processo de geração do cubo. Existe, portanto, necessidade de encontrar outros

caminhos, em contraste com o cálculo integral de agregações de modo a optimizar-se toda a estratégia de materialização de cubos. Ao detalhar o processo de materialização de vistas nas secções 2.6 e 2.7 deste trabalho, verifica-se que uma escolha adequada dos subcubos mais benéficos pode trazer ganhos muito elevados, conseguindo-se com uma percentagem baixa do espaço disponível, valores quase idênticos aos tempos de resposta das interrogações efectuadas. Mas, se este é um processo que consome alguns recursos a nível de espaço e de tempo de processamento, importa encontrar um meio-termo entre os ganhos de desempenho e os recursos disponíveis. A materialização parcial de estruturas multidimensionais oferece um *trade-off* interessante entre o espaço disponível e o tempo de resposta requerido pelos ambientes OLAP. Em vez do cálculo integral do cubo, é possível processar apenas uma parte dos subcubos. Note-se que cada célula do cubo contém um valor agregado. Medidas como contagens ou somatórios são frequentemente usadas, o que pode gerar a alocação de agregações de valor 0 nas células do cubo. Quando o produto das cardinalidades para as dimensões do subcubo é grande, em relação ao número de tuplos diferentes de 0 armazenados nas células, diz-se que é um subcubo disperso ou esparso (*sparce*) [Ross & Srivastava, 1997].

Na realidade, os dados reais são frequentemente de natureza esparsa, o que justifica o estudo e desenvolvimento de técnicas orientadas para esse tipo de dados. Os algoritmos baseados na partição dos dados usam dois princípios a que se recorre frequentemente para realizar operações complexas sobre relações amplas: efectuar a partição das relações em fragmentos que possam ser armazenados em memória e efectuar a operação sobre cada um desses fragmentos independentemente. Apenas os elementos considerados úteis devem ser representados. Isto é conseguido através da implementação de uma estratégia de selecção de subcubos consolidada. Nesse contexto, e ao aplicar-se o conhecido *trade-off* entre o espaço e o tempo, é possível em muitos casos optimizar ainda mais, o processo de selecção de vistas, já que grande parte do espaço de armazenamento do cubo está ocupado por células cujas agregações são de valor insignificante para as análises efectuadas (normalmente porque as células do cubo encontra-se amplamente esparsas dentro do espaço multidimensional de soluções). Na prática, devido à dispersão tendencial dos dados reais, uma estratégia para aumentar o desempenho dos algoritmos nessas situações passa por identificar, antes do

processamento, quais os *group-by* de interesse a materializar. Veja-se o seguinte exemplo: um cliente só pode ir às compras a uma loja de cada vez; esse evento vai gerar algumas células preenchidas, mas a maior parte ficará vazia. Nestas situações, é útil materializar apenas as células de um subcubo (*group-by*) cujas agregações estejam acima de um determinado patamar, de modo a não considerar os nós vazios ou com informação irrelevante. No cubo de vendas, esta situação pode ser ilustrada através das seguintes condições: materializar somente as células dos subcubos onde *count* ≥ *10* (ou seja, onde existirem pelo menos 10 tuplos para as células, dada uma combinação de dimensões) ou apenas as células que representam vendas ≥ *100€*. A definição destes patamares de suporte permite, não só preservar tempo de processamento e espaço de armazenamento, como também focalizar a análise para aquilo que realmente é importante (as células que não cumprem o limite estabelecido são susceptíveis de serem demasiado triviais para justificar uma análise mais aprofundada). Tais subcubos gerados através de materialização parcial são denominados por *iceberg cubes* [Beyer & Ramakrishnan, 1999] - a detalhar nas próximas secções.

3.2. Optimização de Querys Icebergue

A abordagem mais simples para armazenar todas as agregações é numa estrutura *hash* [Ramakrishnan & Gehrke, 2000] aplicável quando estas cabem na memória disponível. Mas ao considerar um ambiente com *n* dimensões, as agregações excedem significativamente o espaço disponível, pelo que não é uma boa opção. Muitas aplicações sobre estruturas multidimensionais de dados, principalmente nos sistemas de suporte à decisão, envolvem a execução de *querys* SQL que processam funções de agregação sobre um conjunto de atributos, devolvendo apenas os valores agregados que satisfazem um predicado de comparação simples em relação a um ou mais patamares definidos pelo utilizador. Quando esse limite é suficientemente restritivo para condicionar o resultado apenas a uma pequena fracção do número total de grupos de um cubo, a interrogação denomina-se por *iceberg query*. Como indicado em [Fang et al., 1997] um protótipo de uma interrogação desta classe para uma dada relação $I\ (a_1, a_2, \ldots, a_k, med)$ limitada por um patamar T pode ser representado e generalizado por:

```
SELECT a1, a2, …, ak, f_agregação (med)
  FROM I
 GROUP BY a1, a2, …, ak
HAVING f_agregação (med) >= T;
```

Figura 31. Notação geral de uma iceberg query

Onde os valores de $(a_1, a_2, \ldots, a_k)$ identificam cada grupo de interesse enquanto que *med* refere-se ao(s) campo(s) a partir da qual a função de agregação está a ser processada, podendo *I* ser uma relação materializada única ou gerada por uma função de junção a partir das relações base do cubo. Os sistemas gestores de base de dados não aplicam nenhuma técnica especial para processar as interrogações icebergue nos ambientes multidimensionais. Por isso, independentemente do valor definido para um determinado patamar mínimo de suporte, tipicamente empregam um dos seguintes algoritmos no processamento dessas interrogações: o primeiro, SMA (*Sort-Merge-Aggregates*), estabelece que uma dada relação *I* é totalmente ordenada em disco em relação aos atributos do *group-by*. Através de uma única análise sequencial sobre a ordenação, são retornados os agregados cujos valores satisfazem a condição de limitação [Garcia-Molina et al., 2000]; o segundo, HHA (*Hybrid-Hash-Aggregate*) indica que uma dada relação *I* é particionada recursivamente por intermédio de funções de *hashing*, resultando em partições sobre os grupos de atributos, colocadas na memória disponível para serem subsequentemente processadas [Graefe, 1993].

Na generalidade, estas estratégias representam um desperdício no processamento de *iceberg querys* uma vez que não consideram o predicado condicional do patamar mínimo de suporte, limitando a sua sensibilidade para a elegância deste tipo de instruções. Motivados por esta observação, foram propostos na literatura um conjunto personalizado de algoritmos para lidar com o processamento eficiente de *iceberg querys* [Fang et al., 1997]. Estes algoritmos (referenciados como CIQE) são baseados em combinações de diversas técnicas de amostragem e *hashing* de modo a eliminar os *group-by* que não fazem parte do resultado da interrogação, optimizando o seu tempo de resposta face aos algoritmos originais SMA ou HHA. Entre os algoritmos CIQE, destacam-se dois deles pelo seu desempenho: 1) *Defer-*

Count, que opera num espaço de soluções, onde uma amostra aleatória da base de dados é utilizada para identificar potenciais *group-by* candidatos, escalando o resultado da amostra em relação ao tamanho da base de dados. De seguida, é executada uma análise *hashing* na base de dados do modo a identificar outros *group-by*, comparando esses com todos os candidatos até ao momento identificados, de modo a obter-se aqueles que cumprem exactamente o patamar definido na interrogação icebergue. O segundo algoritmo 2) *Multi-Stage* utiliza também uma amostra aleatória recolhida da base de dados para processamento. No entanto, os potenciais candidatos são identificados e colocados numa *pool* auxiliar de grupos. De seguida, executa-se uma análise *hashing* que valida cada *group-by* da *pool* de modo a filtrar aqueles que estão "marcados" como potenciais candidatos, sendo esses, posteriormente, submetidos a uma nova avaliação para se tornarem candidatos definitivos.

Os algoritmos CIQE são considerados os pioneiros na optimização do processamento de *querys* icebergue. Contudo, só podem ser aplicados em ambiente icebergue restritivos, especialmente onde os valores agregados dos *group-by* correspondam a uma distribuição altamente enviesada, onde o operador de agregação seja *COUNT()* ou *SUM()* e o predicado de comparação seja ≥. Das limitações indicadas, repare-se na intensidade da última. Ao restringir o predicado a ≥ significa que apenas podem ser processadas *querys* icebergue "altas" (correspondentes à procura de grupos que excedam o patamar mínimo de suporte). Contudo, na prática, é igualmente provável que o utilizador possa estar interessado em efectuar *querys* icebergue "baixas" onde os grupos desejáveis encontra-se sob um determinado patamar. À primeira vista, pode parecer que as *iceberg querys* baixas derivam das altas e, portanto, podem ser manipuladas pelos algoritmos CIQE. Mas, na realidade, as *iceberg querys* baixas são um problema bem mais complexo na medida em que se desconhecem técnicas eficientes para identificar as frequências mais baixas numa distribuição [Ioannidis & Poosala, 1995]. Em [Leela et al., 2002] foi efectuado um estudo de modo a comparar, numa perspectiva quantitativa, o desempenho dos algoritmos CIQE (*Defer-Count* e *Multi-Stage*) em relação as abordagens minimalistas SMA e HHA no processamento de *iceberg querys*, sobre diferentes conjuntos de dados.

Adicionalmente, foi acrescentado um algoritmo ORACLE[18] que sabe, logo de inicio, a identidade dos grupos qualificados para responder a uma *iceberg query* (qualquer um dos algoritmos propostos, na prática, têm de fazer este trabalho de modo a responder à interrogação). Os resultados foram os seguintes (considerando os dados de *input* na tabela 2):

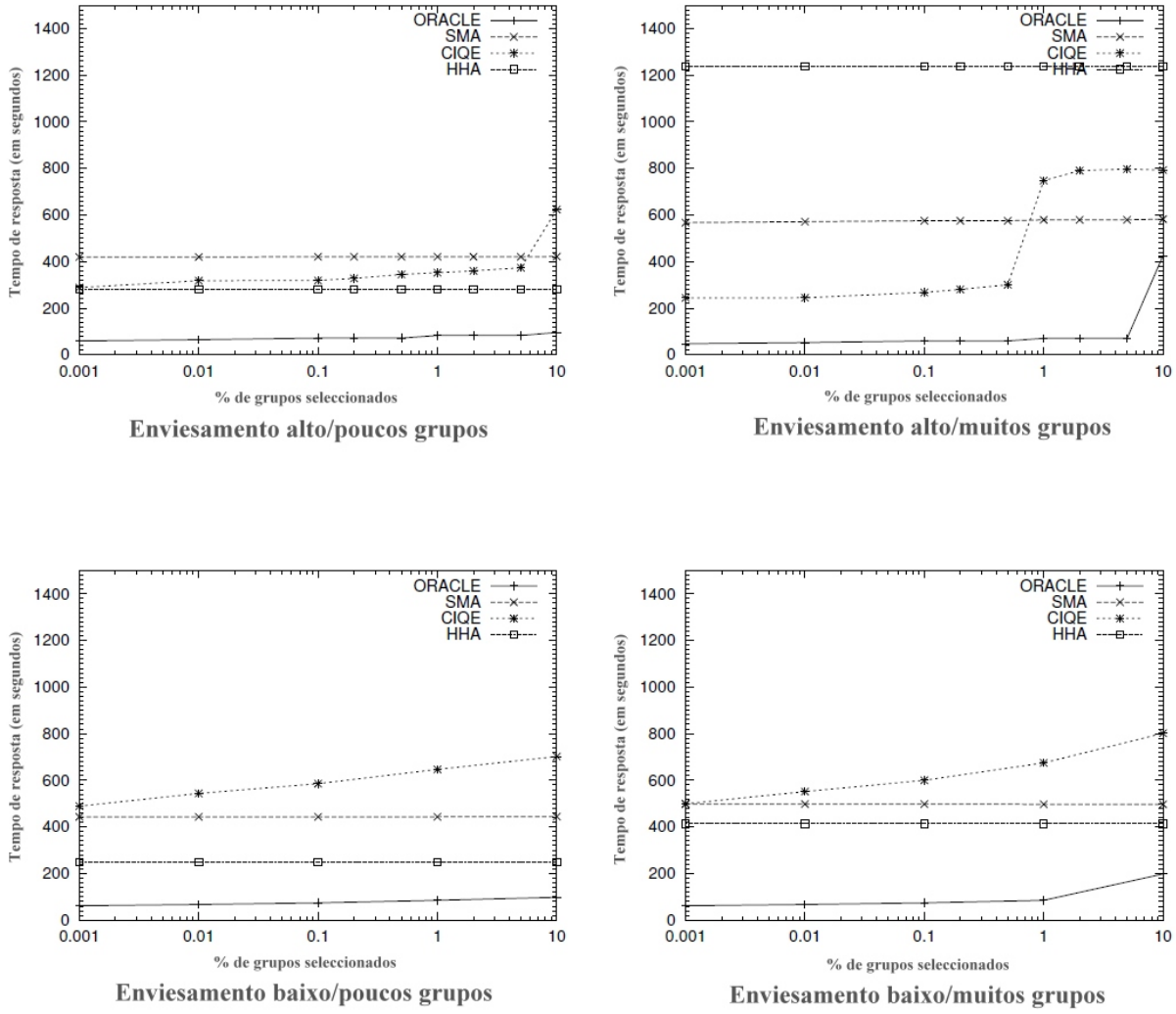


Figura 32. Comparativo - Algoritmos de processamento optimizado de iceberg querys [Leela et al., 2002]

[18] Um algoritmo **ORACLE** é uma técnica usada em testes de *software* para determinar se o teste efectuado passou ou não. Usa-se através da comparação do resultado do sistema em teste, para um determinado *input*, com o resultado determinado pelo algoritmo ORACLE (que é o desejado). É um algoritmo óptimo de teste utilizado como ponto de referencia para medir a viabilidade e o desempenho das situações a testar.

Conjunto Dados	Cardinalidade	Numero Grupos	Tamanho BD	Tamanho Tuplos	Tamanho Grupos	Tamanho Medidas	Enviesamento	Contagem Alvo
D1	1	10M	1GB	16	4	4	1657	194780
D2	2	62M	1GB	16	8	4	1541	194765
D3	1	8.38M	1GB	16	4	4	1,27	28
D4	2	16.4M	1GB	16	8	4	0,89	18

Tabela 2. Características dos conjuntos de dados a testar

O resultado da análise efectuada mostra que os algoritmos CIQE têm um desempenho superior em relação aos SMA para conjuntos de dados com um baixo ou médio número de grupos e um desempenho moderado em situações de grande enviesamento. Na situação em que as relações base encontram-se materializadas e não existe espaço disponível em disco para ordenar as relações, o desempenho do CIQE nunca é duas vez melhor em comparação com o SMA. Mesmo assim, existe ainda uma lacuna acentuada de desempenho entre os algoritmos analisados e o ORACLE, o que evidencia a necessidade de investigar melhores algoritmos optimizados para processamento de *querys* icebergue. Paralelamente a estes resultados, existem mais algumas referências na literatura na tentativa de desenvolver soluções optimizadas para o processamento de *iceberg querys* [Matias & Segal, 1999], [Gilbert et al., 2001], [Lazaridis & Mehrotra, 2001].

Por exemplo, em [Matias & Segal, 1999] é proposto um esquema para fornecer respostas rápidas aproximadas à *query* icebergue, com a intenção de auxiliar o utilizador a redefinir o patamar mínimo de suporte antes de emitir a interrogação icebergue "final", com o limite adequado. A solução pretende assim minimizar a necessidade de um especialista para decidir se a interrogação retorna, de facto, a "ponta do icebergue" desejada. Recorde-se que os algoritmos CIQE funcionam apenas para as funções de agregação do tipo *COUNT()* e *SUM()*. Posteriormente, foram propostos em [Bae & Lee, 2000] dois algoritmos para lidar com *querys* icebergue cuja função de agregação seja do tipo AVG() e que operam através do particionamento lógico de uma relação. Isto permite identificar os grupos candidatos, considerando que para um grupo satisfazer a condição icebergue, deve estar acima do patamar definido pelo menos numa partição. Dirigidos para o problema de selecção de *iceberg cubes* (a discutir na próxima secção) encontram-se igualmente mencionados na literatura diversos

trabalhos cujo objectivo principal consiste no processamento de uma parte restrita de um cubo de dados de modo a reduzir os recursos necessários para calcular e armazenar o cubo. As técnicas envolvidas revelam alguns conceitos avançados no processamento optimizado de *iceberg querys*, maioritariamente relacionados com a filtragem (*pruning*) do *lattice* de correlações [Agrawal & Srikant, 1994]. A estratégia de *pruning* é alargada ao processamento de medidas complexas (incluindo médias) em [Han et al., 2001] e ao processamento de *iceberg querys* em ambientes distribuídos, investigado em [Wagner & Yin, 2001]. Analisados os algoritmos essenciais de processamento de interrogações e as soluções optimizadas de processamento de *querys iceberg*, importa agora reflectir sobre como estas foram estendidas ao processamento parcial de cubos e qual a sua importância na materialização de vistas. Na prática, a formalização do trabalho desenvolvido por [Fang et al., 1997] sobre *iceberg querys* motivou a introdução de uma variante ao problema tradicional *CUBE* (processamento de todas as agregações de uma estrutura multidimensional de dados) denominada por *iceberg cubes* [Beyer & Ramakrishnan, 1999]. Este tipo de estruturas permitem, selectivamente, calcular apenas as partições que satisfazem uma condição agregada especificada pelo utilizador, cuja base de geração é estabelecida pelo conjunto de *iceberg querys* efectuadas sobre um subcubo.

3.3. Processamento de Cubos Icebergue

As tecnologias de *Data Warehousing* e de OLAP requerem, nos dias de hoje, vistas sumarizadas e orientadas ao negócio para uma análise rápida e eficiente no processo de tomada de decisões. Por isso, os dados são modelados multidimensionalmente. Num modelo multidimensional, perspectivas de análise como "produto" ou "clientes" descrevem assuntos de interesse; as métricas, como "total de vendas" é o alvo da análise, em termos de dimensões de negócio. As estruturas multidimensionais generalizam o operador SQL *group-by* [Gray et al., 1996] para processar esses operadores em todas as combinações das dimensões. Cada *group-by* representa um subcubo que envolve um conjunto de agregações agrupadas pelas mesmas dimensões. Dado um conjunto de dimensões i onde cada dimensão tem uma cardinalidade de L_i, o número potencial de subcubos para o cubo de dados será de $(L_i + 1)^i$. O

processamento introduzido pelo operador do cubo *group-by* podem ser enormes, pois para *i* atributos base especificados, $\mathbf{2^i}$ *group-by* são calculados. Quando o operador é usado para responder a um conjunto de *iceberg querys*, é gerado um *iceberg cube* [Beyer & Ramakrishnan, 1999]. A formulação destas estruturas deve ser considerada com uma estratégia dinâmica de selecção de subcubos, a complementar com algoritmos, já discutidos, que identificam os *group-by* de forma estática. Dada uma função agregada, o processamento do cubo passa por calcular todos os subcubos que respondam a estas condições. Os algoritmos optimizados de selecção de subcubos exploram as vantagens do processamento paralelo de cada subcubo mais especifico através dum outro menos especifico (*Top-down*) ou vice-versa (*Bottom-up*). Os *iceberg cubes* foram propostos para processar apenas as agregações interessantes do ponto de vista das necessidades do negócio. Na prática, os utilizadores podem especificar condições icebergue nos valores agregados dos subcubos. Somente aqueles que satisfazem a condição, são seleccionados. Portanto, as *iceberg conditions* podem ser monótonas ou anti-monótonas [Beyer & Ramakrishnan, 1999]. Um constrangimento monótono indica: se um subcubo de agregações violar a condição icebergue, todos os subcubos dependentes a violarão. Este recurso é amplamente usado para filtrar (*pruning*) um *iceberg cube*. Um *iceberg cube* contêm apenas as células agregadas do cubo de dados que satisfazem uma determinada condição, conhecidas como a "ponta do icebergue".

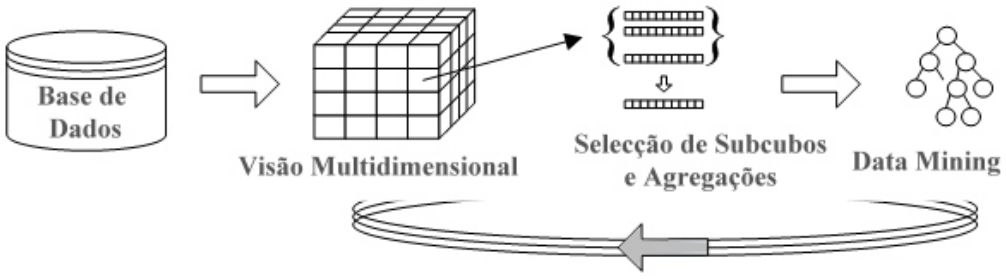


Figura 33. Importância da selecção de subcubos no processo de extracção de conhecimento

Este tipo de estruturas têm como objectivo identificar e calcular apenas os valores que serão os mais adequados para serem materializados e que farão sentido para as análises de suporte à decisão, ou seja, a condição agregada qualifica quais as agregações do cubo que são mais relevantes para uma determinada consulta e, portanto, as que devem ser materializadas.

Essencialmente, as interrogações efectuadas num *iceberg cube* partem da análise de um atributo (ou conjunto de atributos) de forma a obterem-se as agregações que estão acima de um determinado limiar ou patamar de suporte, face a determinadas condições (expressas em SQL por *having count* ou *having sum*, por exemplo). Estas interrogações são conhecidas como *iceberg querys*, porque conseguem lidar apenas com os valores que estão acima de um determinado limiar (a "ponta do icebergue") face ao volume total do cubo de dados (o "icebergue"), utilizando menos espaço de armazenamento e, significativamente, menos tempo de processamento sobre os dados. Análogo ao problema de selecção das vistas consideradas mais adequadas para serem materializadas, a selecção dos *group-by* que, de facto, satisfazem a condição icebergue estabelecida são também matéria de investigação na literatura, estando na base do denominado problema de selecção de *iceberg cubes*. Nesse contexto, o problema é manifestado pela determinação de qual o conjunto de subcubos que satisfazem uma determinada condição agregada (logo, os mais benéficos) de modo a serem integrados numa vista a materializar. Tendo em conta o contexto de dados esparsos, a condição icebergue estabelece que uma partição deve ter pelo menos *n* tuplos, sendo *n* designado como o suporte mínimo da partição ou patamar. Veja-se o seguinte exemplo: para um cubo de três dimensões *(A, B, C)* o problema pode ser representado pela seguinte interrogação:

```
COMPUTE CUBE ICEBERG_CUBE AS
SELECT A, B, C, COUNT(*)
  FROM R
  CUBE BY A, B, C
HAVING COUNT(*) >= min_sup;
```

Figura 34. Criação do iceberg cube a partir de uma iceberg query

A instrução *compute cube* especifica o pré-processamento do cubo *iceberg_cube* com as dimensões *(A, B, C)* e a função de agregação *COUNT(*)*. Os tuplos de entrada encontram-se na relação *R*. A cláusula *cube by* especifica quais as agregações (*group-by*) que são para ser formadas por cada um dos subconjuntos das dimensões em análise. Repare que *min_sup* representa o patamar de suporte mínimo da partição para que uma determinada agregação possa satisfazer a condição icebergue (neste caso, sendo uma contagem, refere-se ao número

mínimo de tuplos que um *group-by* tem de ter). O resultado do processamento da interrogação pode ser usado para responder a qualquer *query* numa combinação de atributos das dimensões *(A, B, C)* que requeiram que *COUNT(*)* seja ≥ que o patamar de satisfação mínimo estabelecido. Se fosse para processar o cubo de dados na sua totalidade, cada *group-by* iria corresponder a um cubóide no *lattice* de dependências. No entanto, a condição especificada na cláusula *having* é conhecida como a condição icebergue, que opera, neste caso, sobre a função *COUNT(*)*. Note-se que o cubo icebergue processado na figura 34 pode ser usado para responder às agregações em qualquer combinação das dimensões distinguidas pela instrução *having count(*) ≥ v* onde *v ≥ min_sup*. Em vez do *COUNT(*)* podem ser usadas também outras funções mais complexas, como por exemplo a média. Na realidade, se fosse suprimida a cláusula *having* do exemplo anterior, iria acabar-se por processar o cubo na sua totalidade. Naturalmente, que se o patamar definido para o cubo icebergue for 1, o resultado vai ser o cálculo integral do cubo. A tabelas 2(a) demonstra uma relação simples dos atributos de quatro dimensões de diferentes cardinalidades. Suponha-se que o patamar para satisfazer a *iceberg query* indica que a combinação dos valores dos atributos devem aparecer pelo menos em três tuplos (*min_sup ≥ 3*). O resultado do algoritmo icebergue é mostrado na tabela 2(b).

A	B	C	D
a1	b2	c1	d1
a2	b1	c1	d2
a2	b2	c2	d2
a3	b2	c2	d1
a3	b3	c1	d1
a4	b3	c3	d2
a5	b2	c2	d1

(a)

Combinações	Count(*)
b2	4
b2-c2	3
c1	3
c2	3
d1	4
d2	3

(b)

Tabela 3. (a) representa o cubo de dados simples e (b) o cubo icebergue derivado do primeiro

Uma abordagem "ingénua" para processar um *iceberg cube* seria, em primeiro lugar, calcular o subcubo na totalidade e só depois, filtrar as células que não satisfazem a condição icebergue. Contudo, esta acção continua ser proibitivamente cara devido às limitações de

espaço de armazenamento. Por outro lado, as abordagens optimizadas defendem que, para processar o *iceberg cube* directamente, não é necessário que os cubóides estejam previamente calculados (este tipo de algoritmos serão evidenciados na secção seguinte deste trabalho). Ao utilizar-se estas estruturas será possível reduzir a carga de processamento daquelas que são consideradas as agregações triviais de uma estrutura multidimensional. Recorde-se que um algoritmo de *iceberg cubing* é motivado pelo facto do processamento integral de uma estrutura multidimensional requer espaço de armazenamento, que é exponencial ao número de dimensões. Para um cubo de grandes dimensões, muitas vezes não é suportável o pré-processamento integral. Por isso, é necessário traçar um método para avaliar quais os fragmentos do cubo que serão mais úteis processar. Um número considerável de autores propõe várias abordagens para processar apenas um subconjunto dos *group-by* ao invés do cubo na sua totalidade [Baralis et al., 1997], [Shukla et al., 1998], [Gupta, H. et al., 1997], [Harinarayan et al., 1996]. Estes algoritmos permitem escolher os *group-by* a processar baseados em diferentes factores tais como o espaço de armazenamento disponível, o tamanho esperado dos *group-by*, o beneficio previsto do pré-processamento das agregações, etc. Contudo, essas soluções dependem de um critério de selecção de subcubos estático. Uma das inovações dos cubos icebergue é exactamente permitir que os utilizadores definam o critério dinamicamente (de acordo com os seus interesses analíticos) durante o processamento das agregações.

3.4. Descrição das soluções optimizadas de exploração de cubos icebergue

Os processos de optimização descritos anteriormente são considerados os alicerces para um processamento eficiente de cubos icebergue. Na realidade, a importância dos algoritmos de exploração de *iceberg querys* e a forma de como estas se coadunam com uma estratégia consistente de selecção dos subcubos mais importantes a materializar, evidenciam a elegância destas soluções no processamento eficiente de estruturas multidimensionais. Importa, agora, adicionar uma outra dimensão fundamental que servirá para completar o ciclo de optimização quando se recorre a este tipo de estruturas, relacionada com a descrição de diferentes algoritmos que permitem gerar *iceberg cubes* de forma mais eficiente e que apresentam

lógicas distintas para evitar o processamento integral de todas as células possíveis no cubo, optimizando o tempo de cálculo, ignorando aquelas que não apresentam uma utilidade analítica aceitável. Note-se que não se pretende demonstrar exaustivamente todas as soluções e algoritmos existentes sobre cubos icebergue, mas apenas referir as mais representativas, pelo seu grau de inovação quando foram apresentadas, pela sua importância ao abrirem uma nova linha de investigação ou família de soluções ou, ainda, pela sua eficácia na actualidade. Irá adoptar-se uma abordagem semelhante ao efectuado na secção 2.8, aquando levantamento das lógicas optimizadas para selecção e manutenção de vistas materializadas, evidenciando a distribuição dos algoritmos propostos. A maioria das soluções é focada na perspectiva centralizada, já que é a mais utilizada e também, até à muito pouco tempo, a única a ser implementada. Contudo, algumas propostas distribuídas encontram-se também presentes na literatura e que interessam mencionar, sobretudo, pelas melhorias que introduzem a nível de capacidade de cálculo paralelo e partilha de recursos no processamento de *iceberg cubes*.

[Zhao et al., 1997] foram dos primeiros autores a apresentar uma abordagem optimizada para o processamento eficiente de estruturas multidimensionais. Note-se que esta proposta é especialmente dirigida ao processamento integral do cubo, não sendo, por natureza, uma solução vocacionada para o processamento de *iceberg cubes*. No entanto, muitos dos seus conceitos podem ser usados para entender grande parte dos algoritmos de selecção optimizada de subcubos a materializar, pelo que será importante mencionar as suas características. O algoritmo de agregação 1) *MultiWay* é um método para calcular um cubo integral de dados recorrendo a vectores multidimensionais como estrutura elementar de dados. É uma abordagem típica MOLAP que utiliza um sistema de endereçamento directo de vectores, cujos valores das dimensões podem ser obtidos através das suas coordenadas na sequência; percorre as células dos vectores de tal forma que não seja necessário repetir a operação para calcular cada uma das agregações. Por questões de desempenho, sobretudo pelo constrangimento do espaço em memória, os vectores têm de ser armazenados, divididos em outros de menor dimensão, recorrendo-se a uma estratégia de *chunking* (divisão de vectores *n-dimensionais* em vários vectores *n-dimensionais* mais pequenos – *chunks* – representados por objectos armazenados em disco). Porém, especialmente no que se refere a dados reais, é

frequente que muitas das células do *chunk* estejam vazias, o que significa que não existem dados para essa combinação de coordenadas. Um *chunk* é considerado denso quando mais de 40% das células contêm um valor válido [Zhao et al., 1997]. Quando esta situação não se verifica, diz-se que o *chunk* é esparso, pelo que é necessário aplicar-se um método de compressão de tal forma que cada célula fica associada a um valor inteiro que indica o seu afastamento (*offset*) em relação ao início do *chunk*, evitando assim o armazenamento de células vazias. Desta forma, cada entrada válida passa a ser representada por um par (*chunk_id* + *offset*). O recurso ao *chunking* assegura a eficiência do processo de carregamento e armazenamento dos valores das células do cubo. Por outro lado, o cálculo dos agregados é efectuado através da "visita" aos valores residentes nas células do vector. A ordem em que as células são visitadas pode ser optimizada de modo a minimizar o número de vezes que cada célula é revisitada, reduzindo, logicamente, custos de espaço e processamento. Este "truque" permite explorar esta ordem de modo a calcular paralelamente os agregados parciais e evitar visitas repetidas às células. Dado que o *chunking* envolve a "sobreposição" de alguns agregados processados, este método é designado por *multiway array aggregation*, na medida em que efectua agregações simultâneas em múltiplas dimensões.

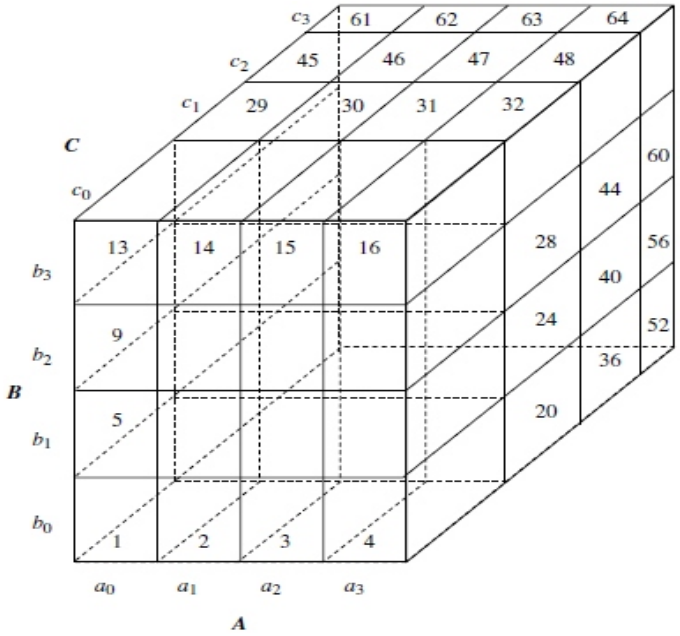


Figura 35. Vector tridimensional para as dimensões (A, B e C) organizado em 64 chunks

De modo a coordenar o processamento simultâneo de vários agregados é utilizada uma árvore de cobertura, para representar a memória mínima que é necessária para processar os *chunks* dada uma determinada ordem de dimensões - *Minimum Memory Spanning Tree (MMST)*. A *MMST* é uma árvore de cobertura mínima para um cubo $(D_1, D_2, ..., D_n)$ construída segundo a ordem de dimensões $O = (D_{j1}, D_{j2}, ..., Djn)$, sendo *j* o nível na *MMST*. Trata-se de uma árvore com $n+1$ níveis, com $(D_{j1}, D_{j2}, ..., D_{jn})$ no nível *n*, e em que qualquer nó num nível *i* inferior ao nível *n* pode ser calculado a partir dos nós no nível acima (que contêm todas as dimensões que compõem esse nó). O cálculo da memória necessária para as agregações é efectuado do seguinte modo: para calcular o agregado $(D_{j1}, D_{j2}, ..., D_{jn})$ do vector $(D_1, D_2, ..., D_n)$ lido na ordem $O = (D_{j1}, D_{j2}, ..., Djn)$ e que contém um prefixo de $(D_1, D_2, ..., D_n)$ de comprimento *p* $(0 \leq p \leq n-1)$, é necessário alocar $\prod_{i=1}^{p} |Di| * \prod_{i=p+1}^{p} |Ci|$ unidades do vector, onde $|Di|$ é o tamanho da dimensão *i* e $|Ci|$ o tamanho do *chunk* para a dimensão *i*. Diferentes ordenações das dimensões $(D_1, D_2, ..., D_n)$ podem gerar *MMST* com diferentes requisitos de memória. Veja-se o seguinte exemplo: ao considerar um vector multidimensional com quatro dimensões *ABCD* com *10x10x10x10 chunks*, onde $|A| = 10$, $|B| = 100$, $|C| = 1000$ e $|D| = 10000$ representam a alocação em memória. Note-se que a *MMST* para a ordenação *(D, B, C, A)* requer, aproximadamente, 4 *gigabytes* de memória, enquanto a *MMST* para a ordenação *(A, B, C, D)* requer apenas 4 *megabytes*!

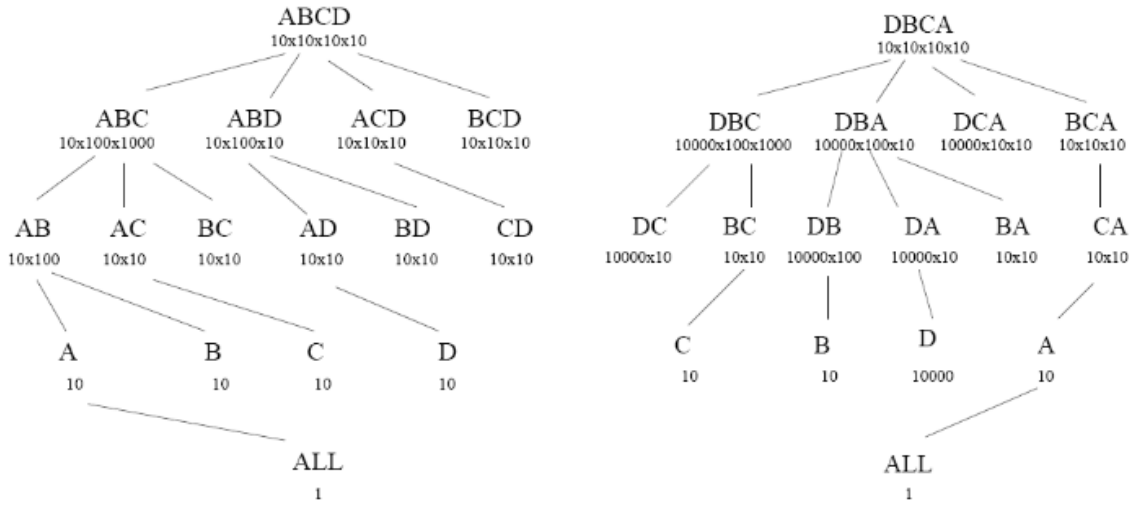


Figura 36. Requisitos de memória para duas ordenações diferentes das dimensões [Zhao et al., 1997]

Através da utilização de técnicas de compressão adequadas em *chunks* esparsos e da ordenação consistente do cálculo de subcubos, o algoritmo *MultiWay* tem mostrado resultados significativamente superiores em relação às propostas tradicionais ROLAP em relação ao processamento de estruturas multidimensionais de dados. Contudo, esta observação funciona apenas em cubos com um número relativamente pequeno de dimensões (já que o número de subcubos a processar é exponencial ao número de dimensões existentes). Imagine-se agora utilizar esta abordagem para processamento de *iceberg cubes*: Relembre-se, desde já, que a propriedade APRIORI [Agrawal & Srikant, 1994] define que, se uma determinada célula não satisfaz o patamar mínimo de suporta da partição, então nenhuma das suas descendentes o satisfará. Infelizmente, o processamento do algoritmo *MultiWay* inicia no subcubo base e vai progredindo no lattice de correlações para cima, em direcção a subcubos mais generalizados. Não pode, por isso, tirar partido da filtragem APRIORI que indica: um nó "pai" só pode ser obtido através do seu filho (mais especifico). Por exemplo, se o *COUNT(*)* de uma célula *c* nos subcubos *(A, B)* não satisfazer o suporte mínimo definido na condição icebergue, então não é possível filtrar e eliminar o cálculo das antecessoras de *c* em A e B, porque a contagem dessas células pode ser superior, em relação a *c*.

Ao contrário da abordagem *MultiWay,* [Beyer & Ramakrishnan, 1999] propõem um algoritmo *Bottom-up* denominado por 2) BUC (*Bottom-up cube*) vocacionado para cubos esparsos e para o problema dos cubos icebergue. Este inicia-se percorrendo o *lattice* de correlações do subcubo *apex* em direcção ao subcubo base. Isto permite que os custos de particionamento dos dados sejam partilhados. Este algoritmo procura combinar a eficiência a nível de *inputs* e *outputs* do algoritmo *Partitioned-Cube* (segundo grupo de algoritmos indicado na secção 3.1) [Ross & Srivastava, 1997] com o beneficio da filtragem APRIORI durante o processamento. Cada dimensão é lida e dividida segundo o número de ocorrências para cada um dos valores dessa dimensão, sendo as restantes dimensões calculadas recursivamente para cada partição. O algoritmo recorre à estratégia dividir para conquistar (*divide-and-conquer*) uma vez que, após uma determinada partição ter sido calculada, todos os subcubos descendentes são calculados antes do algoritmo transitar para a partição seguinte. O movimento da base para o topo, como já demonstrado, permite que seja realizada uma filtragem recorrendo à

propriedade anti-monótona empregue no algoritmo APRIORI, usado para descobrir associações frequentes entre objectos. Para melhor se entender o algoritmo, observe agora o seguinte exemplo aplicado a um cubo icebergue de quatro dimensões *(A, B, C, D)* onde o patamar mínimo de suporte da partição é ≥ *3* (figura 37). Suponha-se os seguintes valores distintos para as dimensões *A* (a_1, a_2, a_3, a_4), *B* (b_1, b_2, b_3, b_4), *C* (c_1, c_2) e *D* (d_1, d_2). Ao considerar-se cada *group-by* uma partição, é necessário processar todas as combinações das agregações para satisfazer o patamar de suporte mínimo definido (neste caso, que tenham pelo menos três tuplos). A figura 37 ilustra como o *input* é particionado, primeiro, em função dos diferente valores dos atributos da dimensão *A, B, C* e *D*. Para isso, o BUC analisa o *input*, agregando os tuplos de modo a obter a contagens para todos, correspondentes às células *(*, *, *, *)*. A dimensão *A* é usada para fraccionar o *input* em quatro partições, uma para cada valor distinto de *A*. O número de tuplos (contagem) para cada valor distinto de *A* é guardado numa variável *dataCount*. O algoritmo ganha tempo aplicando a propriedade APRIORI enquanto procura por tuplos que satisfazem a condição icebergue. Iniciando com o valor a_1 da dimensão *A*, a partição de a_1 encontra-se agregada, criando um tuplo para o *group-by* de *A*, que corresponde à célula $(a_1, *, *, *)$. Suponha-se agora que $(a_1, *, *, *)$ é ≥ *3*; uma chamada recursiva é efectuada na partição para a_1.

O algoritmo vai particionar a_1 na dimensão *B*. Vai verificar a contagem de $(a_1, b_1, *, *)$ para validar se esta satisfaz o patamar de suporte mínimo. Se sim, gera o tuplo agregado para o *group-by AB* e conduz $(a_1, b_1, *, *)$ para a partição em *C*, iniciando com c_1. Imagine-se agora que o número de células para $(a_1, b_1, c_1, *)$ é 2, que não satisfaz o patamar mínimo de suporte. Neste caso, aplicando a propriedade APRIORI, o algoritmo filtra qualquer outra exploração restante dependente de $(a_1, b_1, c_1, *)$. Com isto, evita o particionamento destas células na dimensão *D*. Posteriormente, regressa à partição $(a_1, b_1, *, *)$ e avança para $(a_1, b_1, c2, *)$, repetindo o processo por ai adiante. Ao verificar a condição icebergue sempre que é efectuada uma chamada recursiva, o algoritmo ganha bastante tempo em processamento sempre que a contagem de uma célula não satisfaça o patamar de suporte mínimo. O processo de particionamento é facilitado por um método de ordenação linear *CountingSort* [Sedgewick, 1990] que permite reaproveitar as contagens ordenadas para processar os *group-by* no BUC.

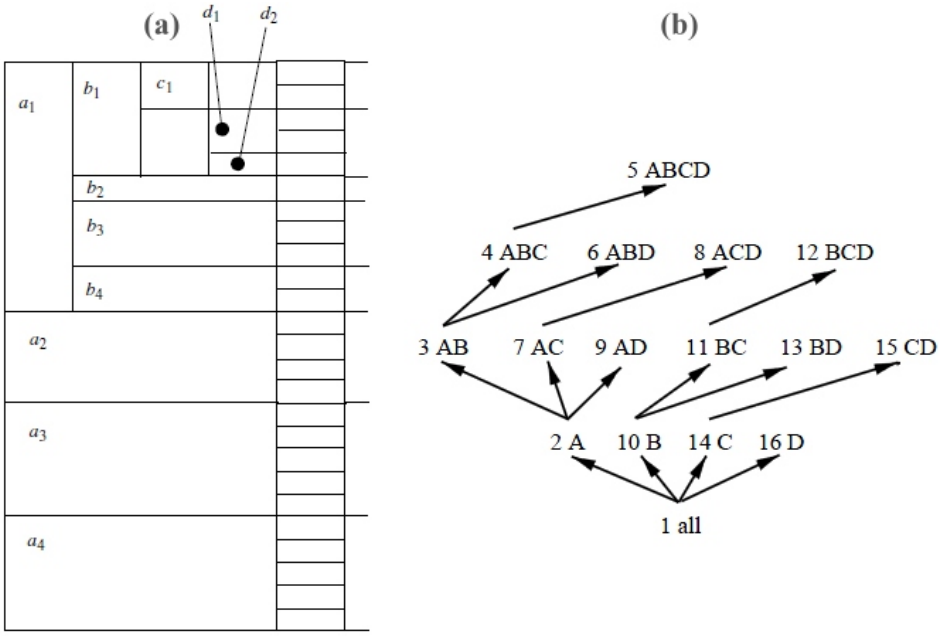


Figura 37. Exemplo BUC onde a) particionamento de 4 dimensões e b) lattice respectivo [Beyer & Ramakrishnan, 1999]

O desempenho do BUC é sensível à ordem das dimensões e ao enviesamento dos dados. Idealmente, as dimensões mais discriminastes devem ser processadas em primeiro lugar e em ordem decrescente de cardinalidade. Quanto maior for a cardinalidade, mais pequenas serão as partições, o que gera um maior número de partições, permitindo mais oportunidades ao BUC para aplicar a filtragem APRIORI. De modo semelhante, quanto mais uniforme for uma dimensão (menor enviesamento), mais adequada estará para ser filtrada. A maior contribuição deste algoritmo é o conceito de partilha dos custos de particionamento. Contudo, ao contrário do *MultiWay*, não partilha o cálculo das agregações entre os *group-by* "pai" e "filho". Por exemplo, o processamento do subcubo *AB* não auxilia o cálculo do *ABC*. Este último deve ser processado essencialmente a partir do zero.

Note-se que algoritmos como o *MultiWay*, *Partitioned-Cube* ou o BUC têm uma distribuição centralizada e que operam de forma sequencial. [Wagner & Yin, 2001] propõe uma curiosa evolução do BUC e um conjunto de outros algoritmos vocacionados para ambientes distribuídos, com objectivos comuns: explorar diferentes abordagens que envolvam cálculo paralelo e optimizar os *inputs* e *outputs* no processamento de *iceberg cubes*. O primeiro,

denominado por *Replicated Parallel BUC* (RP) é uma réplica do algoritmo original BUC, mas adaptado para ser executado paralelamente. Na sua essência, por cada uma das sub-árvores do *lattice* de correlações é gerada uma tarefa distribuída, ou seja, para uma *iceberg query* que envolve *m* atributos, são geradas *t* tarefas. A associação das sub-árvores aos processadores é efectuada de forma simples, através dum algoritmo de *round-robin*[19]. Contudo, o algoritmo RP não é muito eficaz na distribuição de tarefas e carga de processamento, pelo que os autores propõem um novo algoritmo derivado do RP denominado BPP (*Breadth-first writing, partitioned, parallel BUC*) [Wagner & Yin, 2001]. Este difere do primeiro em dois aspectos chave: em primeiro lugar, o conjunto de dados é particionado e executado em paralelo. Por outro lado, o resultado dos subcubos é executado através dum algoritmo de pesquisa *Breadth-first*[20]. Como o BPP é superior ao RP a nível de carga de processamento, é limitado quando os dados base atribuem algum enviesamento a alguns atributos. Esta situação ocorre, sobretudo, pela granularidade do RP e BPP ser relativamente grande e desigual.

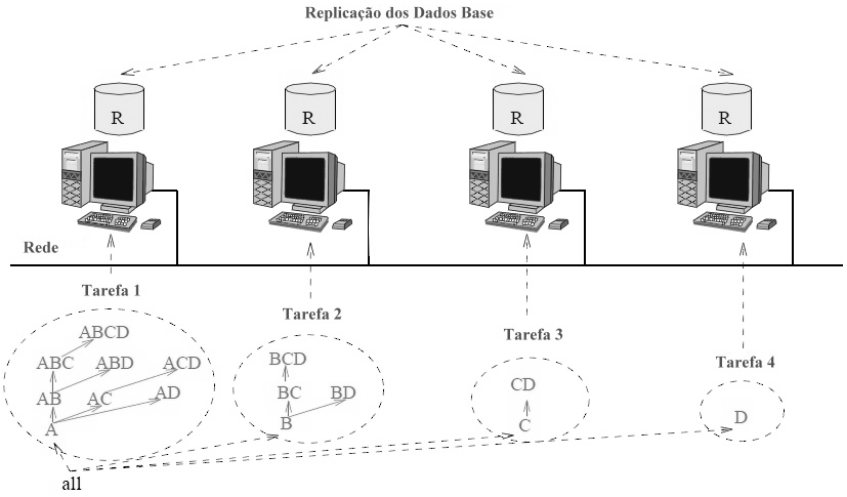


Figura 38. Distribuição do lattice BUC pelo algoritmo RP [Wagner & Yin, 2001]

[19] ***Round-robin*** é um algoritmo elementar que permite agendar processos em sistemas operativos. Atribui intervalos iguais de tempo de processamento a cada processo duma forma circular, gerindo-os sem prioridade. Trata-se de um algoritmo simples e de fácil implementação e pode ser aplicado também na gestão de pacotes de dados, em redes digitais.

[20] Na teoria dos grafos, ***Breadth-first*** é um algoritmo de pesquisa em grafos, que inicia num nó, a raiz, e explora todos os nós vizinhos. Depois, para cada um desses nós, pesquisa os seus "segundos" vizinhos que ainda não foram explorados, e por ai adiante. Termina quando a árvore estiver percorrida na sua totalidade.

De modo a considerar o balanceamento da carga a prioridade principal [Wagner & Yin, 2001] propuseram um novo algoritmo ASL (*Affinity skiplist*) no qual cada subcubo é tratado por uma tarefa. O algoritmo utiliza uma estratégia por afinidade de tarefas de modo a procurar a relação entre duas tarefas (associadas a um mesmo processador) e maximizar a partilha entre elas. Os benefícios desta estratégia motivaram o desenvolvimento de uma outra lógica de processamento denominada PT (*Partitioned tree*) [Wagner & Yin, 2001] que combina as ideias de filtragem APRIORI do BUC e o sistema de afinidade de tarefas do ASL. Efectua o processamento de tarefas de granularidade mais espessas recorrendo a um particionamento binário para dividir os subcubos em tarefas uniformes, de modo a balancear a carga de uma maneira controlada. Este algoritmo híbrido é executado através da lógica BUC, mas a associação de tarefas por afinidade é agendada de forma *Top-down*.

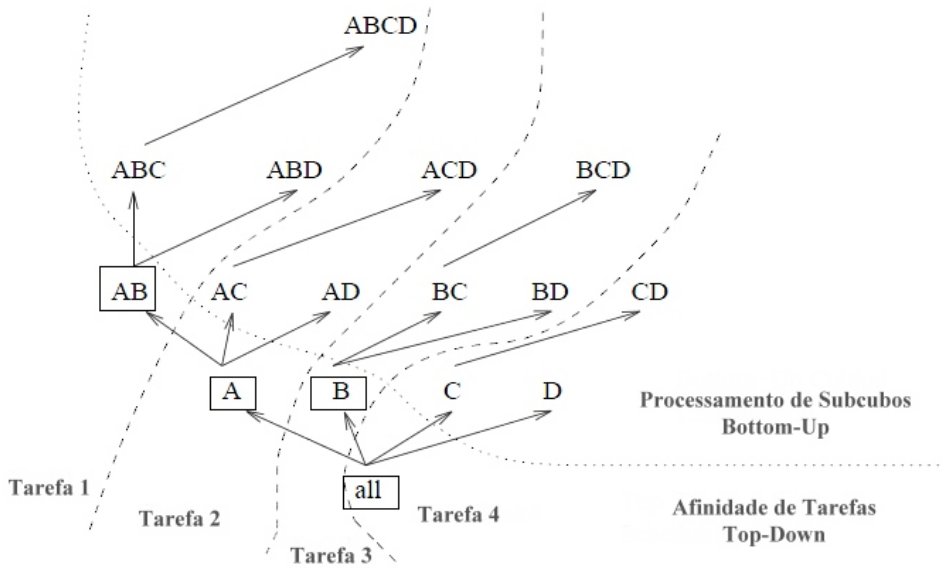


Figura 39. Divisão binária do algoritmo PT em quatro tarefas

Paralelamente a estas propostas de solução para o problema de processamento de *iceberg cubes*, surge um outro algoritmo, que propõe explorar as condições icebergue utilizando, sobretudo, funções complexas, como por exemplo, a média, denominado 3) H-CUBING [Han et al., 2001]. O recurso a este tipo de medidas que, normalmente não obedecem à propriedade anti-monótona, faz com que não possa ser aplicada a filtragem APRIORI. No caso concreto da média, o facto de o valor médio numa célula *c* de um subcubo *A* ser inferior a um valor

mínimo pré-definido não significa obrigatoriamente que o valor das células descendentes seja igualmente inferior. Apesar disso, os autores propõem uma metodologia que deriva numa condição anti-monótona, embora fraca, para testar e filtrar o espaço de pesquisa de soluções aquando a utilização de uma função *AVG()*, denominada filtragem *top-k average*. Adicionalmente, efectuam uma extensão dos métodos tradicionais APRIORI e BUC alargando-os para utilização com a média, denominados *top-k Apriori* e *top-k BUC*. O primeiro explora os cubos icebergue cujo patamar mínimo de suporte seja uma média, através da análise dos candidatos gerados e de um cálculo moderado de cada nível, de modo a evitar que todos os subcubos para trás sejam calculados e as células interessantes sejam seleccionadas por condições intermédias. Contudo, envolve custos de processamento, sobretudo porque pode gerar um número elevado de grupos candidatos. O *top-k BUC* baseia-se nos benefícios do algoritmo original. O BUC constrói o *iceberg cube* através da combinação de um número baixo de dimensões a tender para um número mais elevado. Explora a ordem das dimensões colocando aquelas que são mais discriminantes em primeiro lugar, para posteriormente, e de forma recursiva, particionar os subcubos de acordo com a seriação estabelecida. Em cada passo da partição recursiva, os nós são confrontados com a condição icebergue para remover aqueles que não a satisfazem. O *top-k BUC* particiona um grande conjunto de dados em conjuntos mais pequenos projectados sobre as dimensões correspondentes de modo a localizar o espaço de soluções para cada conjunto particionado. Ao explorar a partição das dimensões e confrontar cada nó com a condição icebergue, este algoritmo resulta em tempos de desempenho interessantes.

Os autores introduzem um novo conceito de estrutura em comparação com os algoritmos anteriormente discutidos, no qual se baseia o H-CUBING. Este algoritmo utiliza uma estrutura em árvore (*hyper-tree*) baseada na *FP-tree* utilizada no algoritmo *FP-growth* introduzido em [Han et al., 2000]. É característica desta estrutura o facto de poder ser gerada através duma única iteração de leitura da base de dados, já que a *hyper-tree* e respectiva tabela de suporte (*header table*) fornecem toda a informação necessária para calcular um cubo icebergue. Cada nível da árvore representa uma dimensão no subcubo base e cada tuplo de *n-dimensões* corresponde a um caminho com *n* nós na árvore. Os nós que se encontram ao

mesmo nível e contêm o mesmo valor estão ligados entre si. Cada nível está ligado à *header table* que regista a frequência de cada um dos valores possíveis das dimensões e mantém as ligações, dos primeiros nós correspondentes, a esses valores. A partir desta estrutura, o cubo icebergue pode ser calculado numa perspectiva *Bottom-up* ou *Top-down*. Em qualquer uma das maneiras, o algoritmo inicia num determinado nível da árvore e analisa todos os agregados que incluem a dimensão correspondente a esse nível e, as dimensões correspondentes, aos níveis superiores. A agregação é facilitada pela tabela de suporte: se o valor para um determinado nó é inferior ao mínimo estabelecido, a agregação é transporta para o próximo nó através do ponteiro lateral. A diferença entre as duas estratégias é o ponto de entrada do algoritmo, ou seja, este inicia-se na base da árvore ou no seu topo. Uma das vantagens deste algoritmo reside exactamente na estrutura em árvore que, ao eliminar dados duplicados, permite que se tire partido do cálculo simultâneo das agregações. São também processadas menos combinações da dimensão antes de se avançar para a próxima, o que beneficia a utilização da filtragem APRIORI neste tipo de estruturas. Porém, à semelhança do BUC, não pode usar os resultados intermédios do processamento das dimensões menores, especialmente indicados para calcular cubos icebergue de elevada dimensionalidade.

De modo a demonstrar a estruturação do algoritmo H-CUBING, veja-se o exemplo patente na tabela 4, a qual representa as vendas de um conjunto de produtos, por mês, dia, cidade, grupo, produto, custo e preço.

Mês	Dia	Cidade	Grupo	Produto	Custo	Preço
Jan.	10	Lisboa	Académico	Impressora	500	485
Jan.	15	Lisboa	Pessoal	TV Plasma	800	1200
Jan.	20	Lisboa	Académico	Câmara	1160	1280
Fev.	20	Santarém	Profissional	Portátil	1500	2500
Mar.	4	Setúbal	Académico	Disco Ex.	540	520
...	...	...	...	...	...	...

Tabela 4. Informação relativa a vendas (exemplo H-CUBING)

O *iceberg cube* das vendas pode ser construído através da interrogação icebergue indicada na figura 40, com a indicação dos respectivos limites. Repare que o subcubo *iceberg_vendas* difere do seu subcubo integral correspondente, na forma da seguinte restrição: são excluídas

todas as células cujo preço médio seja inferior a 800€ ou cuja contagem de tuplos seja menor que 50. A representação em árvore da seguinte situação é endereçada na figura 41.

```
COMPUTE CUBE ICEBERG_VENDAS AS
SELECT Mês, Cidade, Grupo, AVG(Preço), COUNT(*)
  FROM Vendas
  CUBE Mês, Cidade, Grupo
HAVING AVG(Preço) >= 800 AND COUNT(*) >= 50;
```

Figura 40. Processamento do iceberg cube de vendas (exemplo H-CUBING)

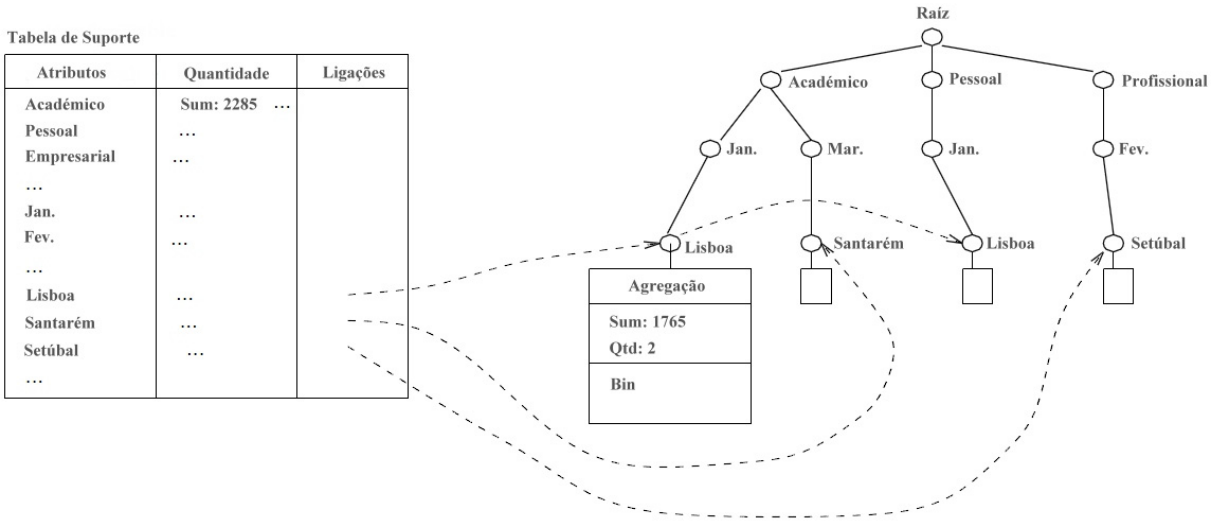


Figura 41. Aplicação do Algoritmo H-CUBING no cálculo optimizado do iceberg cube

A solidez e evolução dos conceitos desenvolvidos sobre os três algoritmos predominantes atrás mencionados (*MultiWay*, *BUC* e *H-CUBING*) permitiu dar origem a um novo método híbrido, o 4) STAR-CUBING (SC) [Xin et al., 2003], baseado numa estrutura dinâmica em árvore para processamento de *iceberg cubes*. Este algoritmo explora ambos os métodos *Top-down* e *Bottom-up*: na ordem de processamento global, o SC utiliza uma abordagem *Top-down*. Adicionalmente, dispõe de uma camada inferior à de processamento que assenta numa abordagem *Bottom-up*, de modo a explorar a noção de dimensão partilhada. Esta integração permite que o algoritmo consiga efectuar agregações sobre várias dimensões (agregações partilhadas) sem perder a capacidade de dividir os *group-by* "pais" e realizar a filtragem APRIORI sobre os *group-by* "filhos" que não respeitam a condição icebergue. Veja-se o

exemplo da figura 42, que representa o processamento de um *iceberg cube* de quatro dimensões *(A, B, C, D)*. Se fosse aplicada apenas a abordagem *Top-down* (semelhante ao *MultiWay*) então os cubos marcados como "filtrados" pelo algoritmo continuariam a ser explorados. O SC permite filtrar os subcubos indicados porque tem em consideração as dimensões partilhadas. *ACD/A* significa que o subcubo *ACD* tem a dimensão partilhada *A*, *ABD/AB* significa que o subcubo *ABD* tem a dimensão partilhada *AB*, *ABC/ABC* significa que o subcubo *ABC* tem a dimensão partilhada *ABC*, e assim adiante. Isto advêm da generalização de que: todos os subcubos fixados na sub-árvore *ACD* incluem a dimensão *A*, todos os subcubos fixados na sub-árvore *ABD* incluem a dimensão *AB* e todos os subcubos fixados na sub-árvore *ABC* incluem a dimensão *ABC* (mesmo se existir apenas um desses subcubos) - denominam-se as dimensões partilhadas, daquelas sub-árvores em particular. A introdução destas dimensões facilita o processamento partilhado do *iceberg cube*. Ao serem reconhecidas desde cedo no *lattice* de correlações, é possível evitar o seu reprocessamento mais tarde. Por exemplo, o subcubo *AB* estendido de *ABD* (figura 42) seria efectivamente filtrado porque *AB* já se encontra calculado em *ABD/AB*. Outro caso semelhante, por exemplo, é o subcubo *A* - estendido de *AD* - ser igualmente filtrado, porque já se encontra calculado em *ACD/A*.

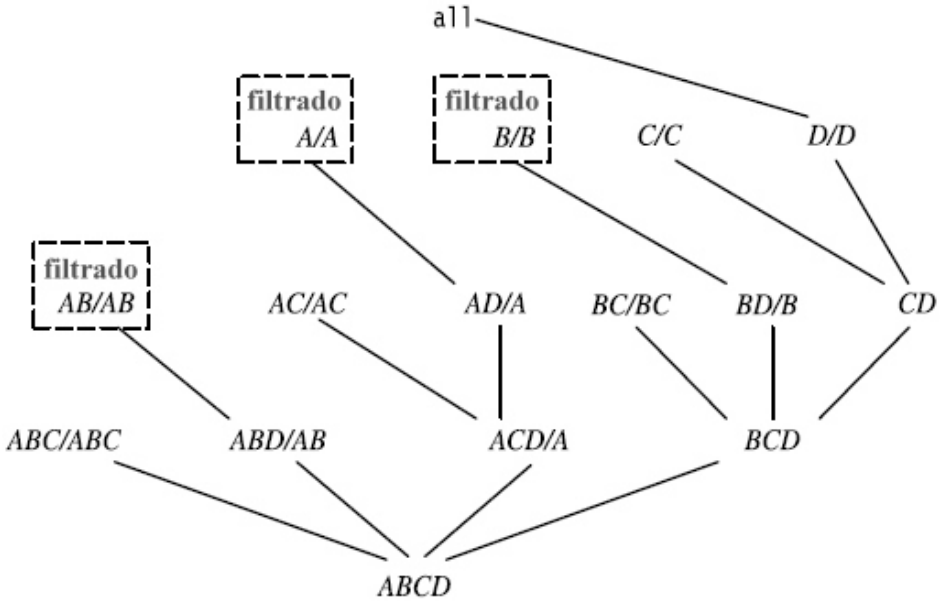


Figura 42. SC: processamento Top-down com expansão Bottom-up das dimensões partilhadas

As dimensões partilhadas, beneficiando duma abordagem *Bottom-up*, permitem realizar a filtragem APRIORI se a medida do patamar mínimo de suporte do cubo icebergue for anti-monótona – tal como o *COUNT(*)*. Nesse contexto, tais células e todas as suas descendentes podem ser filtradas, porque essas são, por definição, mais especializadas (contêm mais dimensões) em comparação com as células das dimensões partilhadas. O número de tuplos cobertos pelas células descendentes será menor ou igual ao número de tuplos cobertos pelas dimensões partilhadas. Portanto, se o valor agregado numa dimensão partilhada falhar a condição icebergue, as células descendentes não a poderão satisfazer também. Para melhor compreender o *modus operandi* do STAR-CUBING é necessário explicitar mais alguns conceitos, nomeadamente, os de *cuboid tree*, *star-nodes* e *start-trees*. As árvores são usadas para representar subcubos individuais. A imagem seguinte mostra um fragmento da *cuboid tree* do cubóide base *ABCD*. Cada nível da árvore representa uma dimensão, e cada nó representa o valor de um atributo. Cada nó é composto por quatro campos: o "valor do atributo", o "valor da agregação", os "apontadores" para possíveis descendentes e o apontador para possíveis nós irmãos. Os tuplos são inseridos um a um na árvore de representação do subcubo (o caminho da raiz em direcção a um nó superior representa um tuplo).

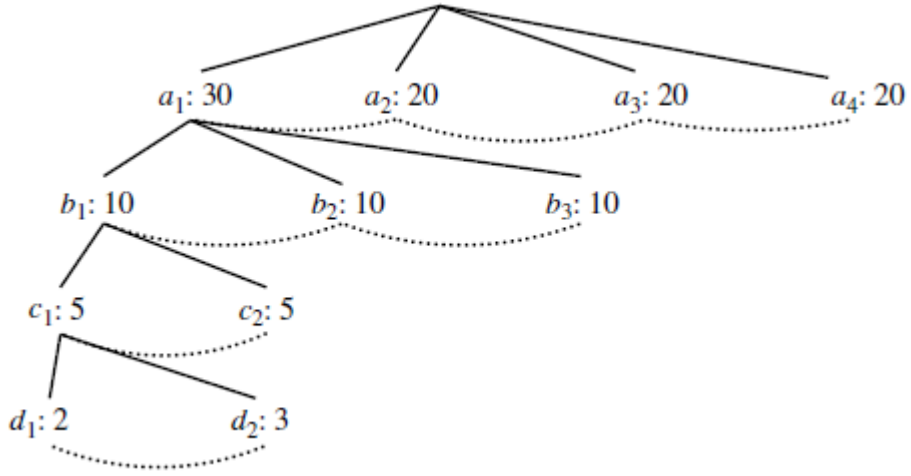


Figura 43. Um fragmento da árvore dum subcubo base

Se o agregado dimensional único associado a um valor do atributo *p* não satisfazer a condição icebergue, é inútil distinguir esses nós no cálculo do cubo icebergue. Então, o nó *p* pode ser substituído por * para que a árvore do subcubo possa ser ainda mais comprimida, de forma a

ganhar espaço de armazenamento, ou seja, o nó *p* num atributo *A* é um *star-node* se o agregado dimensional único em *p* não satisfazer a condição icebergue; caso contrário, *p* é um *non-star-node*. Uma árvore de um subcubo que é comprimida recorrendo a *star-nodes* é denominada por *star-tree*. Veja-se agora o exemplo de construção de uma *star-tree*: considere-se quatro dimensões *(A, B, C, D)* com cardinalidades *(2, 4, 4, 4)* e cinco tuplos, representados na tabela 5(a). Suponha-se que o patamar de suporte mínimo definido é igual ou superior a dois, na condição icebergue. Se assim for, apenas os atributos a_1, a_2, b_1, c_3 e d_4 irão satisfazer a condição. Os agregados do subcubo *1-D* para todos os atributos são indicados na tabela 5(b).

Todos os outros valores estão abaixo do limiar definido e por isso tornam-se *star-nodes*. Ao extenuar-se os *star-nodes*, o resultado comprimido é mostrado na tabela 5(c). Para iniciar-se a construção da *cuboid tree* utiliza-se a tabela comprimida. A *star-tree* resultante é mostrada na figura 44. Para ajudar a identificar quais os *star-nodes*, a *star table* é construída para auxiliar cada *star-tree* (apenas os *star-nodes* são mostrados nessa estrutura). Ao removerem-se os *star-nodes*, a árvore representa uma compressão dos dados originais o que é um bom indicador para salvaguardar o espaço de armazenamento. Contudo, continua a custar tempo de processamento para procurar os nós ou tuplos de uma árvore (esta situação pode ser optimizada, ordenando alfabeticamente os nós para cada dimensão; os *star-nodes* devem ser os primeiros). Genericamente, os nós são ordenados na ordem *, p_1, p_2, ..., p_n para cada nível.

A	B	C	D	Count(*)
a_1	b_1	c_1	d_1	1
a_1	b_1	c_4	d_3	1
a_1	b_2	c_2	d_2	1
a_2	b_3	c_3	d_4	1
a_2	b_4	c_3	d_4	1

(a)

Dimensão	Count = 1	Count ≥ 2
A	-	a_1(3), a_2 (2)
B	b_2, b_3, b_4	b_1 (2)
C	c_1, c_2, c_4	c_3 (2)
D	d_1, d_2, d_3	d_4 (2)

(b)

A	B	C	D	Count(*)
a_1	b_1	*	*	2
a_1	*	*	*	1
a_2	*	c_3	d_4	2

(c)

Tabela 5. a) tabela do subcubo base, b) Agregados 1ª dimensão e c) Tabela comprimida

Utilizando a *star-tree* gerada com o exemplo anterior (figura 44), é possível iniciar o processo de agregação através da abordagem *Top-down*. O primeiro passo (calcular o primeiro ramo da

árvore) é mostrado na figura 45(a). A árvore mais à esquerda representa a *star-tree* base. Cada valor do atributo é mostrado com o seu valor agregado correspondente. Adicionalmente, os subscritos pelos nós na árvore mostram a ordem de andamento no processamento. As restantes quatro árvores *BCD*, *ACD/A*, *ABD/AB* e *ABC/ABC* são as árvores "filhas" da *star-tree* base e correspondem ao nível 3-D dos subcubos acima do cubóide base (figura 44).

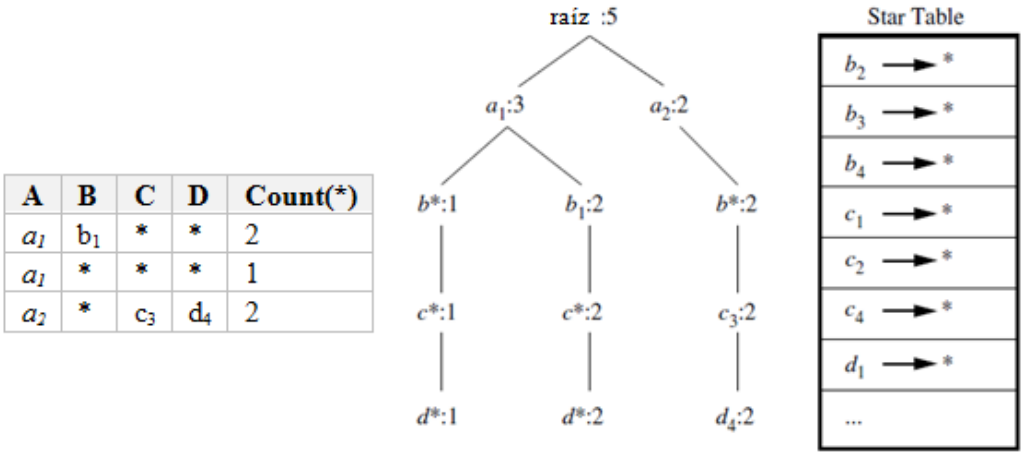
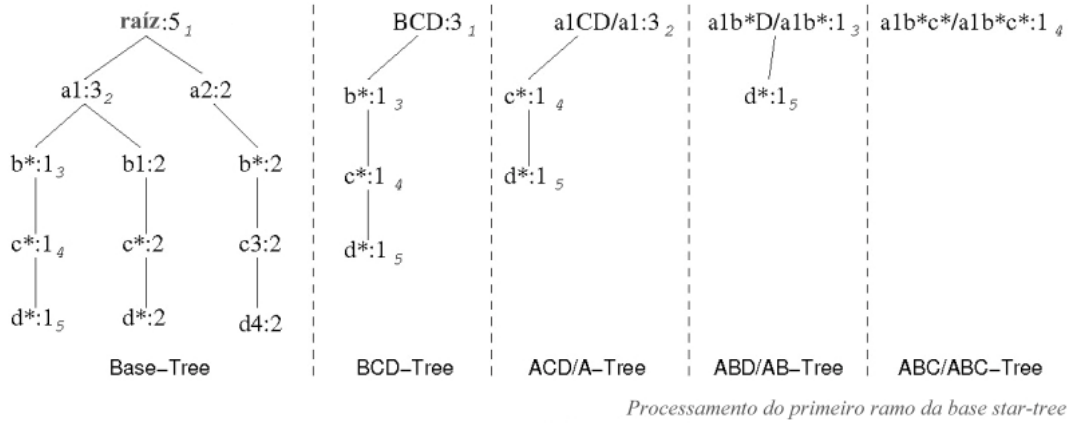


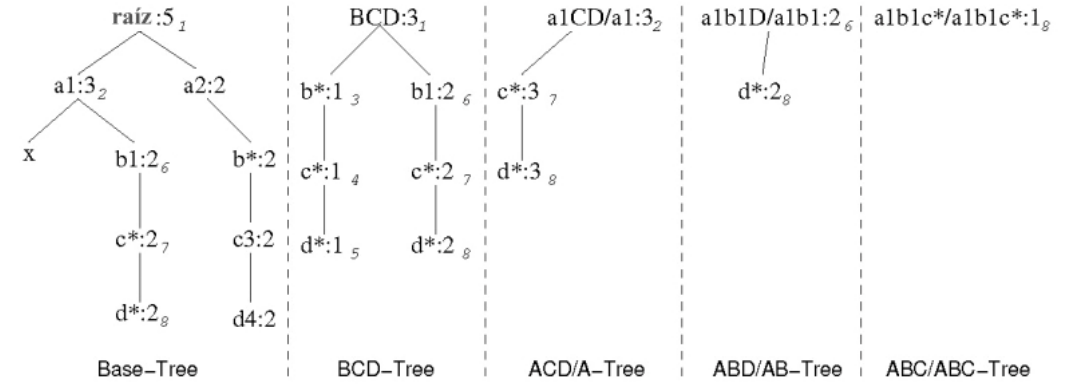
Figura 44. Star-tree e star-table gerada do subcubo base comprimido

Os nós firmados nas árvores descendentes corresponde aos mesmos da *star-tree* base. Por exemplo, quando o algoritmo se encontra no primeiro passo, a raiz da árvore "filha" *BCD* é criada. No passo seguinte, a raiz da árvore "filha" *ACD/A* é criada. No terceiro passo, a raiz da árvore "filha" *ABD/AB* e o nó *b** em *BCD* são criados. Quando o algoritmo atinge o quinto passo, as árvores em memória encontra-se construídas de acordo com o representado na figura 45(a). A mesma lógica é aplicada a todos os ramos da *star-tree* base até o processamento estar concluído (figura 45 (b) e (c)). Um nó tem de satisfazer duas condições de modo a gerar árvores "filhas": a medida do nó tem de satisfazer a condição icebergue e a árvore a gerar tem de incluir, pelo menos, um *non-star-node* (recorde-se que, por definição, todos os *star-nodes* não satisfazem a condição icebergue). No cálculo integral do cubo, se o espaço for denso, o algoritmo SC tem um desempenho equiparável com o *MultiWay* e é muito mais rápido que o BUC ou H-CUBING. Em cubos esparsos, inclusive no cálculo de *iceberg cubes*, o SC obtêm um desempenho superior face aos algoritmo discutido, tornando-se a opção que, até agora, mostra resultados equilibrados em todas as distribuições de dados.



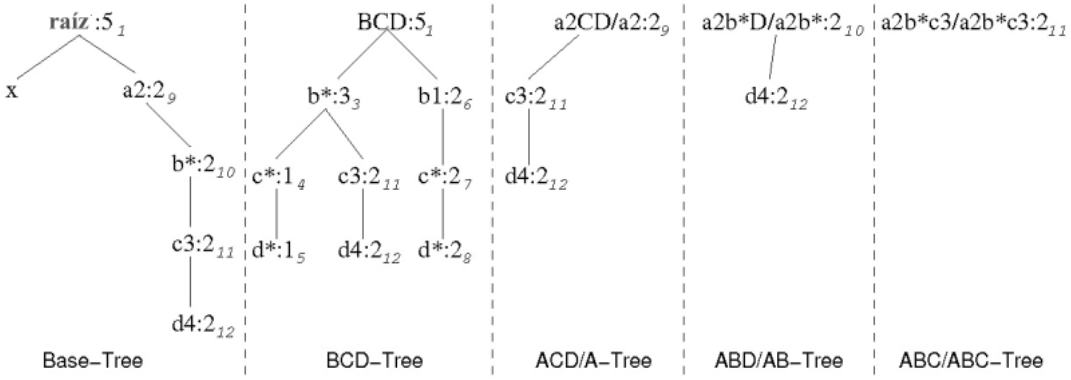
Processamento do primeiro ramo da base star-tree

(a)



Processamento do segundo ramo da base star-tree

(b)



Processamento do terceiro ramo da base star-tree

(c)

Figura 45. Lógica de processamento do algoritmo SC para as sub-árvores correspondentes

O algoritmo STAR-CUBING demonstra, de facto, um desempenho interessante na maioria dos problema de processamento de *iceberg cubes*. No entanto, o seu desempenho começa a degradar-se à medida que as estruturas multidimensionais de dados vão sendo mais esparsas devido, sobretudo, ao custo adicional introduzido pelo cálculo de estruturas em árvore. Por essa razão uma nova abordagem é apresentada, o algoritmo 5) MM-CUBING (MM) [Shao et al., 2004] que opera através da factorização do *lattice* de correlações e de acordo com a frequência dos valores dos atributos. Esta perspectiva difere de todos os outros algoritmos anteriores na medida em que considera a importância da distribuição dos dados, de modo a particionar o *lattice* num sub-espaço denso e em vários sub-espaços esparsos. Estas propriedades tornam este algoritmo deveras flexível em qualquer tipo de distribuição. Recorde-se que, matematicamente, cada (agregação) célula é representada por *(a, b, c, d, ...)* onde *a, b, c* ou *d* podem assumir um valor na dimensão correspondente ou "*" que caracteriza todos os valores. Estas células formam o *lattice* multidimensional de dependências. A cardinalidade de cada dimensão no *lattice* é 1 (o valor "*") mais a cardinalidade da dimensão correspondente dos tuplos de entrada. Em termos de notação, a dimensão dos tuplos de entrada são denotados por *N*, a cardinalidade da *i* dimensão por L_i e o número de tuplos de entrada por *T*. Cada tuplo contribui $\mathbf{2^N}$ células (base e agregados), mas, na realidade, existem em conjunto $\prod_{i=1}^{N}(Li + 1)$ células.

Virtualmente, é possível construir uma tabela "ingénua" onde cada tuplo é uma linha e cada célula, uma coluna. Se o tuplo contribuir para a célula, a intersecção de linha e da coluna é 1, caso contrário é 0. Então o somatório de todos os elementos de uma linha é $\mathbf{2^N}$ e o somatório de todos os elementos numa coluna é o suporte da célula. Se L_i ou *N* for grande, a tabela será muito esparsa. O que é importante notar é o facto de existirem muitas colunas a somar para obter-se um limite menor que o patamar mínimo definido na condição icebergue, sendo o seu cálculo inútil. Esta é uma das principais razões da dificuldade em obter-se um algoritmo óptimo no problema de processamento de *iceberg cubes*. As abordagens anteriores tentam resolver este problema apresentado uma conceptualização do *lattice* numa estrutura em árvore. Seja numa direcção *Top-down* ou *Bottom-up*, qualquer um dos métodos ignora o valor da densidade dos dados. Na realidade, o processamento é baseado numa estrutura fixa. Apesar

de usarem estruturas, umas mais adaptativas e heterogéneas em diferentes dimensões de dados, os valores de uma dimensão com diferentes frequências são sempre tratados da mesma forma, já que são representados por uma única célula na estrutura. De modo a optimizar a eficiência e adaptabilidade de um algoritmo de *iceberg cubing*, é necessário distinguir o tratamento dos valores frequentes dos infrequentes, porque os primeiros tendem a contribuir para identificar as células mais passíveis de serem usadas. Ao contrário da anterior estrutura em grelha, a factorização do *lattice* diferencia as células frequentes das infrequentes, e esses valores são representados por pontos separados. Deste modo, a nova estrutura em grelha tem $\mathbf{3^N}$ nós. Se *M* representar as células frequentes, *I* as infrequentes e * » *all*, um *lattice 3-D* pode ser representado assim: *({M, I, *}, {M, I, *}, {M, I, *})*. Recorde-se que as células infrequentes devem ocorrer pelo menos uma vez no patamar mínimo de suporte definido. Um valor que se verifique abaixo do limiar nunca irá ocorrer numa célula icebergue, devido à sua definição. É desejável partilhar o processamento das células frequentes com *; para as infrequentes, o algoritmo irá filtrá-las o mais cedo possível. Desse modo, é possível factorizar o *lattice* de soluções nos seguintes sub-espaços: *1 ({M, *}, {M, *}, {M, *}), 2 (I, {M, I, *}, {M, I, *}) e 3 ({M, *}, I, {M, I, *}), 4 ({M, *}, {M, *}, I)*. Nenhum dos sub-espaços intersecta com outro e a soma de todos eles representa o *lattice* original. A imagem seguinte demonstra esta notação.

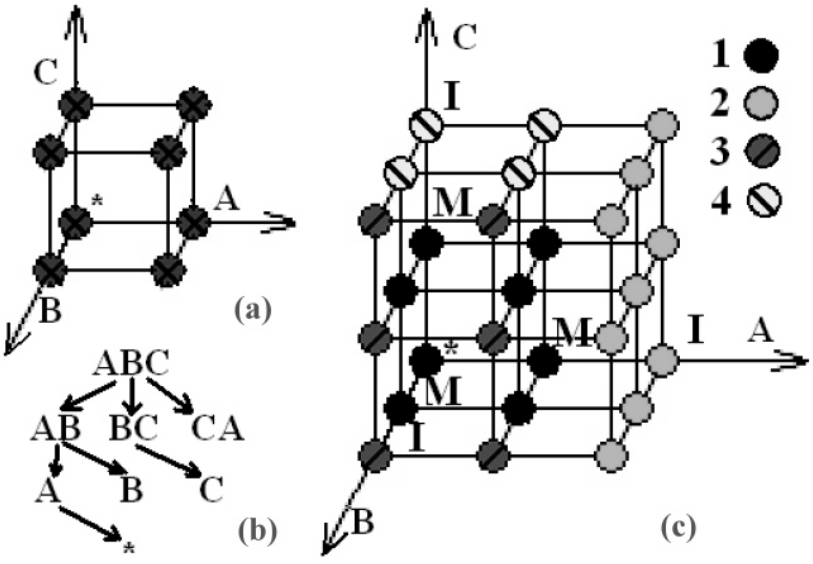


Figura 46. a) Lattice comum, b) árvore lattice comum e c) factorização do lattice espacial [Shao et al., 2004]

Veja-se a figura 46(c): os valores de cada dimensão no primeiro sub-espaço (1) são aqueles mais frequentes ou *. As células neste quadrante têm provavelmente um maior suporte na condição icebergue (sub-espaço denso). As células no sub-espaço (2) são menos frequentes, pelo que o suporte à condição icebergue não será tão grande. Uma situação semelhante acontece no sub-espaço (3) mas com uma diferença: o valor da primeira dimensão só pode ser frequente ou *, caso contrário, este sub-espaço iria colidir com o (2). O último sub-espaço (4) é semelhante ao (3) à excepção das duas dimensões com valor frequente ou *. À excepção do primeiro, todos os restantes sub-espaços são considerados esparsos. Note-se que apesar de existir apenas um espaço denso (o que não é conforme, ao comparar-se com base de dados reais) as chamadas recursivas dos restantes irão explorar outros sub-espaços densos por dentro deles. Na prática, o algoritmo MM efectua os seguintes passos: a) antes da factorização atrás mencionada, é efectuada a contagem da frequência dos valores de cada dimensão. De seguida, estes são ordenados descendentemente para determinar os que têm maior e menor frequência. Estas acções são asseguradas por um algoritmo *Count-and-Sort* [Shao et al., 2004]. No passo seguinte b) é efectuada a factorização já mencionada. Uma nota apenas: os valores mais frequentes são determinados recorrendo a uma heurística que representa o valor total de "trabalho" efectuado pelas agregações simultâneas: *work_done* = $T \; x \; \prod_{i=1}^{N}(1 + sum[i]/T)$ onde *T* é o número de tuplos, *sum[i]* o número de tuplos com maior frequência na dimensão *i* e *N* o número de dimensões. A tabela de agregações simultâneas pode ser calculada através de: $\prod_i(1 + MC[i])$ onde *MC[i]* representa o número de valor mais frequentes no conjunto das *i* dimensões.

Completada a factorização do espaço de soluções, inicia-se c) o processamento do espaço denso recorrendo a agregações simultâneas. O cálculo do espaço denso considera apenas as células frequentes e os * (ao contrário do *MultiWay*, que processa todo o *lattice*). As agregações são armazenadas num vector e calculadas por uma lógica dinâmica de programação. Assim que as agregações foram completadas, é possível obter todas as células com um suporte não inferior ao limiar definido na condição icebergue. Em último lugar, inicia-se d) o cálculo dos espaços esparsos recorrendo a chamadas recursivas a eles próprios e beneficiando das vantagens da filtragem APRIORI. Os autores revelam que o algoritmo MM

tem um desempenho superior em quase todo o tipo de distribuição de dados, ultrapassando na maioria dos casos, os algoritmos anteriormente analisados. Contudo, o consumo excessivo de memória é um dos seus principais problemas, devido sobretudo à complexidade das chamadas recursivas (a pior situação para o MM será mesmo quando os valores mais frequentes de uma dimensão estiverem sempre correlacionados com os valores menos frequentes de uma outra dimensão). Mesmo assim, o algoritmo MM-CUBING supera largamente em desempenho os algoritmos BUC e STAR-CUBING em distribuições ditas uniformes (parcialmente densas ou esparsas, medianamente enviesadas). Note que a maioria soluções propostas até ao momento para o problema do processamento de *iceberg cubes* operam numa perspectiva centralizada, à excepção da evolução paralela do BUC [Wagner & Yin, 2001] que defende uma solução paralela e de distribuição de carga. Esse trabalho motivou largamente o desenvolvimento do próximo algoritmo a discutir, vocacionado, sobretudo, para ambientes de cálculo distribuído, com o objectivo de aumentar o desempenho no processamento de *iceberg querys* complexas.

Determinado pelo sucesso alcançado do STAR-CUBING e das abordagens paralelas atrás mencionadas, o algoritmo 6) PnP (*Pipe n'Prune*) proposto por [Chen et al., 2005] revela-se uma potente solução híbrida para o processamento paralelo de *querys* icebergue complexas e de grande dimensão. As inovações introduzidas pelo PnP revelam-se através da capacidade de integrar rigorosamente os métodos de agregação *Top-down* e *Bottom-up* (com filtragem APRIORI) para além da sua destacada eficiência face aos seguintes cenários: a) processamento sequencial de *querys* icebergue; b) processamento de *querys* icebergue em memória externa; c) processamento de *querys* icebergue em ambientes paralelos e distribuídos. Os autores propõem um novo operador denominado *PnP operator*. Para um determinado *group-by v*, o operador efectua os seguintes passos: constrói todos os group-by *v'* (prefixo de *v*) através de uma única operação de ordenação/leitura (*piping*[21]) [Sarawagi et al., 1996] pressupondo a filtragem do cubo icebergue. Posteriormente, utiliza esses *group-by* do

[21] Em ciências e tecnologias de informação, ***piping*** indica um conjunto de elementos de processamento de dados ligados em série, onde o resultado de um elemento é o *input* do elemento seguinte. Os elementos de uma *pipeline* são muitas vezes executados em paralelo ou em intervalos de tempo; nesse caso, recorre-se à utilização de *buffers* entre os elementos para agilizar o processamento.

prefixo para efectuar o processamento *Bottom-up*. O uso da propriedade APRIORI permite filtrar os novos *group-by*, que são os pontos de partida para outras operações de *piping*. Um exemplo de operador PnP é mostrado na figura 47(a). Este operador é aplicado recursivamente até que o conjunto de *group-by* do cubo icebergue tenham sido gerados. Na figura 47(b) é possível ver também um exemplo de árvore PnP, que representa todo o processo de cálculo de um *iceberg cube* de 5 dimensões.

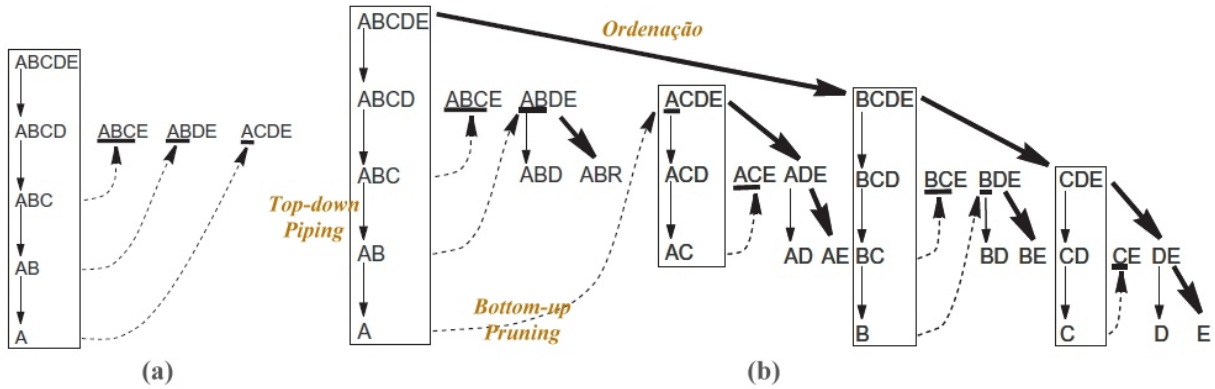


Figura 47. Algoritmo PnP onde a) operador PnP e b) árvore PnP [Chen et al., 2005]

A explicação do cenário a) será efectuada com base numa *iceberg querys* de 5 dimensões, ilustrada na figura 47(b). O algoritmo inicia com a rotina *PnP-1 (R, ABCDE, ∅)* onde *ABCDE* são os *group-by* que contêm todas as dimensões de *R*, ordenadas decrescentemente, por cardinalidades. Isto irá resultar, em primeiro lugar, na fila *ABCDE-ABCD-ABC-AB-A* para depois serem criadas as versões filtradas de *ABCE*, *ABDE* e *ACDE* para as operações de *piping* subsequentes. A tabela 6 mostra a execução completa para o conjunto de dados base *R* indicado na primeira coluna da mesma. Os *buffers*[22] b[5] … b[1] representam o resultado das operações de *piping* enquanto que R_3 ... R_1 mostra os resultados da filtragem APRIORI. Note que o operador PnP utiliza apenas um única passagem pelo conjunto de dados. As linhas horizontais da tabela 6 indicam os casos onde a agregação ou filtragem ocorre. De seguida, a chamada recursiva inicia o operador PnP para os *group-by ABCE*, *ABDE* e *ACDE*. O prefixo encontra-se a sublinhado na figura 47(b) para os fragmentos de *ABCE*, *ABDE* e *ACDE*

[22] Em ciências e tecnologias de informação, um ***buffer*** representa uma região temporária da memória principal, usada para reter os dados de um processo, enquanto são movidos de um lado para o outro.

respectivamente. Representa as chamadas recursivas para os elementos já processados nas operações de *piping*. A chamada recursiva inicia o operador PnP para o *group-by BCDE* que desencadeia o processamento do cubo icebergue para todos os grupos que não contenham *A*. O resultado final é mostrado na figura 47(b).

R **ABCDE**	*b*[5] **ABCDE**	*b*[4] **ABCD**	*b*[3] **ABC**	R_3 **ABCE**	*b*[2] **AB**	R_2 **ABDE**	*b*[1] **A**	R_1 **ACDE**
11111 1	11111 1			filtrado		1111 2		1111 1
11112 1	11112 1	1111 2				1112 1		1112 1
11122 1	11122 1	1112 1	111 3			1122 1		1122 1
11211 1	11211 1	1121 1	112 1	filtrado	**11 4**		**1 4**	1211 1
21111 1	21111 1	2111 1		filtrado		filtrado		filtrado
21121 1	21121 1							
21122 1	21122 1	2112 2	211 3		21 3		2 3	
31111 1	31111 1	3111 1		filtrado		3111 1		3111 1
31121 1	31121 1					3121 2		3121 1
31122 1	31122 1	3112 2	311 3			3122 1		3122 1
31221 1	31221 1			filtrado		3123 1		3221 1
31223 1	31223 1	3122 2	312 2		**31 5**		**3 5**	3223 1
41111 1	41111 1	4111 1	411 1	filtrado	41 1	filtrado		4111 2
42111 1	42111 1			filtrado		filtrado		4112 1
42112 1	42112 1	4211 2	421 2		42 2			4121 1
43121 1	43121 1	4312 1	431 1	filtrado	43 1	filtrado	**4 4**	

Tabela 6. Tabela de processamento PnP para ABCDE

Sobre o cenário b) uma vez que o PnP efectua a ordenação dos elementos, será simples estender o algoritmo para ser processado em memória externa. Em comparação com acções evidenciadas no cenário a) aqui todas as operações de ordenação são endereçadas para a memória externa. Deve-se, no entanto, ter algum cuidado com a análise das operações de agregação e *pruning*, já que pode ocorrer o esgotamento dos *buffers*. Outra diferença em relação ao primeiro cenário está relacionada com a invocação das chamadas recursivas. Em memória externa, é necessário guardar a tabela de agregações num ficheiro em disco. Cada linha será, depois, invocada em ciclo para serem aplicadas as chamadas recursivas. Repare-se que estas operações são independentes e podem ser sujeitas a alguma latência de disco

"escondida", originada por sobreposição do processamento e por operações de *input/output*. De modo a utilizar esse efeito convenientemente, o algoritmo dispõe de um gestor de *I/O* que resulta numa significativa melhoria do desempenho. No caso do terceiro e último cenário c) é demonstrado na figura 48(a) como o algoritmo PnP pode ser paralelizado de modo a ser executado num ambiente distribuído. Um ambiente multi-processador consiste em *p* processadores P_0 ... P_{p-1} cada um com memória e disco próprios, ligados por uma rede ou comutador. Assume-se como *input,* os dados base de *n-dimensões R* armazenados numa tabela de *n* linhas que são distribuídos por *p* processadores, como indicado na figura 48(b). Cada *group-by* do *output* (o cubo icebergue) será também particionado e distribuído por *p* processadores. Este processo consiste em listrar a tabela em cada *p* disponível. Quanto esta acção está concluída, o acesso a um *group-by* pode ser feito com a máxima banda de *I/O* disponível através da paralelização do acesso a *p*.

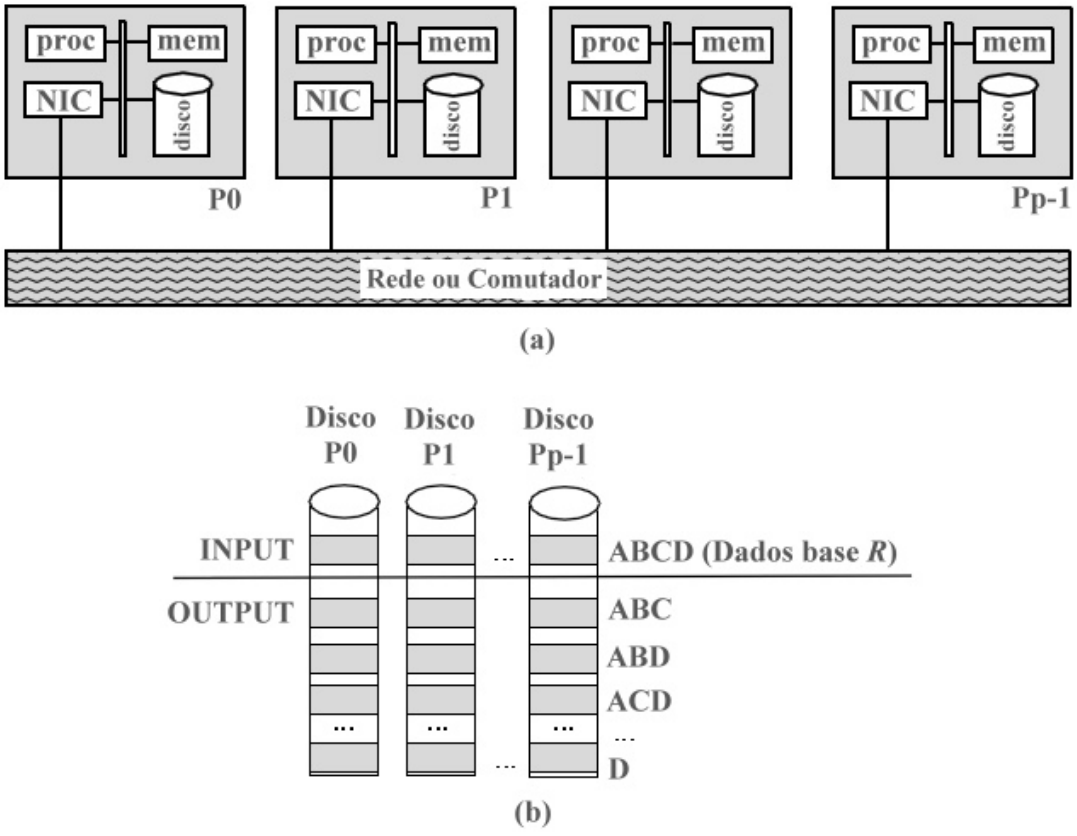


Figura 48. a) ambiente distribuído com b) listras sobre cada disco individual de p [Chen et al., 2005]

A paralelização do algoritmo PnP permite, da árvore ilustrada na figura 47(b), criar uma "floresta" obtida através do particionamento da árvore PnP em várias *t trees*, uma para cada dimensão. Ao iniciar-se na árvore T^1 de R, esta é listrada sobre os p processadores, onde P_1 guarda $T^1_i = R_i$ e executa em cada processador P_i o *input* T^1_i. Esta acção permite criar a primeira árvore na floresta PnP. De seguida, é calculada, em cada processador P_i, a árvore T^2_i a partir de T^1_i. As agregações da segunda são calculadas, removendo a dimensão da primeira, de modo a eliminar duplicados (recorrendo a uma ordenação sequencial). Assim, é possível executar em cada processador P_i a árvore T^2_i e adicioná-la à floresta PnP. Este processo é iterado $\times$ vezes até que todos os *group-by* estejam construídos. Este tipo de arquitectura dispõe de um consolidado balanceador de carga que permite minimizar o ónus das comunicações e fazer uso de uma gestão plena das entradas e saídas entre máquinas. O algoritmo PnP proporciona ganhos substanciais no cálculo de conjuntos de dados que são difíceis de lidar pelos algoritmos sequencias, já mencionados. Mais importante ainda é que o PnP é escalável e, portanto, beneficia do poder de processamento assim que mais processadores forem adicionados ao ambiente de cálculo analítico, resolvendo as situações indicadas em [Wagner & Yin, 2001]. Na generalidade, este algoritmo é muito eficiente no cálculo de cubos icebergue sobre dados reais, adaptando-se particularmente em situações com elevado número de registos, dimensionalidade e cardinalidade.

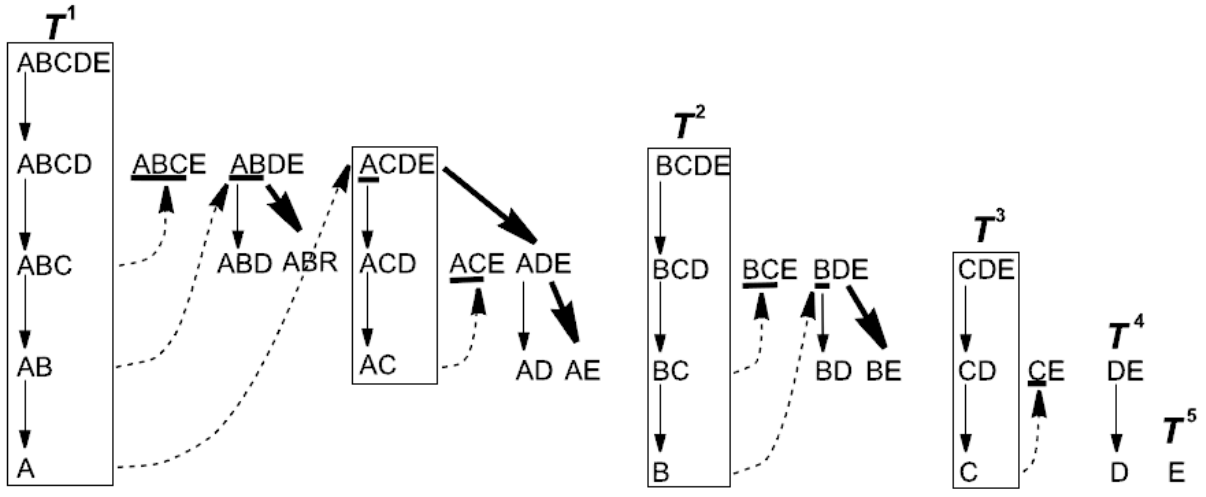


Figura 49. A floresta PnP, com a identificação das sub-árvores Ti geradas [Chen et al., 2005]

Em [Xin et al., 2006] é proposto um outro algoritmo de processamento de cubos que utiliza os algoritmos STAR-CUBING e MM-CUBING denominado 7) C-CUBING. Note-se que este algoritmo não é particularmente dirigido para o cálculo de cubos icebergue, mas sim para o processamento integral do cubo, recorrendo aos métodos de redução de dados baseados em *closed cubes* [Lakshmanan et al., 2002]. No entanto, optou-se por divulgar as suas características fundamentais pela importância que os contributos dos algoritmos icebergue estendem a outros métodos de processamento de cubos no processo de materialização de vistas. Na sua essência, este algoritmo efectua o cálculo das agregações, no processamento de *closed cubes*, recorrendo a uma nova medida algébrica denominada por *closedness* [Xin et al., 2006]. A integração de *iceberg cubes* e *closed cubes* origina o conceito *closed iceberg cubes*. O problema do processamento desta nova perspectiva consiste em calcular todas as células *closed* que satisfazem um determinado patamar mínimo de suporte, definido na condição icebergue.

A abordagem *closed* é indicada na literatura como um método eficiente para cálculo integral de um cubo de dados sob a forma de estruturas condensadas, *dwarf* e *Qucient* [Lakshmanan et al., 2002], [Lakshmanan et al., 2003], [Wang et al., 2002], [Sismanis et al., 2002], [Sismanis et al., 2004]. Neste tipo de abordagem existem duas operações frequentes: a verificação de *closedness* (*c-checking*) e a filtragem de células *not-closed* (*c-pruning*). Com um custo de processamento reduzido, a informação *closed* de uma célula pode ser agregada numa medida e a verificação (*c-checking*) pode ser efectuada simplesmente para testar esse valor. A adaptação de algoritmos icebergue revela-se eficiente no cálculo de espaços dispersos e permite definir um patamar de utilidade na selecção de células *closed*, de modo a optimizar os algoritmos tradicionais que recorrem a este método. Nesse contexto, os autores propõem três lógicas baseadas nessa tecnologia: C-CUBING (MM), C-CUBING (SC) e C-CUBING (SA). Todos eles superam as abordagens de cálculo de *closed cubes* anteriores. O C-CUBING (MM) mostra-se particularmente eficiente quando a filtragem APRIORI domina o cálculo (herdada do cálculo icebergue). Por outro lado, os algoritmos da família STAR (BUC*, MultiWay* e SC) têm melhor desempenho quando a operação *c-pruning* é significativa.

Recentemente, foram propostas mais duas soluções para processamento de *iceberg cubes*: 8) IX-CUBING [Jian et al., 2007] dirigido unicamente para o processamento optimizado de *iceberg querys* sobre dados no formato XML (*extensible markup language*) e 9) MT-CUBING [Li et al., 2008] que integra uma solução *multi-tree* híbrida, que permite processar estas estruturas através de algoritmos *Top-down* e *Bottom-up*. Veja-se de imediato, a primeira. É, de facto, notória a crescente utilização do formato XML proposto pelo W3C[23], com o objectivo de codificar e uniformizar diferentes tipos de dados, de modo a facilitar o seu entendimento e manipulação em ambientes distintos. Sendo o XML um formato semi-estruturado, dados do mesmo tipo podem ser representados de diferentes maneiras. Além disso, os dados em falta são também uma situação comum. Mesmo assim, os autores propõem o algoritmo IX-CUBING para representar operações OLAP baseadas em dados XML, através da construção de um *iceberg cube* baseado num conjunto de dados XML de modo a responder eficientemente às interrogações agregadas. Na sua essência, o trabalho proposto inicia-se pela extensão dos cubos, baseados em dados relacionais, para XML. Apesar de não ser uma acção trivial, são propostos vários métodos para lidar com os dados irregulares ou incompletos. Posteriormente, é investigado como é que os *IX-cubes* podem ser usados para responder a interrogações analíticas. Neste campo, é também proposto um método baseado em indexação *B+-tree* para acelerar as *querys* de consulta às agregações.

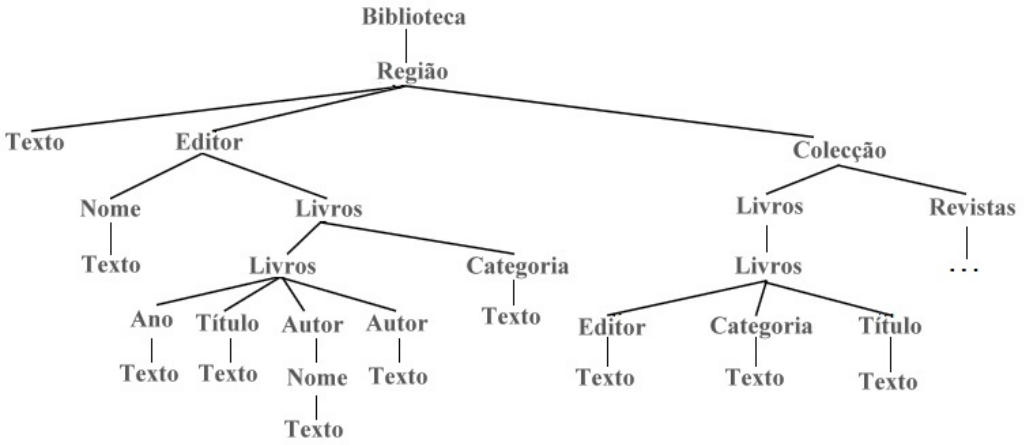


Figura 50. Exemplo árvore XML: dados de um tipo representados de 2 maneiras diferentes

[23] *World Wide Web Consortium*

Assim, dada uma árvore XML, um cubo icebergue IX é especificado por 3 tuplos ‹E_{xcube}, M_{xcube}, D_{xcube}› onde E_{xcube} é uma entidade, M_{xcube} uma medida e D_{xcube} um conjunto de dimensões. E_{xcube} é dado por um caminho que acaba numa posição de interesse na árvore. Um nó na estrutura XML que seja igual a E_{xcube}, denomina-se o nó de contexto. Uma dimensão é definida por 2 tuplos ‹*NomeD, Caminhos*› onde *NomeD* representa o nome da dimensão e *Caminhos*, o conjunto de direcções onde o ponto de partida é o nó de contexto. Uma medida é definida por 3 tuplos ‹*FuncaoAgg, Caminhos, min_sup*› onde *FuncaoAgg* representa a função de agregação, *Caminhos* é o conjunto de direcções a serem consideradas para agregação e *min_sup*, o patamar de suporte mínimo da *query* icebergue. Naturalmente, só os agregados que ultrapassarem este valor serão armazenados no *IX-cube*.

Entidade alvo	//Livros
Medida	‹ COUNT, {.}, 25000 ›
Dimensões	
Nome	Caminhos
Editor	{C_{e1} = ./editor/texto(), C_{e2} =./../../../editor/@nome/texto()}
Categoria	{C_{c1} = ./categoria/texto(), C_{c2} = ./../categoria/texto()}
Autor	{C_{a1} = ./autor/texto(), C_{a2} = ./autor/nome/texto()}

Tabela 7. Tabela de especificação de um IX-cube

Este algoritmo usa uma estrutura em árvore para armazenar um *IX-cube*. Dada uma árvore XML e a definição de um *IX-cube* ‹E_{xcube}, M_{xcube}, D_{xcube}›, o *IX-cube* resultante é uma árvore directamente identificada por T_{xcube} = ‹*raiz, E, V,* l_E› onde *raiz* representa a origem do *IX-cube*, *V* o conjunto de nós e *E* o conjunto de arestas do T_{xcube}. Cada aresta tem três atributos: *val*, que corresponde a um valor válido da dimensão ou □ (val.raiz = □); *ins*, o conjunto de nós de contexto; e *agg*, referindo-se aos agregados processados nos valores das medidas dos nós de contexto em *ins*. Por último, l_E representa a função que rotula as arestas em *T*, relacionando-as com o nome da dimensão. A figura 51 mostra o *IX-cube* calculado, com base

na árvore XML mostrada na figura 50, dada a tabela 7 de especificação. Como se pode observar, os prefixos comuns são partilhados na árvore de modo a ganhar espaço de armazenamento. Para além disso, este esquema permite que os agregados dos nós internos possam ser calculados, para que a filtragem à cabeça possa ser aplicada, baseada no patamar mínimo de suporte da condição icebergue. As linhas pontuadas particionam os agregados para as diferentes dimensões. Separam igualmente um *IX-cube* em diversas camadas, uma para cada dimensão (ou a conceptualização da hierarquia de uma dimensão). Existe uma camada de topo para o *"all"* e uma de base, onde se encontram as "folhas" da árvore.

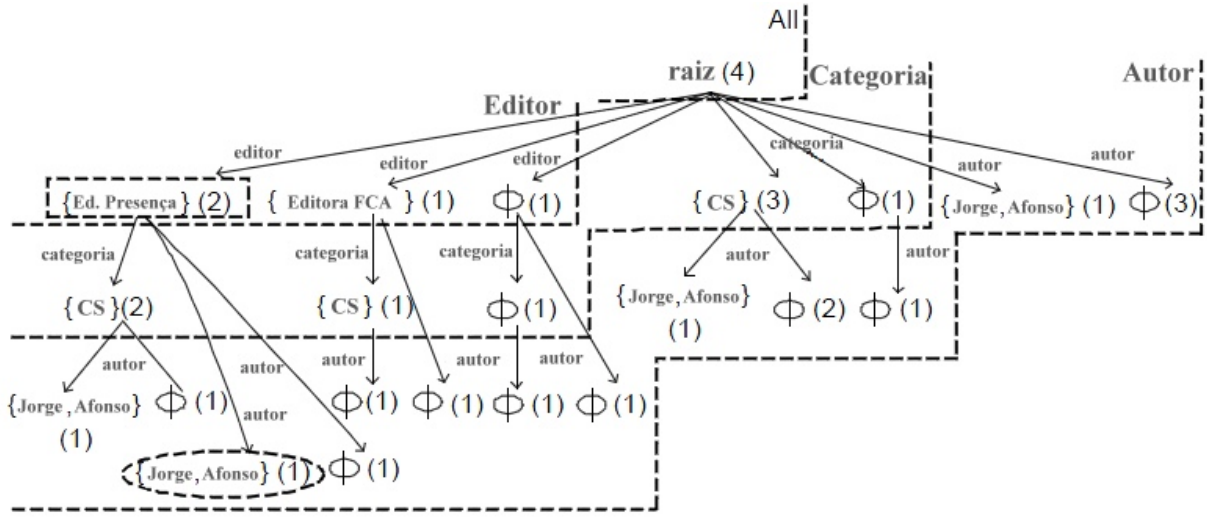


Figura 51. Representação do processamento do IX-cube na árvore XML

Este tipo de estrutura pode ser usada para responder a qualquer interrogação OLAP. Assumindo que este algoritmo materializa todos os subcubos, é possível gerar resultados ao pesquisar cada uma das camadas do *IX-cube*. Através de operações de *roll-up* e *drill-down* é possível navegar nos ramos do cubo. Para acelerar o tempo de resposta das interrogações é proposto um método de indexação *CubeIndex*. O cálculo do *IX-cube* segue uma abordagem *Bottom-up* que inclui as seguintes etapas: o *input* consiste numa árvore XML *T*, a definição do *IX-cube* $\langle E_{xcube}, M_{xcube}, D_{xcube} \rangle$, τ e *min_sup*, onde τ representa as dimensões e medidas válidas e *min_sup*, o patamar de suporte, para o cálculo das *querys* icebergue. É usado □ para marcar os valores em falta das dimensões correspondentes. Depois, para cada nó de contexto, a sua medida e o valor das restantes dimensões formam um tuplo. Finalmente, é aplicado um

algoritmo semelhante ao BUC para processar o *IX-cube*. Os resultados são depois indexados para rápido acesso, quando necessários. Outra das mais recentes abordagens relacionada com o processamento de *iceberg cubes* foi proposta por [Li et al., 2008] denominada 9) MT-CUBING (*multi-tree*) que visa colmatar muitas das limitações das abordagens *Top-down* e *Bottom-up* identificadas ao longo desta secção. Essencialmente, os autores propõem uma abordagem híbrida que integra essas duas tecnologias. A gestão global é efectuada por uma aproximação *Top-down* para beneficiar do cálculo partilhado. Através do processamento das ordenações no sentido oposto da abordagem *Top-down*, este algoritmo permite beneficiar da filtragem APRIORI. A diferença em relação ao BUC é que este algoritmo é capaz de fazer *pruning* sem processar qualquer partição. Baseia-se numa estrutura em árvore com prefixo especializado denominada *attribute-partition tree* (*AP-tree*). Esta estrutura é composta por nós de partições e atributos para além de facilitar a ordenação e a filtragem rápida das agregações. Os autores revelam que este algoritmo é mais rápido em qualquer situação em relação ao BUC, no processamento dos mesmos *iceberg cubes*.

Do estudo efectuado, o algoritmo 10) CT-CUBING (*cross table*) [Cho et al., 2005] aparece em último lugar pois utiliza uma abordagem diferente face a todas as soluções atrás descritas. De facto, todas elas assumem que os dados encontram-se armazenados numa única estrutura base, que depois é utilizada para o cálculo de *iceberg cubes*. Contudo, na prática, um *Data Warehouse* é organizado maioritariamente por múltiplas tabelas derivadas de um modelo multidimensional [Kimball, 1996]. Tendo em conta o conhecido *trade-off* tempo e espaço, a materialização de uma única estrutura universal através das junção de diferentes tabelas incorre em custos elevados ou mesmo inacessíveis num ambiente de *Data Warehouse* real. Por isso, os autores do CT-CUBING (CTC) propõem investigar o cálculo de *iceberg cubes* através do acesso a diferentes tabelas do *Data Warehouse*. Surpreendentemente, as avaliações efectuadas mostram que o algoritmo pode ainda ser mais eficiente a nível de tempo e espaço em relação à solução da tabela única materializada, utilizada pelas outras abordagens. A tabela 8 representa duas estruturas do *Data Warehouse* separadas (*F* e *D*) e uma estrutura de dados *B* universal única. O método tradicional seria processar o cubo icebergue recorrendo apenas à tabela única $B = F \times D$. Contudo, a materialização da mesma ocorre em custos

relacionados com o tempo de processamento e espaço de armazenamento. Veja-se, então, um exemplo de cálculo de *iceberg cubes* recorrendo às tabelas (*F e D*) sem necessidade de materializar e consultar a tabela *B*. Logo à partida, para qualquer combinação de atributos na tabela *D*, o valor agregado é $m = aggr(\{m_1, ..., m_l\})$. Portanto, se *m* satisfazer a condição icebergue, então todas as combinações de atributos de *D* são células icebergue. Então, processa-se essas células usando apenas a tabela *D* que contém apenas um tuplo. No método tradicional, seria preciso usar vários tuplos na tabela *B* para calcular as células icebergue. Note-se que, para qualquer célula icebergue que envolva atributos da tabela *F*, o valor agregado pode ser calculado através dela própria. No método tradicional, seria necessário calcular as células icebergue utilizando uma tabela universal *B* bem mais ampla.

K	A_1	$\cdots$	A_n	M
k	$a_{1,1}$	$\cdots$	$a_{1,n}$	m_1
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
k	$a_{l,1}$	$\cdots$	$A_{l,n}$	m_l

Tabela F

K	B_1	$\cdots$	B_m
k	b_1	$\cdots$	b_m

Tabela D

A_1	$\cdots$	A_n	K	B_1	$\cdots$	B_m	M
$a_{1,1}$	$\cdots$	$a_{1,n}$	k	b_1	$\cdots$	b_m	m_1
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$a_{l,1}$	$\cdots$	$a_{l,n}$	k	b_1	$\cdots$	b_m	m_l

Tabela base universal B = F x D

Tabela 8. Tabelas exemplo para cálculo de iceberg cubes [Cho et al., 2005]

O algoritmo CTC não necessita de materializar a tabela *B*; em vez disso, opera tendo em conta três passos: em primeiro lugar, o algoritmo propaga as chaves e as medidas de cada tabela dimensão; Depois, o cubo icebergue local em cada tabela é processado; no final, o *iceberg cube* global é derivado de todos os locais gerados. Estes procedimentos permitem comprovar a efectividade do algoritmo CTC sobre a abordagem tradicional de uma única tabela base universal materializada, revelando-se particularmente eficiente e escalável em *Data Warehouses* de grandes dimensões. Note-se que qualquer célula icebergue que respeite uma condição icebergue, deve ser projectada nas tabelas de facto e dimensões localmente, na medida em que os cubos icebergue locais são calculados directamente a partir dessas estruturas (o cálculo dos *iceberg cubes* locais é dinâmico e pode ser efectuado recorrendo a

qualquer algoritmo de cálculo, por exemplo, o BUC). Note que o CTC nunca efectua junções de duas ou mais tabelas. Em vez disso, apenas conduz as interrogações *group-by* em cada atributo chave, de modo a propagar o agregado para a dimensão correspondente. Adicionalmente, ainda recorre a técnicas de indexação para melhor o desempenho. Quando existem múltiplas tabelas de dimensão é muito mais eficiente efectuar a propagação dos agregados do que efectuar junções de tabelas e materialização de uma única tabela base. Depois dos cubos serem propagados, são minerados independentemente, recorrendo à mesma condição icebergue. Note-se que também não existe necessidade de unir tabelas de factos e dimensões para gerar uma célula icebergue global. Em vez disso, as células locais são interligadas pelas suas "assinaturas", guardadas num vector *bitmap* e indexadas, o que permite evitar redundâncias no processamento (podem ser armazenadas em disco, o que permite salvaguardar espaço de memória). Na abordagem tradicional pode haver necessidade em procurar a mesma porção da tabela para obter a célula icebergue local; muitas vezes, naturalmente, para diferentes células icebergue locais.

Depois, as células icebergue que envolvam atributos em múltiplas tabelas de dimensão são derivadas dos icebergues locais. As operações de junção de células icebergue locais são asseguradas por uma *h-tree* [Xin et al., 2003] na medida em que é necessário pesquisar a tabela de factos uma vez para cada célula icebergue local. Esta acção é assegurada usando apenas as chaves estrangeiras de cada uma das tabelas de factos. A complexidade espacial da *h-tree* no CTC é de $O(kn)$ onde k é o número de tabelas de dimensão e n o número de tuplos das tabelas de facto. Ao contrário das abordagens tradicionais, o CTC não necessita de armazenar todas as tabelas em memória para obter o melhor desempenho. Em vez disso, carrega-as uma a uma. As células icebergue locais podem ser mantidas e indexadas em disco. Todavia, o CTC pode enfrentar constrangimentos de memória quando armazena a *h-tree* de cada uma tabela de factos. No entanto, esta árvore tende a ser menor do que a tabela de factos e muito menor que a tabela base única. Mesmo assim, se a *h-tree* for muito extensa para caber em memória, podem ser usadas as técnicas de gestão de disco discutidas em [Xin et al., 2003]. Os resultados desta solução mostram que a aplicação do algoritmo CTC é mais eficiente e escalável em comparação ao BUC.

3.5. Benchmarking dos Algoritmos Icebergue

O processamento de estruturas multidimensionais é um dos passos fundamentais dos sistemas OLAP. Muitas vezes, é irrealista calcular integralmente um cubo de dados, quando confrontado com elevada dimensionalidade e cardinalidade. A geração de cubos icebergue é um método efectivo para processar apenas os agregados cujos *group-by* se encontrem acima de um determinado patamar de suporte mínimo, de modo a determinar apenas os valores considerados vantajosos, ganhando em espaço e tempo de processamento. Como se viu na secção anterior, é notório que existem vários algoritmos para processar, de forma eficiente, estas estruturas, que recorrem a lógicas que exploram várias propriedades do cubo de dados e medidas (como a aplicação da função anti-monótona). Nesse contexto e, de forma a garantir o correcto entendimento do comportamento dos algoritmos icebergue mais predominantes, optou-se por replicar os gráficos de desempenho obtidos pelos autores de cada um dos algoritmos. Essencialmente, todos os comentários desenvolvidos nesta secção partiram da análise detalhada dessas imagens, que permitiram tirar conclusões mais concretas sobre o real processamento de *iceberg cubes*, entendimento esse fundamental para se investigar "a fundo" o *modus operandi* de cada uma das lógicas de selecção e em que situações devem ser (ou não) adaptadas. Note-se que este trabalho pretende criar, exclusivamente, um *survey* detalhado do mundo icebergue; os gráficos utilizados servem para complementar os comentários que foram efectuados sobre o desempenho de cada algoritmo, estando, por isso, devidamente assinalados com a indicação dos seus autores.

As experiências realizadas foram implementadas num PC compatível com Windows XP, com processador Pentium IV a 2.0 Ghz e 1GB de memória RAM. As lógicas distribuídas foram implementados num pequeno *cluster* de 32 bits. Os algoritmos foram implementados em C++ e os resultados marcados nos gráficos incluem o tempo de cálculo e de I/O. Os testes foram realizados apenas sobre conjuntos de dados limitados ao espaço de memória disponível. Veja-se agora a notação que irá ser utilizada para avaliar e comentar os testes comparativos realizados: *D* identifica o número de dimensões, *C* a cardinalidade de cada dimensão, *T* refere-se ao número de tuplos do cubo base, *M* ao nível mínimo de suporte e *S* identifica o

grau de enviesamento da distribuição. Quando *S* for 0 indica que a distribuição de dados é uniforme; à medida que vai aumentando, o enviesamento será maior. Os primeiros algoritmos a avaliar serão o *MultiWay*, o BUC e o H-CUBING (*Top-down* e *Bottom-up*). Veja-se os resultados obtidos em [Zhao et al., 1997], [Beyer & Ramakrishnan, 1999], [Han et al., 2001] referentes à dimensão dos tuplos e densidade dos dados.

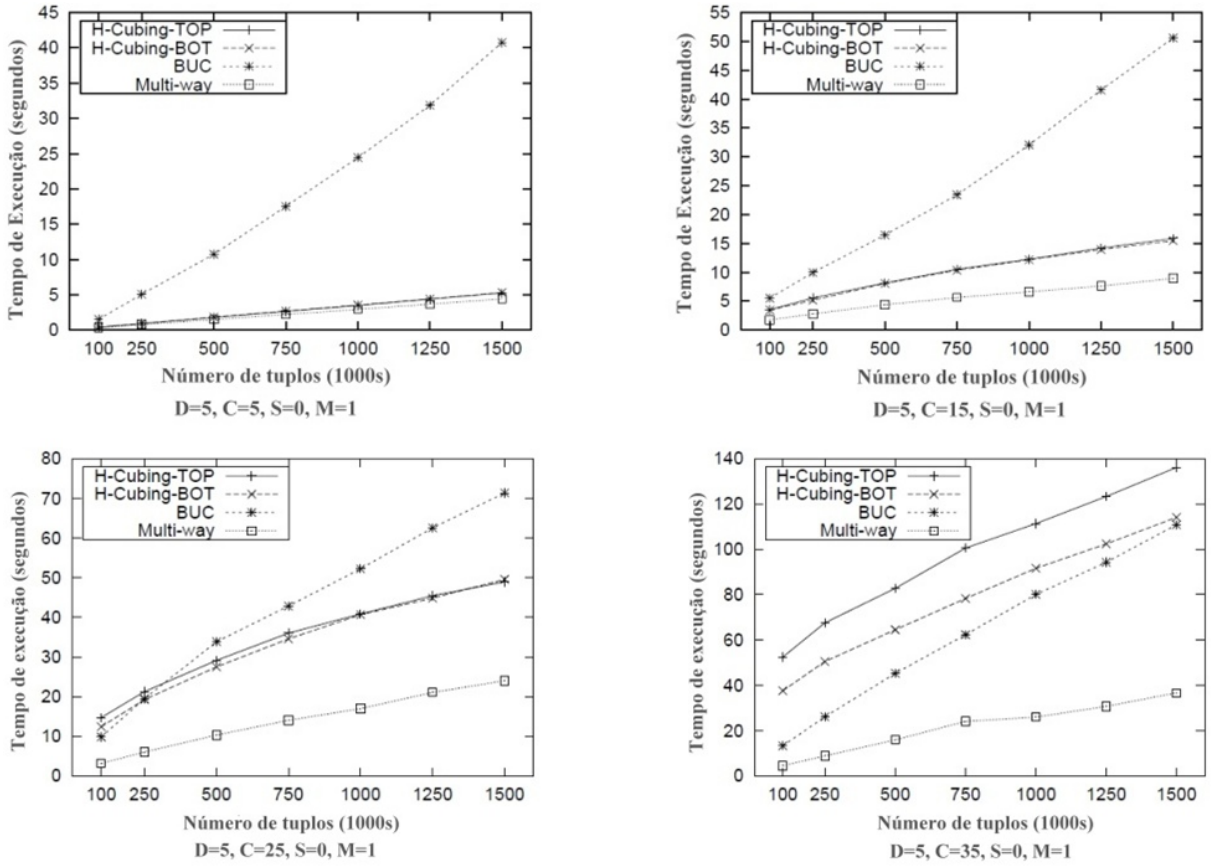


Figura 52. Avaliação desempenho: baixa dimensionalidade, alta e baixa cardinalidade

A considerar, logo em primeiro lugar, o número de tuplos *T* das relações base. Os gráficos da figura 52 mostram o desempenho dos quatro algoritmos face a um *D* baixo, numa distribuição uniforme. Façamos algumas observações: o BUC, sendo um algoritmo *O(n)*, responde linearmente ao aumento do *T*, o que faz sentido, já que o tempo de cálculo advêm da ordenação das contagens; o *MultiWay* responde de uma forma quase linear ao *T*, sobretudo porque este não afecta o tamanho dos *chunks* no *MultiWay*. Uma outra observação demonstra

que o H-CUBING tem um desempenho progressivamente inferior à medida que o *C* aumenta. Esta situação revela uma importante fraqueza destes algoritmos: quando o *C* é elevado e os dados são densos, o H-CUBING degrada-se imenso (devido à construção da *h-tree*). Note-se também que nestes exemplos, todas as agregações satisfazem o patamar mínimo de suporte *M=1*. O *MultiWay* parece ser a opção mais interessante em situações de baixa dimensionalidade, dados densos e distribuições uniformes, com um patamar de suporte baixo. De seguida, irá examinar-se o tempo de execução em situações de maior dimensionalidade. A evolução dos algoritmos é demonstrada nos gráfico da figura 53, com ligeiras alterações de parâmetros [Zhao et al., 1997], [Beyer & Ramakrishnan, 1999], [Han et al., 2001].

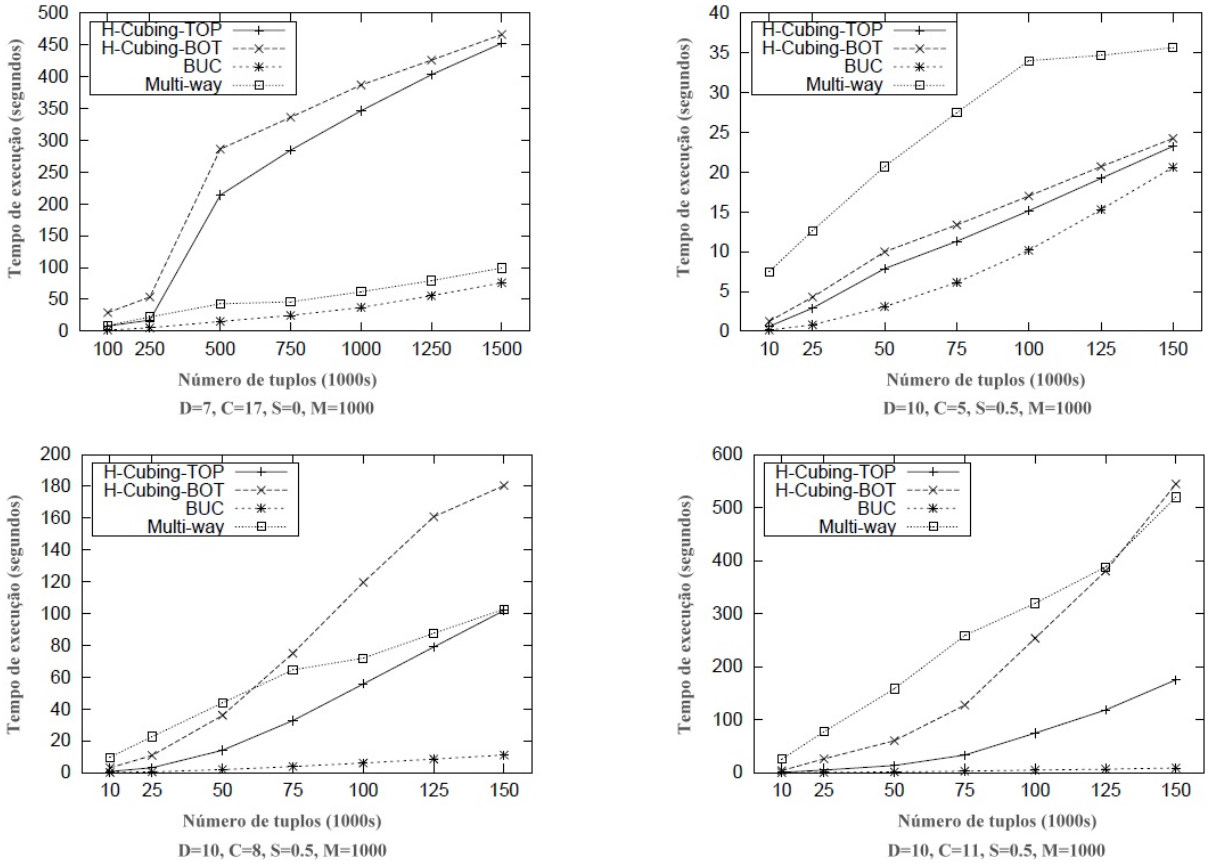


Figura 53. Avaliação desempenho: alta dimensionalidade, cardinalidade moderada

Como era previsível, o algoritmo *MultiWay* tem um desempenho exangue face aos restantes. A razão está relacionada com o facto d este ser exponencial ao *D* porque todos os *group-by* têm de ser processados. À medida que o *D* aumenta, os dados vão ficando cada vez mais

esparsos. Da mesma forma, os algoritmos H-CUBING têm um desempenho claramente inferior ao BUC sobretudo porque a *h-tree* é mais ampla e profunda; no BUC, o aumento é consequente do processo normal, linear de resposta. Repare-se também que o algoritmo HC (*Bottom-up*) torna-se pior em relação ao HC (*Top-down*) quando o *C* aumenta de tamanho.

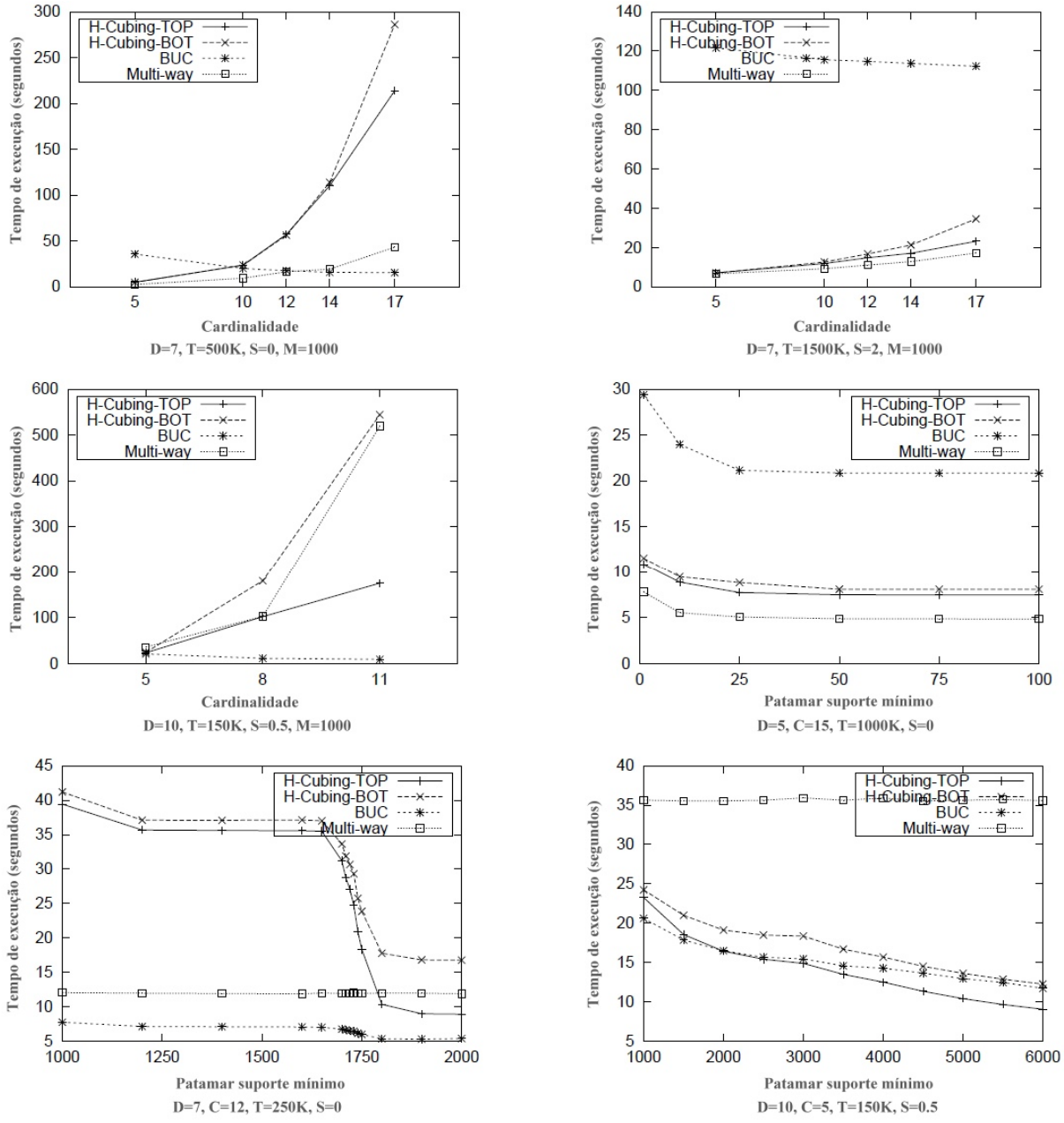


Figura 54. Avaliação desempenho: cardinalidade e patamar de suporte mínimo

Outra questão intimamente relacionada com a dimensionalidade, é a cardinalidade *C*. À medida que o *C* cresce, os quatro algoritmos aumentam em tempo de execução, o que é razoável, já que existem mais valores em cada dimensão para serem agregados: o algoritmo *MultiWay* tem mais *chunks*, o BUC tem que efectuar mais ordenações e a *h-tree* do H-CUBING tende a ser mais ampla. As duas respostas H-CUBING à cardinalidade são as mais interessantes: o aumento é exponencial. A razão está associada ao facto de cada nível da *h-tree* aumentar, pelo que a cardinalidade está a afectar a execução do algoritmo múltiplas vezes. A figura 54 demonstra, para além da evolução da cardinalidade em cenários de dimensionalidade moderada e elevada, a influência que o patamar de suporte mínimo tem nos algoritmos icebergue [Zhao et al., 1997], [Beyer & Ramakrishnan, 1999], [Han et al., 2001]. Sobre este último, à excepção do *MultiWay*, todos os algoritmos beneficiam de alguma forma de filtragem que explora a propriedade anti-monótona nas medidas contadas. Atente ao comportamento dos mesmos: o *MultiWay* não dão resposta ao aumento do *M*; por outro lado, o BUC e os algoritmos H-CUBING aumentam a sua velocidade de processamento quando o *M* é aumentado. No caso do primeiro, este fenómeno está relacionado com o facto das partições que "falham" o *M* não serem exploradas. No caso dos algoritmos H-CUBING, a evolução do *M* torna a *h-tree* mais ágil a nível de cálculo, evitando a invocação de muitas chamadas recursivas. Observe-se também que os algoritmos H-CUBING respondem melhor com um *M* elevado face ao BUC.

Um outro factor importante a ter em conta no desempenho dos algoritmos é o enviesamento dos dados em ambientes de baixa, média e alta dimensionalidade. Com um *D* baixo, o enviesamento dos dados beneficia o processamento de todos os algoritmos. Adicionalmente, para um *D* baixo e um *T* baixo, os dados são esparsos e razoavelmente compactos. Os algoritmos HC, na maioria das situações, conseguem ultrapassar o BUC, aproximando-se mesmo dos níveis do *MultiWay*, que domina em situações de baixa dimensionalidade. No gráfico à direita da primeira linha, o número de tuplos é muito alto, o que significa que os dados são muito densos. Neste contexto, somente o BUC apresenta melhorias. Nas situações de dimensionalidade moderada, como já verificado, os algoritmos HC tornam-se muito mais rápidos em situações de maior enviesamento. Esse fenómeno deriva do crescimento do *S*,

onde muitos nós da *h-tree* desapareceram ou encontram-se por baixo da condição icebergue. Em situações de elevada dimensionalidade o desempenho do BUC decresce à medida que os dados vão sendo mais enviesados (é uma reversão completa do algoritmo face a um *D* baixo). A figura 55 mostra os gráficos de desempenho dos algoritmos, com diferentes parâmetros de entrada [Zhao et al., 1997], [Beyer & Ramakrishnan, 1999], [Han et al., 2001].

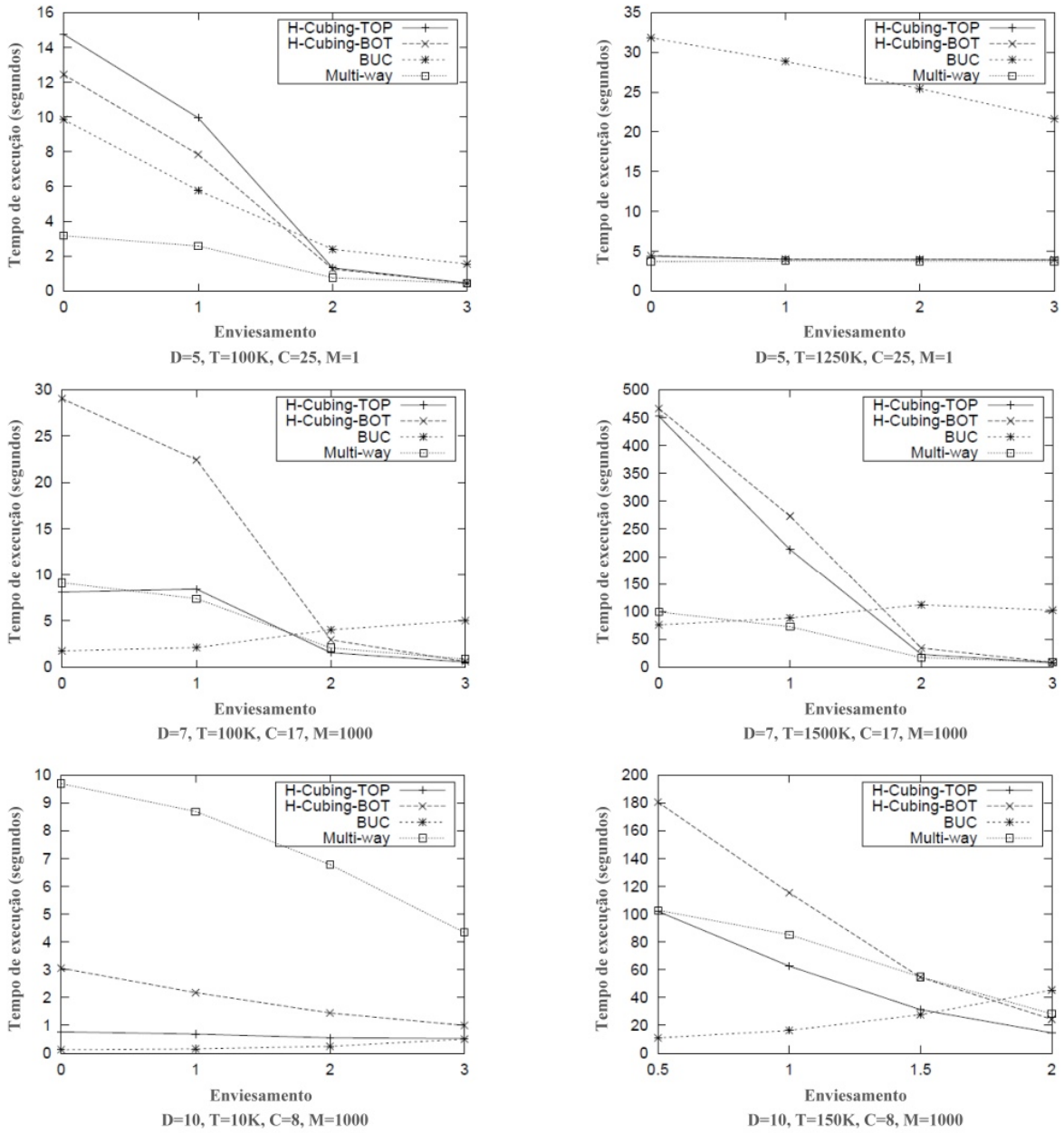


Figura 55. Avaliação desempenho: enviesamento das distribuições de dados

Genericamente, não existe um "vencedor" definido; todos eles apresentam as suas vantagens e desvantagens: o *MultiWay* apresenta melhorias assim que o *S* aumenta, particularmente em situações de baixa dimensionalidade. Por outro lado, parece justo afirmar que o BUC e o HC (*Top-down*) são os melhor algoritmos em ambientes com um *D* elevado. O BUC é superior quando a distribuição dos dados tende a ser mais uniforme; o HC (*Top-down*) quando os dados tendem a ser mais esparsos. Neste momento, será interessante introduzir um novo algoritmo icebergue para análise, no que diz respeito ao desempenho. Veja-se, de imediato, os gráficos de desempenho do STAR-CUBING (SC) para cada um dos parâmetros de entrada (indicados na figura 56) [Xin et al., 2003]. Note-se que, à excepção do *MultiWay*, todos os algoritmos testados até agora utilizam alguma forma de filtragem que explora a propriedade anti-monótona. No que diz respeito ao número de tuplos, verifica-se que o tempo de processamento aumenta para todos os algoritmos, à medida que a quantidade de dimensões e cardinalidade evolui [Xin et al., 2003]. Mesmo assim, o STAR-CUBING mostra-se mais eficiente face às restantes soluções. Recorde-se que, tanto o *MultiWay* como o H-CUBING não são apropriados para tratar situações de dimensionalidade e cardinalidades elevadas. Por essa razão, o desempenho do STAR-CUBING apenas será comparado com o BUC.

Sobre à cardinalidade, repare-se que o SC não é sensível a essa questão. Porém, em situações de elevada cardinalidade, o BUC aumenta o seu desempenho devido, sobretudo, ao crescimento da dispersão dos dados. O SC, apesar de iniciar a filtragem mais cedo, é condicionado pelo tamanho da *star-tree*. No que diz respeito ao patamar mínimo de suporte, é curioso que o SC no *min_sup=1000* consegue obter um desempenho 50% superior ao BUC. O tempo de I/O já não domina o cálculo neste ponto. Uma sugestão: mudar do SC para o BUC em situações em que o produto das cardinalidades seja razoavelmente maior, comparado com o tamanho dos tuplos. No caso do *S*, o *MultiWay*, o HC e o SC apresentam um desempenho superior em distribuições enviesadas. Todavia, em conjuntos de dados esparsos, o desempenho do BUC degrada-se à medida que o *S* aumenta, reflectindo o oposto do STAR-CUBING. No geral, o SC é mais rápido que o BUC no cálculo de *iceberg cubes* e o seu desempenho acentua-se quando o *min_sup* tende a ser menor. Para além disso, demonstra ser uma solução equilibrada em todos os tipos de distribuições de dados.

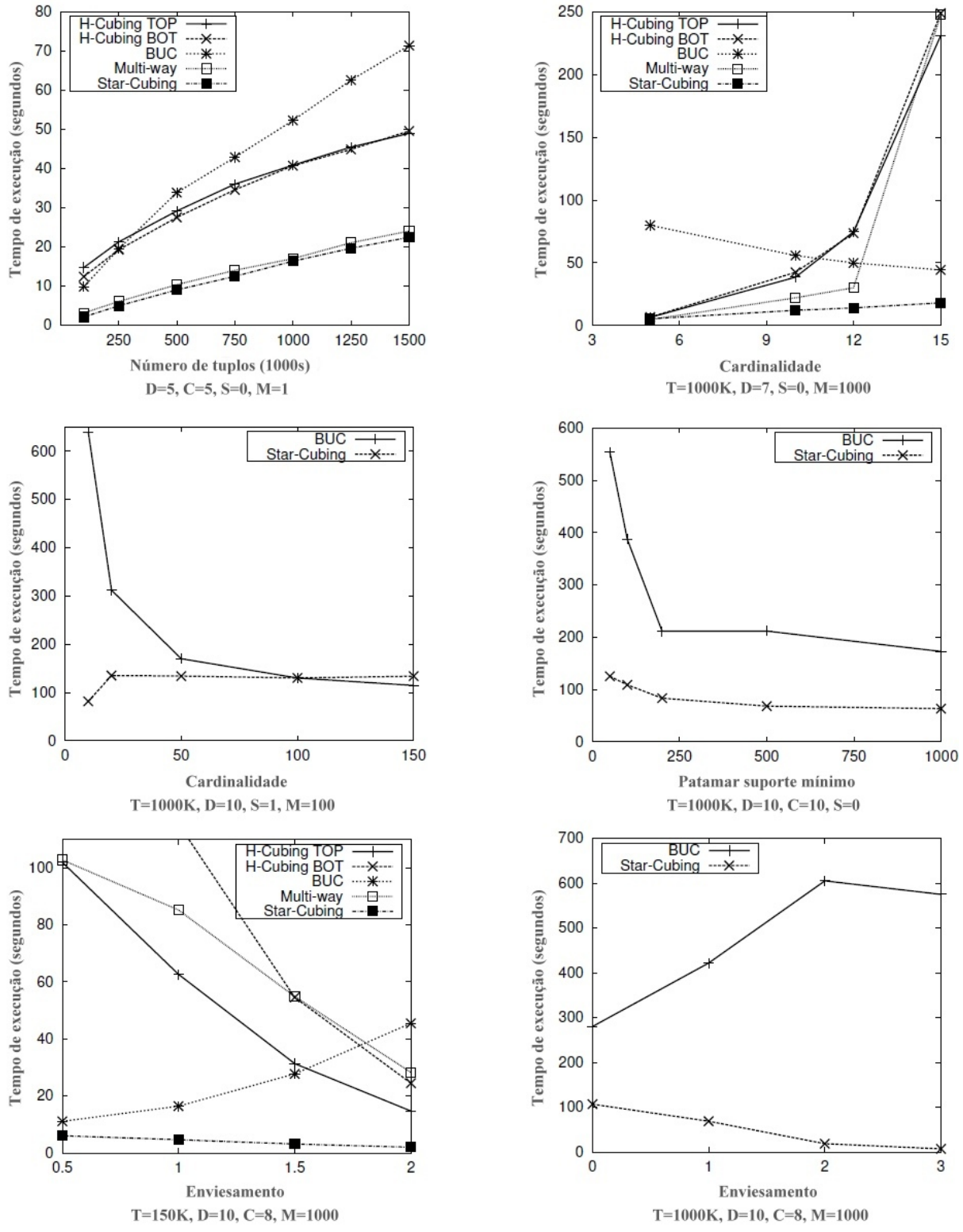


Figura 56. Avaliação de desempenho: STAR-CUBING em diferentes cenários

Introduz-se agora mais um algoritmo para análise de desempenho, o MM-CUBING (MM) [Shao et al., 2004]. O racional de comparação será semelhante ao efectuado com os algoritmos anteriores. O desempenho será medido considerando as duas soluções que, até ao momento, demonstraram maior robustez no processamento de *iceberg cubes*: o BUC e o STAR-CUBING. Veja-se a evolução dos gráficos indicados na figura 57 [Shao et al., 2004]. No que diz respeito ao número de tuplos, o comportamento evidenciado é semelhante ao demonstrado no SC, ou seja, o tempo de processamento aumenta com a evolução do número de dimensões e cardinalidade. No entanto, o MM mostra-se mais eficiente em relação às restantes soluções. No que diz respeito à dimensionalidade, o tempo de execução de todos os algoritmos cresce rapidamente: o SC é o pior classificado, em situações de elevada cardinalidade, enquanto que o MM tem um desempenho ligeiramente superior ao BUC. Sobre a cardinalidade, o BUC consegue obter melhores resultados em ambientes dispersos (quando a cardinalidade é elevada) e o SC, um comportamento superior em ambientes densos (de cardinalidade baixa). O MM exerce um bom desempenho em ambas as distribuições.

No que diz respeito ao *min_sup*, quando este é 50, o BUC tem um desempenho "pobre" porque invoca, principalmente, a filtragem APRIORI (um patamar pequeno tende a prevenir as acções de *pruning*). Assim que o *min_sup* começa a crescer, todos os algoritmos diminuem o seu tempo de execução e a diferença entre eles começa a ser nula. Sobre o enviesamento, é possível verificar que o BUC tem um desempenho inferior em relação aos restantes algoritmos. Note-se que, apesar dos dados não serem muito densos, um *min_sup* pequeno impede que a filtragem APRIORI seja efectuada logo cedo. Outra observação interessante é o facto do BUC ser o mais lento quando o *S=2*. Isto porque os dados pouco enviesados são praticamente uniformes e o BUC não pode realizar o *pruning* antes do particionamento, ao contrário dos dados enviesados, que contêm um conjunto alargado de células que podem ser filtradas imediatamente. O gráfico do canto inferior direito considera-se o mais expressivo face a um cenário real, com variação de cardinalidade entre os 25 e os 400. Quando o *S=0*, o SC tem um desempenho inferior devido à dispersão da distribuição de dados. Por outro lado, quando o *S* aumenta, o desempenho do BUC é degradado pela parte densa da distribuição.

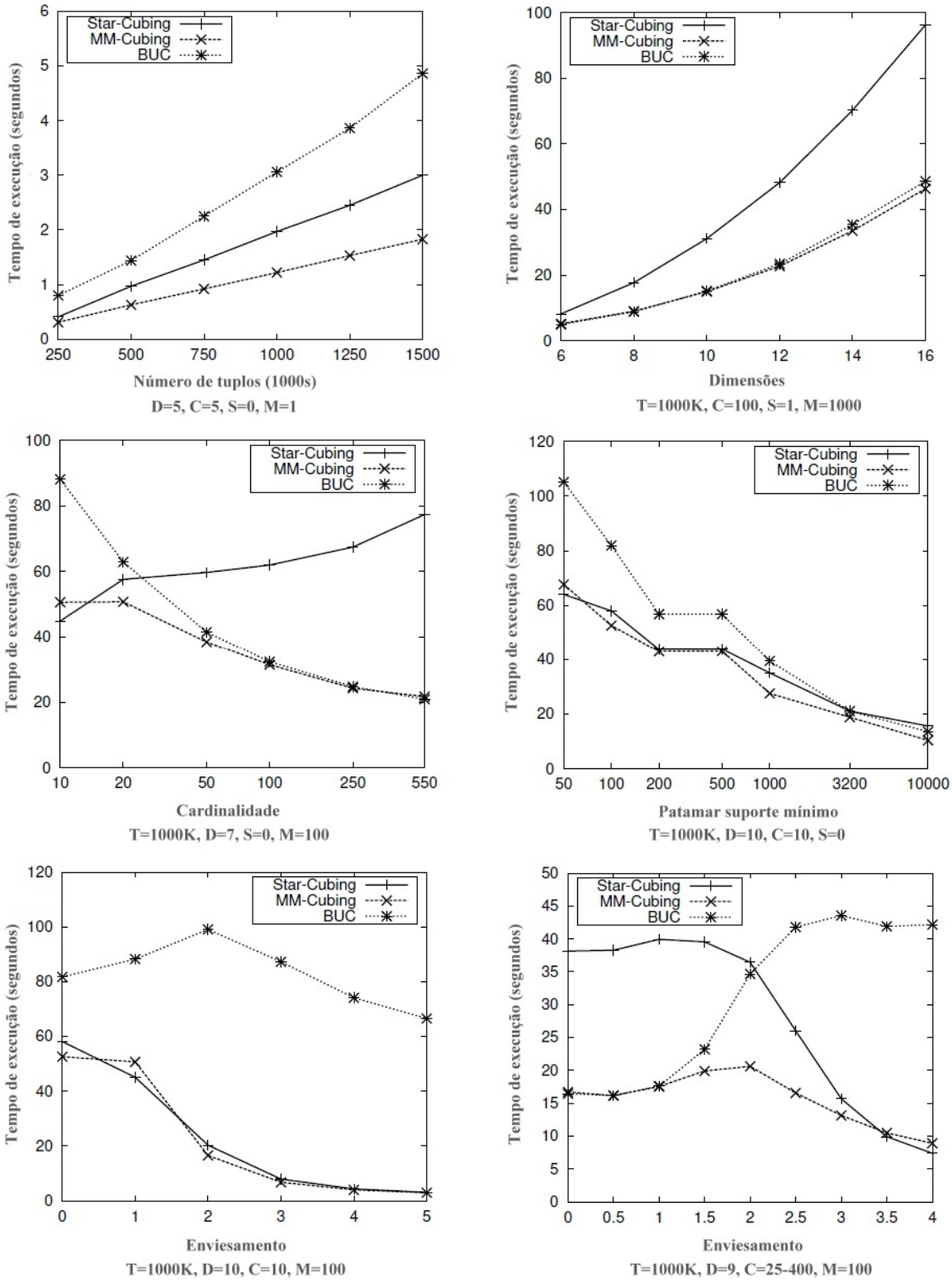


Figura 57. Avaliação de desempenho: MM-CUBING em diferentes cenários

Apesar de não existir um vencedor claro para todas as situações, o algoritmo MM-CUBING demonstra, até ao momento, ser a melhor proposta para ambientes heterogéneos, com uma solução perto da óptima. Analise-se agora os gráficos de desempenho relativos a um outro algoritmo icebergue, o PnP, visíveis na figura 58 [Chen et al., 2005]. Existem três grupos de experiências a incluir neste domínio: o primeiro está relacionado com o processamento de *iceberg cubes* em memória interna; posteriormente, serão avaliados os efeitos do processamento PnP em memória externa; por último, serão comentados os resultados do algoritmo em ambientes paralelos e distribuídos. Veja-se, de imediato, o primeiro grupo. No que diz respeito ao número de tuplos, todos os algoritmos têm um crescimento acentuado devido à cardinalidade; na presença de uma distribuição dispersa, o BUC e o PnP têm um desempenho superior em relação o SC. Apesar do BUC iniciar com um desempenho superior (entre os 2000s e os 3000s) troca de posição com o PnP, à medida que a dispersão dos dados vai aumentando. Sobre a cardinalidade, é igualmente possível visualizar o grau de dispersão dos dados. A seguinte fórmula retorna o nível de dispersão de um cubo: $Ds = T / \prod_{i=1}^{C} |Ci|$.

Note-se que, com um *M=100*, o PnP e o BUC têm um comportamento muito semelhante, quando o *Ds* encontra-se entre os 0,02 e 0,00002. Quando o *Ds* é superior a 0,02, o SC tende a melhorar o seu desempenho. Sobre os restantes parâmetros, o PnP é comparável com o SC e o BUC no que diz respeito ao nível de variações de dimensionalidade, *min_sup* e enviesamento. Seja em cubos densos (*C=10* ou *Ds=5*) ou esparsos (*C=100* ou *Ds=0,000005*) é possível verificar que o desempenho sequencial do PnP é muito estável. No geral, o PnP atinge os melhores níveis de eficiência quando se encontra no intervalo de dispersão $0.00002 \leq Ds \leq 0.02$, e mesmo assim, tem um desempenho semelhante ao BUC e ao SC em situações extremas de dispersão e densidade. É uma alternativa interessante especialmente em situações onde a estabilidade do desempenho sobre um grande número de parâmetros de *input* seja essencial. No que diz respeito ao segundo grupo de experiências (processamento do algoritmo PnP em memória externa) observe os resultados obtidos nos gráficos de desempenho (figura 59) [Chen et al., 2005]. Os testes foram realizados sobre um conjunto de dados, que oscilou entre 1 a 20 milhões de tuplos, com uma dimensionalidade *D* e memória disponível variáveis.

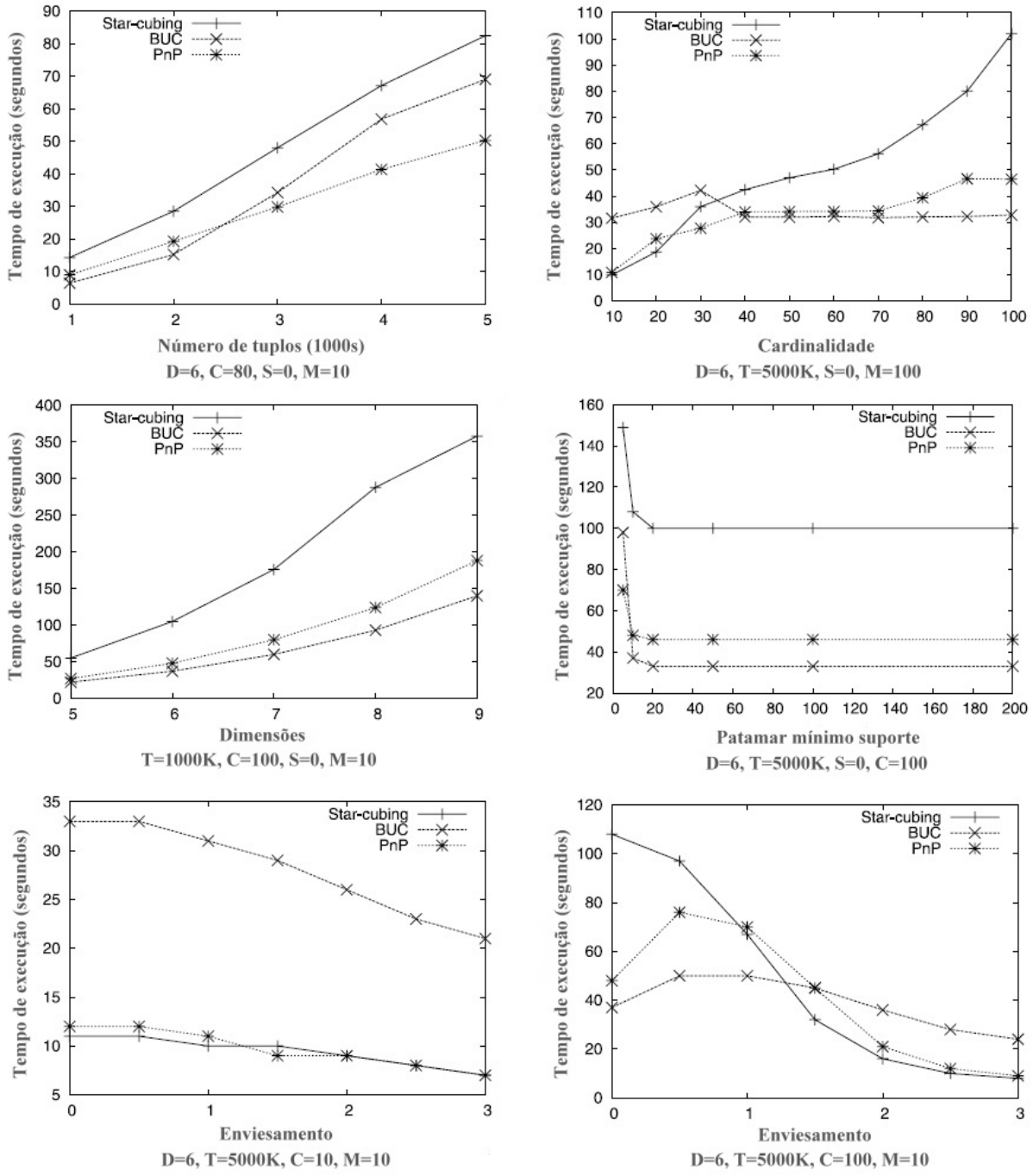


Figura 58. Avaliação desempenho: PnP sequencial, em memória interna

Recorde-se que esta variante do algoritmo é composta por uma análise linear simples da memória, pelo que não necessita de estruturas complexas de armazenamento. Por isso, o PnP torna-se razoavelmente fácil de implementar em sistemas de memória externa, utilizados

sobretudo para o cálculo de *querys* icebergue mais complexas. Para além disso, este algoritmo dispõe de um gestor de I/O proprietário de modo a ter-se controlo total das operações em disco, bem como da latência "escondida" na sobreposição das transacções.

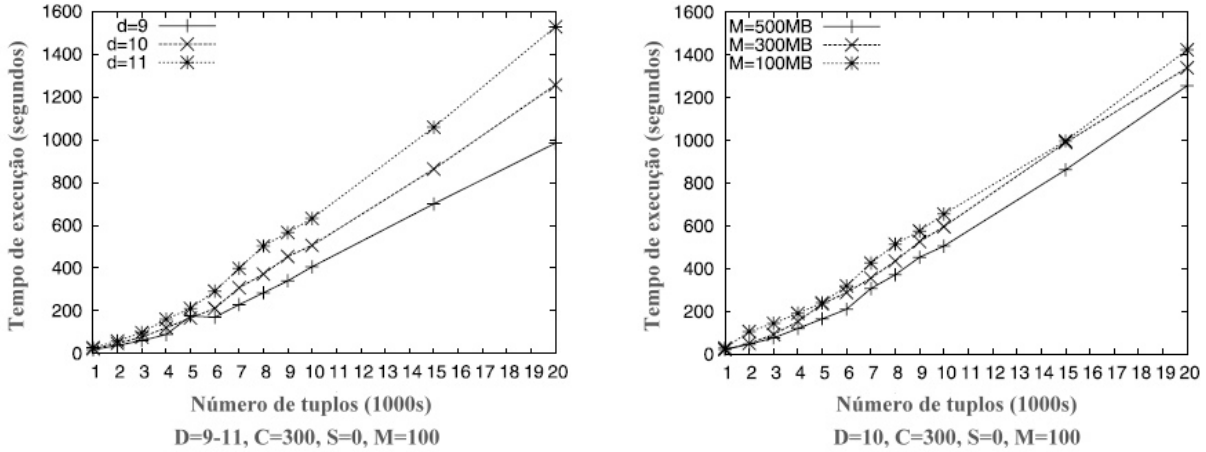


Figura 59. Avaliação desempenho: PnP em memória externa

Genericamente, a avaliação deste grupo demonstra perda mínima de eficiência quando o PnP muda o processamento da memória interna para a externa. O tempo de execução da memória externa (quando o PnP é forçado a utilizá-la por limitações da memória principal) é apenas ligeiramente superior ao tempo de execução da memória principal interna para a mesma *iceberg query*. No gráfico à esquerda, da figura 59 [Chen et al., 2005], verificam-se curvas semelhantes, mesmo com o aumento do *D*, devido sobretudo aos efeitos da filtragem APRIORI. Assiste-se a uma ligeira alteração no tempo, correspondente à mudança para memória externa, entre os 5 e os 7 milhões de linhas, que dependem da dimensionalidade do *iceberg cube* em geração. No gráfico da direita [Chen et al., 2005] são visíveis os benefícios do aumento da memória disponível para efectuar o processamento do algoritmo. Contudo, as disparidades entre as curvas são mínimas. Isso sugere que o PnP faz uso total da memória externa disponível para além de demonstrar um bom desempenho, mesmo em cenários com condições limitadas a nível de espaço. Sobre o último grupo de experiências (referente à implementação do PnP em sistemas paralelos e distribuídos), repare no comportamento do PnP quando a distribuição da carga é partilhada por diferentes processadores. Esta avaliação é baseada no código fonte de memória externa do PnP com a adição do paralelismo, em

ambientes distribuídos. Os testes focam, sobretudo, o desempenho progressivo do PnP já que esta é uma das métricas chave para avaliar os sistemas distribuídos de base de dados [DeWitt & Gray, 1992]. Os gráficos de desempenho são relatados na figura 60 [Chen et al., 2005].

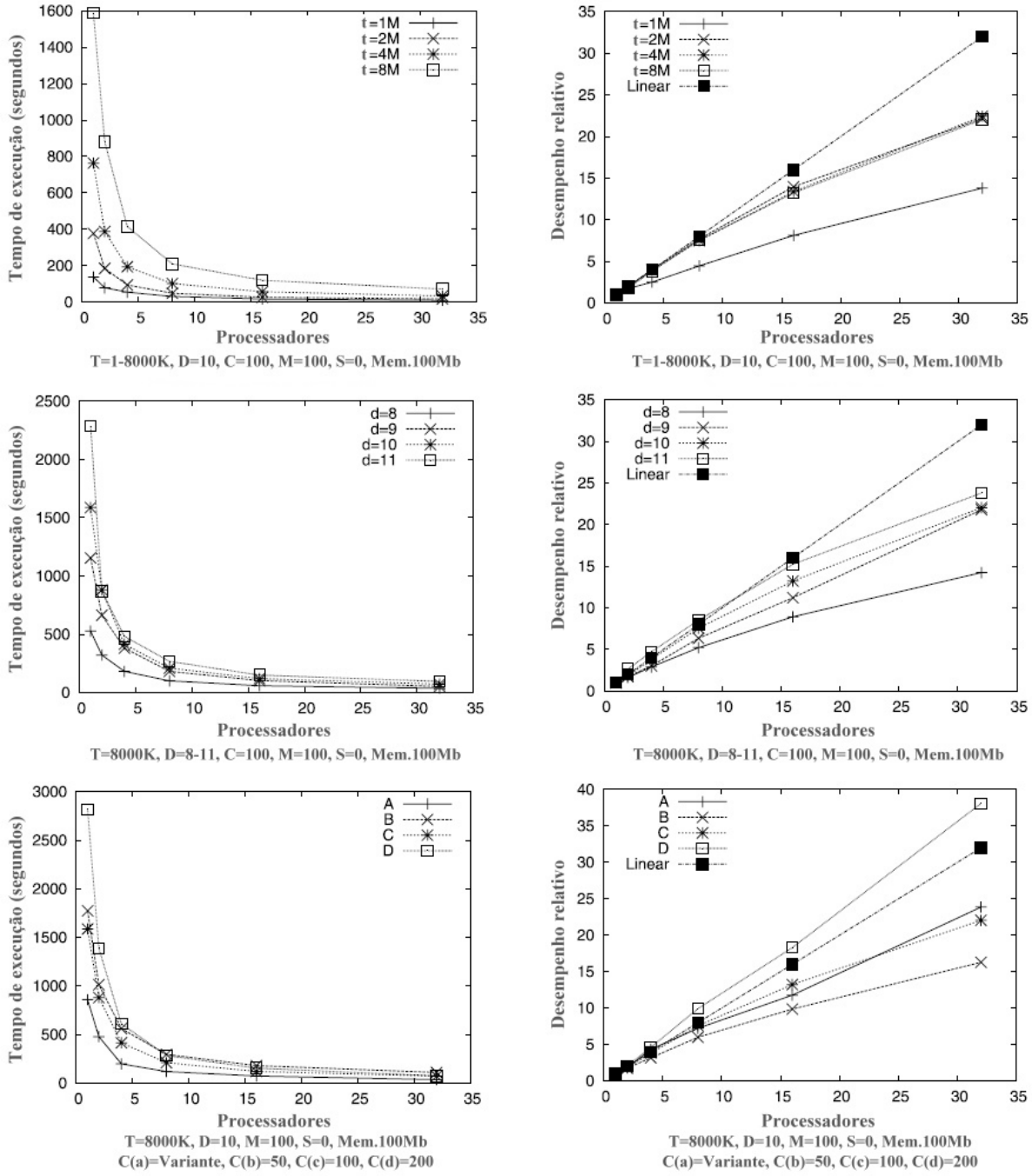


Figura 60. Avaliação desempenho: PnP paralelo, distribuído

Os testes efectuados consistiram em incrementar o número de processadores disponíveis para o cálculo paralelo do PnP de modo a determinar o tempo (e desempenho correspondente) obtido na variação de parâmetros chave, tais como o tamanho da distribuição de dados, a dimensionalidade e a cardinalidade. A primeira linha de gráficos da figura 60 [Chen et al., 2005] relaciona o número de tuplos de entrada, num intervalo de 1 a 8 milhões, com o aumento de desempenho correspondente, à medida que são adicionados mais processadores. Note-se que no intervalo de 2 a 8 milhões de tuplos, o ponto de aceleração ideal corresponde a valores até oito processadores. A partir dai, assiste-se a um decréscimo lento do desempenho decorrente do cálculo reduzido, que dificulta o domínio da sobrecarga da comunicação em cada máquina. No que diz respeito à dimensionalidade, repare que a aceleração do desempenho cresce à medida que as dimensões aumentam: quando isto acontece, existem exponencialmente mais vistas sobre o cubo, o que implica um aumento acentuado no tempo de cálculo local. O tempo de comunicação também aumenta, mas mais devagar. Na prática, pelo dobro do número de vistas, o PnP requer apenas uma ordenação paralela adicional. A situação da cardinalidade das dimensões face ao *input* pode afectar o desempenho em duas vertentes: ou o produto das cardinalidades afecta a dispersão dos dados ou o rácio do tamanho das maiores cardinalidades em *p* afecta a qualidade do balanceamento paralelo da carga.

Na última linha de gráficos da figura 60 [Chen et al., 2005] o conjunto de dados *d* é denso e carece, ao mesmo tempo, de uma qualquer dimensão de elevada cardinalidade. Os conjuntos de dados *a* e *c* são esparsos e têm grandes cardinalidades máximas em simultâneo, o que se traduz num balanceamento da carga superior. No caso do conjunto *a* assiste-se pela primeira vez a uma aceleração linear perfeita do desempenho, motivada pelo crescimento de *p*. À medida que isso acontece, os dados alocados em cada processador diminuem, permitindo uma adaptação dinâmica do algoritmo PnP. No que diz respeito ao patamar de suporte mínimo, analise-se os gráficos da figura 61 [Chen et al., 2005]. Para *min_sup* mais baixos (*M=100-500*) o tempo de processamento é maior e a aceleração do desempenho do PnP é quase linear até 16 processadores. Para além disso, assiste-se a um decréscimo acentuado em direcção aos 32 processadores. As curvas de *M=100* e *M=500* encontram-se praticamente sobrepostas porque a maioria dos valores do cubo icebergue é maior que 500 (o resultado traduz-se quase

no mesmo volume de dados para ambos os *M*). Em situações de limites de suporte mínimo superiores (*M=1000-2500*), a aceleração do desempenho decresce rapidamente porque, a esse nível, a maior parte dos dados são filtrados e existe menos processamento local para "disfarçar" os custos de comunicação.

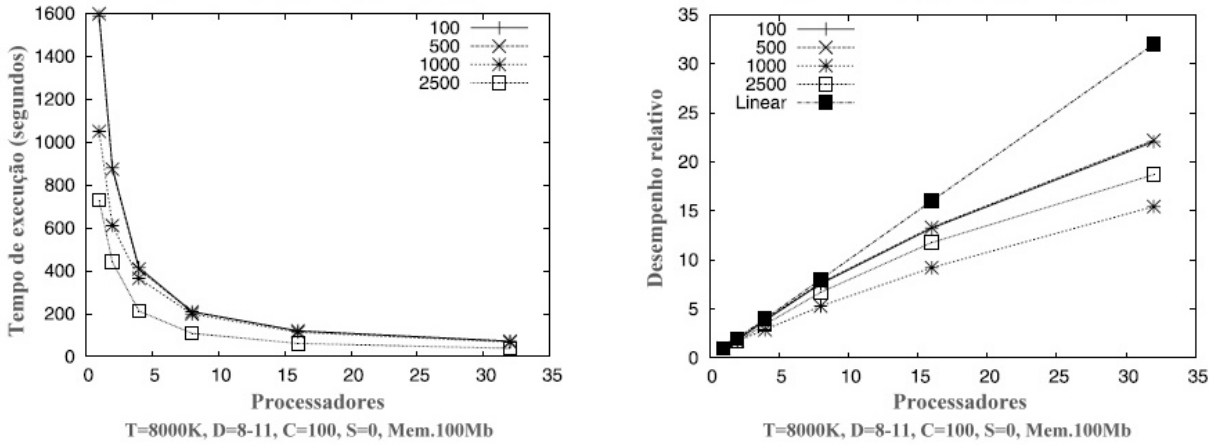


Figura 61. Avaliação desempenho: PnP paralelo, distribuído (2)

No geral, para o cálculo paralelo de *iceberg cubes* em ambientes distribuídos, o algoritmo PnP apresenta uma escalabilidade interessante e providencia um aumento de desempenho linear à medida que o número de processadores vai aumentando. Para além disso, o PnP consegue obter um desempenho benéfico, tanto em ambientes esparsos como densos. Em conjuntos de dados reais, o PnP demonstra ser um algoritmo muito eficiente, para além de ser o mais indicado para processar distribuições de dados consideradas complexas para os métodos sequenciais, sobretudo por causa da sua extensão, dimensionalidade e cardinalidade. Até ao momento, foi efectuada uma análise detalhada dos algoritmos icebergue mais predominantes, sobre diferentes distribuições de dados e aplicando lógicas de selecção centralizadas e distribuídas. Para finalizar a avaliação de desempenho das lógicas de selecção icebergue, importa analisar mais um algoritmo: o CT-CUBING (CT) [Cho et al., 2005], que se distingue pela sua inovação e robustez, sobretudo, ao lidar com grandes repositórios de dados. Este será comparado com os resultados obtidos pelo BUC. Os testes efectuados baseiam-se em conjuntos de dados que seguem uma distribuição enviesada. Recorde-se que a lógica deste algoritmo passa por calcular o cubo icebergue directamente através de diferentes tabelas do

Data Warehouse. Os restantes parâmetros de entrada são definidos da seguinte forma: assume-se uma tabela de factos de 5 dimensões, um *T=1000K* e um *C=10*. Para além disso, supõem-se três tabelas de dimensão, cada uma com três atributos. O factor de enviesamento *S* é igual a 1. Ao pensar-se num pequeno *Data Warehouse* gerado através dos dados acima mencionados, se existem *n* dimensões na tabela de factos e *k* tabelas de dimensão ($n \geq k$) e se cada dimensão tem *l* atributos, então a tabela base única terá $(l \times k + (n - k))$ dimensões. Assim, por defeito, o repositório terá onze dimensões.

A função de agregação utilizada na condição icebergue é o *COUNT(*)*. Portanto, o domínio, a cardinalidade e a distribuição dos dados não têm efeito sobre os resultados obtidos. O *M* está definido de acordo com a seguinte lógica: [*COUNT(*) ≥ T da tabela de factos × 5%*]. Em todos os testes efectuados, o tempo de execução do CT corresponde ao tempo que o algoritmo demora a processar o *iceberg cube* de diferentes tabelas, incluindo o tempo de CPU e I/O. No caso do BUC, o tempo de execução refere-se apenas ao tempo que o algoritmo demora a calcular um *iceberg cube* da tabela base única, incluindo o tempo de CPU e I/O. O tempo de obtenção da tabela base universal não é contabilizado. De modo a simplificar a comparação dos algoritmos, assume-se que a tabela universal base pode ser guardada, na sua totalidade, na memória principal (quanto tal situação não se verifica, o desempenho do BUC decresce substancialmente). O CT não necessita de armazenar todas as tabelas em memória. Em vez disso, carrega-as uma a uma. As células icebergue locais podem ser indexadas e armazenadas em disco. O maior consumo de memória do CT está relacionado com a necessidade de armazenar a *h-tree* de toda a tabela de factos. Felizmente, a *h-tree* tende a ser sempre mais pequena que a tabela de factos e muito mais pequena em relação à tabela universal base. Contudo, se a árvore ultrapassar a memória disponível, podem ser aplicadas as técnicas mencionadas em [Xin et al., 2003]. Os testes finais efectuados pelos autores do algoritmo são demonstrados pelos gráficos disponíveis nas figuras 62 e 63, cujas curvas são auto-explicativas [Cho et al., 2005]. Genericamente, o CT-CUBING evidencia um desempenho superior, sendo sempre o mais eficiente e escalável em comparação com o BUC. O desempenho do BUC demonstrou-se consistente com os testes previamente efectuados ao longo desta secção.

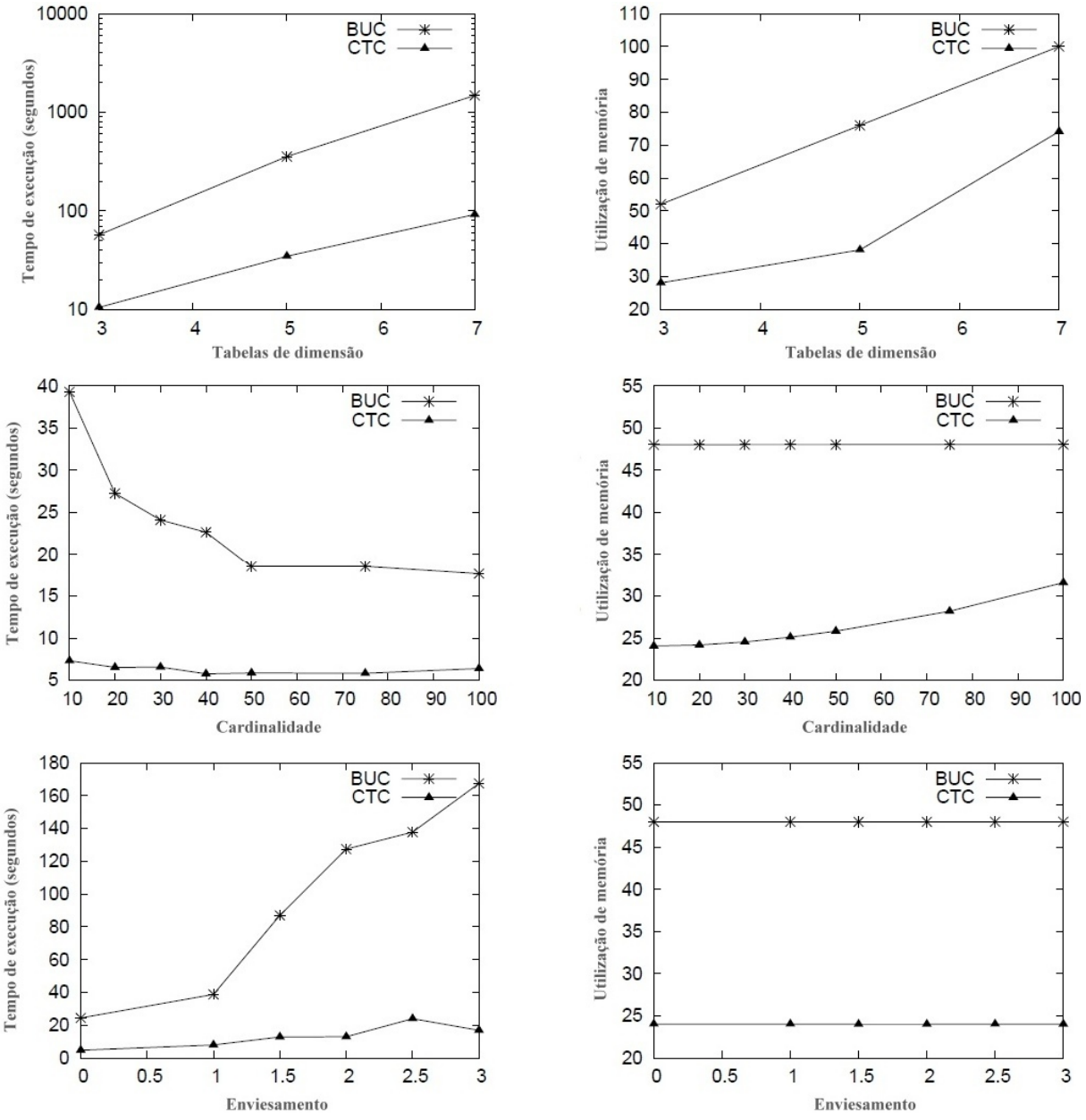


Figura 62. Avaliação desempenho: várias configurações CT

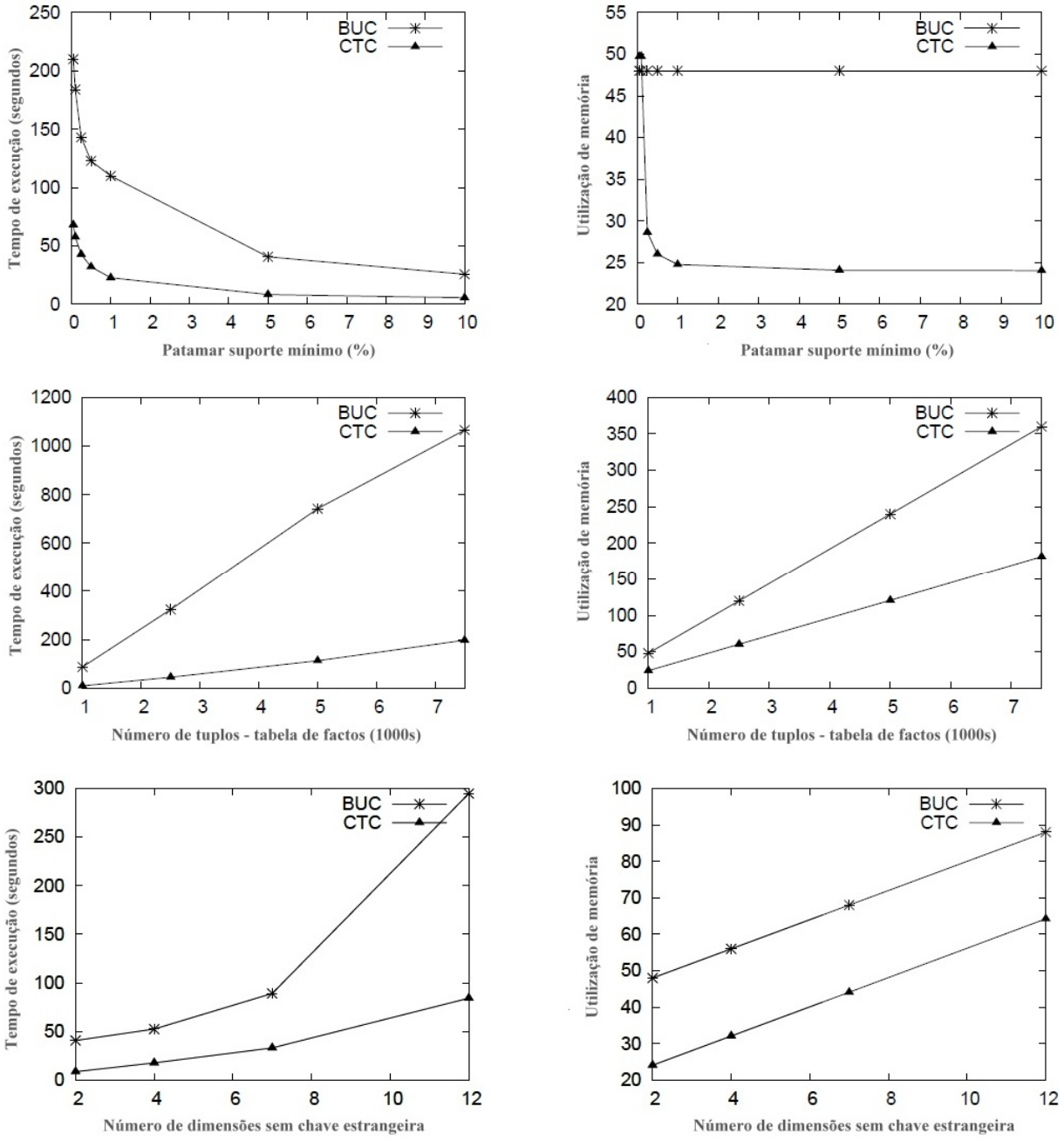


Figura 63. Avaliação desempenho: várias configurações CT (2)

Capítulo 4

Conclusões e Trabalho Futuro

4.1. Comentários Finais e Trabalho Futuro

Uma das tecnologias mais importantes e proeminentes dos sistemas informáticos que, apoiam o processo de tomada de decisões nas organizações, são os sistemas de processamento analítico (OLAP). Na prática, através da exploração de vistas multidimensionais criadas sobre os dados armazenados no *Data Warehouse*, os sistemas OLAP permitem aos utilizadores efectuarem operações de *drill-down* ou *roll-up* sobre as hierarquias, *slice* e *dice* dos atributos específicos ou mesmo executar diversas operações estatísticas. De modo a suportar essas funcionalidades, os sistemas de processamento analítico dependem piamente do suporte de um modelo conceptual, representado na forma de um cubo de dados. Genericamente, um cubo permite que os utilizadores possam ter uma visão multi-perspectiva sobre os dados organizacionais e, em diversos níveis de sumarização. É constituído por um subcubo base, que representa a visão de menor granularidade: contém o complemento integral das d dimensões (ou atributos) rodeadas pelos conjuntos $(2^d - 1)$ de subcubos, que representam o nível de agregação do subcubo base, simultaneamente numa ou mais dimensões. O operador CUBE (uma extensão sintáctica do SQL) [Gray et al., 1996] foi proposto como um meio para simplificar o processo de construção de um cubo de dados. Após a publicação dos primeiros trabalhos realizados neste domínio, um número considerável de projectos de investigação começou a emergir na literatura, focando-se sobretudo, no desenvolvimento de algoritmos

eficientes para o processamento de estruturas multidimensionais, de modo a dar resposta ao rápido crescimento dos *Data Warehouses* e, consequentemente, ao volume acentuado de dados analíticos. Nessa perspectiva, este trabalho focou o estudo de métodos que permitem processar, de forma optimizada, as estruturas multidimensionais de dados, reflectindo sobretudo, sobre os problemas decorrentes da sua selecção, materialização e manutenção (condições fundamentais de desempenho em qualquer sistema OLAP). Na prática, grande parte das características destes sistemas procede da forma como os utilizadores vêem o negócio e como estes estabelecem as suas necessidades. Tais exigências impostas aos sistemas de processamento analítico acabaram por determinar uma evolução iminente e imutável dos repositórios de dados e arquitecturas subsequentes. A própria complexidade das interrogações e o perfil de utilização motivaram novas preocupações e incertezas relacionadas com a capacidade e tempo de resposta das consultas analíticas impostas pelos agentes de decisão. A concretização dessas respostas agregadas às próprias consultas, em tempo útil, estabeleceram a necessidade de criação das vistas materializadas.

Contudo, ao longo deste trabalho diversas vezes foi referido que a complexidade dimensional do negócio e as exigências dos utilizadores em obter resultados a curto prazo determinam a impossibilidade de materializar uma estrutura dimensional de dados, na sua totalidade. Por isso, existe precisão em recorrer a uma selecção eficaz das agregações consideradas mais benéficas para responder às necessidades de negócio. Este problema, denominado pela escolha de vistas a materializar, constitui um dos focos principais de investigação no domínio dos sistemas de processamento analítico, que motivou largamente a compilação deste trabalho. Uma selecção adequada pode traduzir-se, de facto, em ganhos muito significativos, conseguindo-se, com uma percentagem baixa do espaço de materialização total do subcubo, valores quase idênticos para o tempo de resposta às interrogações colocadas, tomando, como valor comparativo base, a materialização completa do cubo. Este ganho potencial mostrou a relevância do problema, que mereceu, desde muito cedo, a atenção da comunidade científica e para o qual foi sendo proposto um variado número de soluções. O surgimento de novos desafios e soluções arquitecturais incentivou a investigação neste domínio, já que importa adaptar e estender as soluções tradicionais, às novas realidades. Nesse contexto, aparece a

possibilidade de tirar partido do perfil de utilizador, permitindo a aludida possibilidade de selecção das estruturas mais apropriadas. No entanto, alterando-se as necessidades dos utilizadores, será necessária a readaptação das estruturas multidimensionais de dados. Irão existir subcubos materializados que deixarão de ser úteis, e outros, não materializados, que seriam, porventura, benéficos. Esta reestruturação deverá, desejavelmente, acompanhar de muito perto as mutações no perfil de utilização. Então, se a materialização das agregações é significado de desempenho, uma selecção optimizada dos subcubos traduzir-se-á, certamente, numa melhoria da satisfação dos utilizadores e da qualidade das consultas analíticas que suportam o processo de tomada de decisões. Nessa perspectiva, importa referir um outro vector de investigação deste trabalho relacionado com o desenvolvimento do conceito de *iceberg cubes*. De facto, nos dias de hoje, a carência de informação útil e disponibilizada atempadamente motivou o aparecimento de técnicas de optimização para aumentar o desempenho dos sistemas OLAP. A materialização de vistas tem-se mostrado um bom caminho a seguir, mas deve ser considerada apenas como uma parte do processo de optimização (a escalabilidade dos sistemas analíticos e o perfil de utilização são, igualmente, factores chave que devem ser considerados no ciclo de optimização OLAP).

Porém, a materialização tradicional e integral do cubo, cujos constrangimentos tempo e espaço são notórios, é hoje considerada uma realidade quase inatingível, dada a evolução exponencial dos sistemas de *Data Warehousing* e de processamento analítico. A materialização parcial representa, por outro lado, um interessante *trade-off* entre o espaço de armazenamento e o tempo de pré-processamento de vistas. Aqui são analisadas algumas técnicas de optimização denominadas "icebergue" como resposta à reformulação do problema de materialização integral de vistas. Basicamente, as interrogações efectuadas num *iceberg cube* partem da análise de um atributo (ou conjunto de atributos) de forma a obter-se as agregações que estão acima de um determinado limiar ou patamar de suporte, face a determinadas condições (expressas em SQL por *having count,* ou *sum*, por exemplo). Estas interrogações são designadas, como se viu, por *iceberg querys*, uma vez que conseguem lidar apenas com os valores que estão acima de um determinado patamar (a "ponta do icebergue") face ao volume total do cubo de dados (o "icebergue"), de modo a utilizar menos espaço de

armazenamento e, significativamente, menos tempo de processamento. Análogo ao problema de selecção das vistas consideradas mais adequadas para serem materializadas, a selecção dos cubóides que, de facto, satisfazem a condição icebergue estabelecida são também matéria de investigação na literatura, estando na base do denominado problema de selecção de *iceberg cubes*. Nesse contexto, o problema é expresso pela determinação de qual o conjunto de subcubos que satisfazem uma determinada condição agregada de modo a serem integrados numa vista a materializar. Repare que esta acção irá permitir filtrar muitas agregações elementares, valores incorrectos ou nulos que não trazem qualquer beneficio em termos analíticos. Assim, apenas serão materializados os nós representativos (materialização parcial), o que significa um ganho considerável em termos de tempo de cálculo e espaço em memória. A distinção das técnicas de *iceberg cubing* são assinaladas pela avaliação critica dos seus algoritmo. Por isso, importa evidenciar conclusivamente as características de cada um. Note que todos eles permitem gerar *iceberg cubes* de forma eficiente mas apresentam lógicas distintas de funcionamento. É difícil nomear um "vencedor", já que a escolha acertada depende de diferentes factores e condições.

No caso do 1) *MultiWay* [Zhao et al., 1997] verifica-se um comportamento superior quando é aplicado em situações de baixa dimensionalidade. Contudo, se a dimensionalidade tender a ser muito enviesada, o algoritmo 3) H-CUBING [Han et al., 2001] evidencia-se superior. Por outro lado, o algoritmo 2) BUC [Beyer & Ramakrishnan, 1999] é o mais eficaz em ambientes de elevada dimensionalidade, no tratamento de dados esparsos e uniformes. Todavia, demonstra um fraco desempenho no tratamento de dados muito condensados e enviesados. Note-se que os testes efectuados sobre estes algoritmos (secção 3.5) ainda não são suficientes para uma análise profunda em ambientes de elevada dimensionalidade e cardinalidade e necessitam de um planeamento mais cuidado, de modo a serem avaliados de forma mais precisa. Para além disso, algumas alterações nos algoritmos podem constituir uma melhoria significativa: por exemplo, no caso do BUC, seria sensato implementar o método de ordenação *QuickSort* em adição ao *CountingSort*, o que permitiria alternar para o primeiro, quando os dados fossem suficientemente densos, obtendo uma eficácia ainda maior. Posteriormente, é sugerido na literatura um outro algoritmo, o 4) STAR-CUBING [Xin et al.,

2003] que integra a força de ambas as abordagens *Bottom-up* e *Top-down* já analisadas. Repare-se que esta lógica de selecção distingue-se pela sua capacidade em efectuar agregações partilhadas, tirando partido das dimensões distribuídas entre os subcubos principais e seus descendentes. Adicionalmente, permite aplicar a filtragem APRIORI, logo que possível, de modo a rejeitar as células menos promissoras do cubo icebergue. Os testes efectuados a este algoritmo confirmam que o STAR-CUBING é um método deveras auspicioso. No processamento de um cubo de dados integral, se o conjunto de dados for denso, o seu desempenho é comparável com o *MultiWay* e é bastante mais rápido em relação ao BUC ou ao H-CUBING. Por outro lado, se o conjunto de dados for esparso, o STAR-CUBING é significativamente mais rápido em relação ao *MultiWay* e ao H-CUBING e, ligeiramente mais rápido do que o BUC, na maioria das situações. No processamento de *iceberg cubes*, o algoritmo STAR-CUBING consegue conquistar um desempenho superior em todas as distribuições de dados. Algum trabalho futuro a ser realizado neste âmbito será, seguramente, alargar o domínio do algoritmo a outros métodos optimizados de cálculo de estruturas multidimensionais: por exemplo, em *condensed cubes* e *approximate cubes* [Barbara et al., 1997], [Imielinski et al., 2002]. Ulteriormente, uma outra abordagem de cálculo de cubos icebergue é proposta na literatura, que actua através da factorização do *lattice* espacial de soluções, denominada de 5) MM-CUBING [Shao et al., 2004].

Este algoritmo, à semelhança da lógica de selecção anterior, aufere um comportamento regular na maioria das distribuições de dados. Vale a pena referir duas técnicas de optimização que o algoritmo associa: a reordenação das dimensões, que permite acelerar o processo de cálculo do cubo, através da geração de um sub-espaço com as dimensões mais reduzidas; e a aplicação de uma estratégia coerente de selecção das agregações maiores, que afecta directamente o desempenho. Repare-se que a estratégia *greedy* implementada tem um desempenho razoável, mas é possível ir mais além, introduzindo uma estratégia congruente de selecção dos valores agregados superiores das dimensões. Os testes de desempenho comprovam que, para conjuntos de dados uniformes, o algoritmo MM-CUBING tem um comportamento semelhante em relação ao STAR-CUBING e ao BUC. Em distribuições de dados enviesados, o MM-CUBING apresenta resultados superiores em comparação com

qualquer um dos algoritmos anteriores. Porém, identifica-se, de imediato, uma deficiência que necessita claramente de ser melhorada: o excessivo consumo de memória para efectuar o processamento do cubo icebergue, motivado, principalmente, pelas chamadas recursivas necessárias para calcular as agregações. Em investigação futura, à semelhança do STAR-CUBING, seria interessante estender a metodologia de factorização do *lattice* de correlações a outros métodos optimizados de processamento de cubos. Para além disso, a análise e agrupamento de dados semelhantes pode também beneficiar com este tipo de abordagem. Decorrente do STAR-CUBING, um novo algoritmo híbrido é proposto na literatura direccionado, especialmente, para o processamento paralelo de *iceberg querys* de grande dimensão. Note-se que a maioria das soluções analisadas até ao momento, para o problema do processamento de *iceberg cubes*, operam numa perspectiva centralizada, ou seja, aplicadas a um único sistema central de processamento analítico, responsável pela gestão de todas as consultas e interrogações geradas e pelo desempenho do cubo. Importa, por isso, evidenciar neste trabalho também os algoritmos que operam em ambientes paralelos e distribuídos, de modo a aumentar o desempenho no cálculo de cubos icebergue. O algoritmo 6) PnP (*Pipe n'Prune*) [Chen et al., 2005] revela-se promissor, na medida em que assenta numa estrutura sólida, constituída pela integração da agregação de dados *Top-down*, com a capacidade *Bottom-up* de redução de dados, conseguida através da aplicação da filtragem APRIORI.

Anteriormente, na secção 3.5, foi efectuada uma análise extensiva ao desempenho do PnP para um conjunto alargado de cenários. Repare-se que no cálculo de interrogações icebergue, os algoritmos BUC e STAR-CUBING apresentam intervalos de dados densos e de enviesamento significativo onde o primeiro supera o segundo e vice-versa. Em ambos os casos, o PnP consegue um comportamento muito próximo das melhores lógicas de selecção, nas diferentes situações, o que o torna numa alternativa interessante em aplicações onde a estabilidade do desempenho sobre um conjunto alargado de parâmetros de entrada é importante. No processamento de interrogações icebergue em memória externa, observa-se uma perda mínima na eficiência do algoritmo, quando a memória principal é totalmente consumida. Aliás, na prática, repare que o tempo de execução em memória externa do PnP é duas vezes menor em comparação com o tempo de cálculo da mesma interrogação icebergue,

na memória principal. No cálculo paralelo e distribuído de *iceberg cubes*, o PnP apresenta uma escalabilidade interessante e favorece o aumento do desempenho, à medida que vão sendo adicionados mais processadores ao motor de processamento global do algoritmo. No geral, o algoritmo PnP tem um comportamento favorável em conjuntos de dados esparsos e densos. Distingue-se, sobretudo, no cálculo eficiente de *iceberg cubes* que derivam de dados reais, especialmente, em situações de difícil resolução pelos métodos sequenciais (dada a elevada dimensionalidade, cardinalidade e dimensão). Outras abordagens compreenderam a adaptação de métodos icebergue em função das necessidades específicas de cada universo de utilização. É exemplo disso, o método 7) C-CUBING [Xin et al., 2006] que integra os algoritmos icebergue MM-CUBING e STAR-CUBING com os benefícios propostos por outro método de redução de dados (*closed cubes*) [Lakshmanan et al., 2002]. Apesar do domínio de aplicação ser diferente, importar referir algumas conclusões que os algoritmos icebergue introduziram no processamento de *closed (iceberg) cubes* [Lakshmanan et al., 2003]. Na literatura indicada anteriormente, encontram-se referências a alguns testes de desempenho efectuados a estes algoritmos, com variações de cardinalidade, enviesamento, patamar mínimo de suporte e dependências de dados.

Por exemplo, o *closed* MM-CUBING tem um comportamento muito interessante quando a filtragem icebergue domina o processamento do cubo. Por outro lado, o *closed* STAR-CUBING tem um desempenho superior em situações de baixa cardinalidade. Como trabalho futuro, sugere-se mais investigação no domínio dos *closed cubes* e a análise de padrões frequentes de utilização, para simplificar e optimizar o processo de cálculo das agregações. Recentemente, foram propostas mais duas soluções para processamento de *iceberg cubes*: o algoritmo 8) IX-CUBING [Jian et al., 2007], dirigido exclusivamente para o processamento optimizado de *iceberg querys* sobre dados no formato XML e o 9) MT-CUBING [Li et al., 2008], que integra uma solução *multi-tree* híbrida, que permite processar estas estruturas através de abordagens *Top-down* e *Bottom-up*, de modo a beneficiar das vantagens de cada uma delas (inclusive da filtragem APRIORI). Qualquer uma das soluções propostas carece de mais investigação e, sobretudo, de mais testes comparativos com outros algoritmos icebergue de modo a avaliar-se efectivamente o seu desempenho real. Uma outra nota sobre o algoritmo

IX-CUBING: apesar da codificação XML utilizar uma filosofia de dados semelhante aos restantes algoritmos icebergue, a técnica de cálculo diferencia-se, sobretudo, devido à notória complexidade introduzida pela manipulação de estruturas XML semi-estruturadas. Algum trabalho futuro a realizar passará, seguramente, pela simplificação destas estruturas de modo a facilitar o cálculo icebergue sobre as estruturas multidimensionais de dados. Sobre o algoritmo MT-CUBING, dado que é baseado numa estrutura em árvore, existem à partida, limitações claras a nível da utilização da memória disponível para o cálculo das agregações, no caso do número de nós ser muito elevado. Sendo o MT-CUBING um algoritmo híbrido que integra as abordagens *Top-down* e *Bottom-up*, é necessário considerar as restrições de desempenho de cada uma delas. Mesmo assim, os estudos comparativos disponíveis na literatura revelam que estes dois algoritmos conseguem atingir um desempenho superior, em diferentes cenários, comparativamente com o BUC, quando confrontados com o cálculo dos mesmos *iceberg cubes*. Do estudo efectuado, o algoritmo 10) CT-CUBING [Cho et al., 2005] aparece em último lugar pois utiliza uma abordagem diferente face às restantes soluções já mencionadas. Os autores propõem investigar o cálculo de *iceberg cubes* através do acesso a diferentes tabelas do *Data Warehouse*. Inesperadamente, as avaliações efectuadas demonstram que esta solução pode ainda ser mais eficiente a nível de tempo e espaço em relação à solução tradicional de uma tabela única materializada.

Contudo, dos testes de desempenho assinalados na secção 3.5 é possível identificar algumas lacunas que necessitam de especial atenção. Destaca-se, logo em primeiro lugar, o consumo excessivo de memória que pode decorrer da necessidade em armazenar toda a tabela de factos numa estrutura em árvore, para processamento. Outra situação que pode ser melhorada é a essência da própria estrutura em árvore gerada durante o processamento. Note-se que esta solução foi implementada para extinguir a necessidade da leitura integral da tabela de factos cada vez que é processada uma célula icebergue local. Todavia, um número agravado de "ramos" e nós significa um custo exponencial a nível do tempo de processamento e espaço de armazenamento, necessário para calcular o cubo icebergue. Mesmo assim, o CT-CUBING evidencia a sua eficiência e escalabilidade em *Data Warehouses* de grandes dimensões, sendo por isso, a abordagem mais próxima da realidade empresarial. Em comparação com outros

algoritmos, principalmente com o BUC, os resultados obtidos demonstram que o CT-CUBING é consistentemente mais escalável e eficiente em qualquer tipo de distribuição de dados, devido, sobretudo, ao mecanismo optimizado de geração de células icebergue agregadas e atribuição de assinaturas (ou rótulos) que permitem reconhecer univocamente as células a calcular. Em suma, relativamente aos algoritmos de optimização propostos foi mostrada a sua aplicação no cálculo de estruturas multidimensionais de dados, focando as técnicas de optimização icebergue para redução dos dados de um hipercubo, condições estas, essenciais, para obtenção de um sistema de processamento analítico robusto, fiável e escalável, capaz de responder expeditamente às decisões mais complexas, consequentes das transformações do mundo empresarial moderno. O detalhe da avaliação realizada sobre cada algoritmo icebergue revelou-se de grande interesse em termos de qualidade e aplicabilidade das soluções, o que justifica claramente a utilização de técnicas icebergue no cálculo de cubos, em cenários simples e complexos. Mesmo sem ser possível distinguir um "vencedor" único, os estudos efectuados revelam que, numa perspectiva centralizada, os algoritmos da família STAR relevam ser os mais promissores. Por outro lado, o CT-CUBING demonstra ser o mais eficaz em *Data Warehouses* de grandes dimensões, sendo a abordagem mais próxima da realidade empresarial actual. Numa perspectiva distribuída, o algoritmo PnP demonstra ser uma solução escalável e dinâmica, que beneficia de todas as vantagens do cálculo paralelo potenciado pela distribuição da carga de processamento.

A sistematização da adequação de cada algoritmo icebergue proposto poderá ser avaliada em detalhe na grelha que sintetiza as principais características de cada um, face ao problema de selecção de cubos (ver anexos). De salientar que a informação disponibilizada pretende ser um complemento importante na decisão de escolha do algoritmo mais apropriado para um determinado cenário de geração de *iceberg cubes*. Desmistificados os algoritmos icebergue, interessa, finalmente, ressalvar a importância das técnicas de optimização de cubos (sobretudo no domínio icebergue) de modo a justificar a sua utilização em situações reais. Já por diversas vezes se discutiu a importância do tempo de resposta das interrogações envolvidas no processo de tomada de decisões. A maioria do trabalho existente na literatura sobre este domínio tem-se desenvolvido no contexto do cubo de dados. No esforço de reduzir o tempo

de resposta das interrogações e tornar o processo de consulta mais eficiente, recorre-se ao processo de materialização de vistas, que incide sobre determinadas partes do cubo. Dos métodos tradicionais de selecção de estruturas multidimensionais, existe um conjunto de técnicas avançadas de optimização que permitem reduzir ou comprimir o cubo de dados original, de modo a atenuar as desvantagens decorrentes do conhecido problema de selecção de vistas, revertido em custos temporais - selecção, materialização e manutenção - e de espaço - armazenamento das agregações. Repare-se que ao percorrer uma vista materializada, apesar do aumento no poder de consulta sobre esses dados, a hierarquia de cada dimensão é fixa (dimensões categóricas). As interrogações efectuadas nesse conjunto ficam limitadas às operações de *drill-down* ou *roll-up* ao longo da hierarquia fixa. Um outro inconveniente resultante da materialização de vistas está relacionado com a necessidade de fixar o valor que se pretende medir e a respectiva função agregadora de interesse. Isto significa que as interrogações que necessitam de medir diferentes valores utilizando diferentes funções de agregação irão requerer o cálculo de subcubos maiores.

Na prática, uma solução próxima da óptima passaria por: a) empregar uma técnica de redução ou compressão de cubos de modo a reduzir o espaço de armazenamento; b) utilizar um método que não exigisse a fixação antecipada da discretização das hierarquias ao longo de cada dimensão de consulta e c) que seja capaz de tratar cada dimensão como um alvo potencial de medida, com suporte a múltiplas funções de agregação, sem aumentar os custos de armazenamento. As abordagens tradicionais, como já referido, encontram-se limitadas pela necessidade de fixar, de antemão, as agregações das dimensões. Portanto, se um cubóide é materializado considerando, por exemplo, a "idade", como dimensão agregada, só poderá ser usado para responder a interrogações agregadas nesse domínio restrito. Não pode ser utilizado, por exemplo, para responder a questões como o "salário médio das pessoas de uma determinada faixa etária". O processamento eficiente dessas interrogações exige a concretização de mais subcubos de dados, resultando numa maior sobrecarga do espaço de materialização. Dado que o tamanho do cubo cresce exponencialmente com o aumento da dimensionalidade, mesmo um simples subcubo de dados pode significar requisitos de armazenamento expressivos (dai ser necessário limitar as estruturas multidimensionais de

dados a algumas dimensões). Mesmo assim, a materialização de vistas continua a ser o método mais utilizado para optimizar qualquer tipo de resposta a consultas OLAP. Todavia, a importância do desempenho destes sistemas impôs novas alternativas para o problema de materialização de vistas: não importa apenas materializar, é necessário saber também quais os cubóides mais benéficos para serem concretizados, de modo a reduzir tempo de processamento e espaço de armazenamento. Na prática, importa perceber quais as vistas mais interessantes do ponto de vista de negócio que devem ser materializadas e se, possível, aliar esta acção com algoritmos optimizados de compressão ou redução de cubos, de modo a considerar apenas as hipóteses com valor analítico credível, e que, de facto, sejam úteis no processo de tomada de decisões. Nesse contexto, são introduzidos, como resposta ao problema de selecção de cubos, os algoritmos de redução de dados que permitem materializar apenas uma percentagem da estrutura de dados, proporção essa obtida através de uma avaliação reflectida de quais os subcubos com valor significativo, que poderão ser benéficos para as consultas OLAP.

Aqui, a geração de *iceberg cubes* assumiu um papel fundamental, não só pela sua definição (evidenciada no capítulo 3) mas também pelas vantagens que figura no processamento de cubos. À partida, destacam-se, de imediato, duas vantagens evidentes: as estruturas icebergue permitem efectuar uma análise rápida sobre as regras de associação entre as agregações, na medida em que só utilizam os padrões mais frequentes. Esta situação é visível através da dinâmica que estas estruturas introduziram na manipulação do *lattice* multidimensional de dependências. Para além disso, os testes de desempenho realizados demonstram que os algoritmos icebergue têm um desempenho muito superior no processamento de cubos em distribuições muito dispersas, dada a capacidade que têm em seleccionar apenas as agregações que, de facto, têm valor analítico. Todas as outras células são filtradas e retiradas do cálculo, reflectindo-se num aumento significativo do desempenho e diminuição dos custos de armazenamento. A selecção adequada pode traduzir-se em ganhos muito elevados, conseguindo-se, com uma percentagem baixa do espaço de materialização total do subcubo, valores praticamente idênticos para responder às interrogações colocadas (em relação à concretização completa do cubo). Surpreendentemente, muitas são as referências na literatura

que indicam que grande parte das células de um cubo nem sequer são usadas para responder às interrogações colocadas pelos agentes de decisão. Ao utilizar apenas as agregações significativas, é possível representar em apenas 20% do espaço disponível, o cubo de dados completo (reflectindo-se, logicamente, em aumentos notáveis de tempo e espaço). Das inúmeras vantagens apontadas aos algoritmos icebergue são, da mesma forma, citados alguns inconvenientes que têm sido referidos, de certa maneira, ao longo desta secção, decorrentes das exiguidades herdadas pelo problema de materialização de vistas. As discussões existentes na literatura revelam que a aplicação de cubos icebergue tem sido demonstrada somente em ambientes de baixa dimensionalidade. Isto significa que ainda não foi possível testemunhar efectivamente a sua aplicabilidade em sistemas OLAP actuais e reais, cujo nível de complexidade, dimensionalidade e cardinalidade seja muito elevado. Para além disso, repare-se que os métodos icebergue são apenas variações do modelo conceptual actual do cubo de dados, e por isso, não podem ser consideradas soluções totalmente eficazes. No caso de um *iceberg cube*, por exemplo, este não pode ser actualizado incrementalmente e o patamar de suporte mínimo definido não pode ser imediatamente reajustado, de acordo com as necessidades. Na prática, as técnicas de redução de dados apenas podem atingir benefícios limitados já que tendem a ter esta característica estática, depois de se materializar o cubo.

É, por isso, necessário mais do que a aplicação de técnicas de redução ou compressão de cubos. O trabalho futuro nestes domínios passará, certamente, por aliar as técnicas de optimização com sistemas inteligentes capazes de prever as necessidades analíticas e recalibrar dinamicamente a materialização dos subcubos em função disso. Note-se que esta abordagem tende a reduzir a dispersão do cubo resultante, mas precisa que sejam desenvolvidas heurísticas adicionais para decidir quais os agregados mais adequados que devem ser materializados. Aqui deve ser considerado também o perfil de utilização, permitindo a referida possibilidade de selecção das estruturas mais apropriadas. Porventura, em direcção à reestruturação dinâmica, deve ser equacionado o duplo suporte de manutenção integral e incremental, o que permitirá descrever os algoritmos e condições arquitecturais que possibilitarão uma actualização mais fina das estruturas multidimensionais. Isto permitirá ultrapassar os problemas mais comuns do cálculo icebergue, potenciando a criação de um

modelo conceptual capaz de avaliar cada subcubo dinamicamente, que suporte múltiplas funções de agregação (sem aumentar os custos de armazenamento), adequadas para responder instantaneamente a qualquer interrogação analítica despoletada pelos utilizadores no processo de tomada de decisão.

4.2. Contribuições da Dissertação

Este trabalho, claramente focado na optimização de estruturas multidimensionais de dados, apresenta duas grandes linhas de investigação: as abordagens de selecção para o problema de processamento de cubos (capítulo 2) e as propostas de solução optimizadas para reduzir os custos decorrentes dos primeiros, sob a forma de cubos icebergue (capítulo 3). Por outro lado, acrescente-se que o âmbito espacial de aplicação das soluções de optimização foi sendo sucessivamente estendido e também a abrangência dos próprios modelos (suporte a novas características) foi sendo igualmente alargada. Desta forma, as publicações foram reflectindo o trabalho realizado, acompanhando o seu desenvolvimento temporal. Esta dissertação, de algum modo, reflectiu também essa abordagem, já que, naturalmente, a organização temporal acompanhou a evolução na complexidade dos próprios modelos e algoritmos. Relativamente às lógicas de selecção icebergue analisadas, foi mostrada a sua aplicação em ambientes de processamento analítico, de repositório único ou distribuído, face a uma multiplicidade de cenários analíticos. Os estudos de desempenho analisados permitiram sistematizar as características de cada algoritmo e criar uma base rica de conhecimento sobre este domínio que poderá ser utilizada como ponto de partida para qualquer trabalho que pretenda evoluir nesta área de investigação.

De salientar, também, que os algoritmos desenvolvidos revelaram um bom desempenho quando submetidos a uma utilização intensiva, apesar de ainda estarem longe dos resultados esperados, com aplicação real. Para além disso, a escalabilidade dos algoritmos relativa à complexidade do modelo dimensional OLAP, levanta alguns problemas. A inclusão de conceitos de programação dinâmica na génese de cada algoritmo icebergue, se permitiu tempos de execução baixos e assim, uma escalabilidade imediata, pode, para esquemas

complexos, constituir uma limitação: o tamanho das estruturas de dados necessárias ao suporte dos valores materializados pode tornar-se excessiva, impondo a utilização de capacidades de memória muito elevadas. Esta limitação manifesta-se duplamente nos algoritmos de custos, discutidos anteriormente no capítulo 2. Além de necessitarem da informação relativa às dependências entre cada conjunto de subcubos, precisam também dos valores dos pesos relativos aos custos de interrogação e manutenção. Para esse problema, uma possível solução poderá ser a adopção de um compromisso: gerar as dependências e pesos quando requeridos, guardando-os para utilização futura numa cache, recorrendo a uma política de admissão e remoção, a seleccionar, de entre as disponíveis. Para finalizar, e como se depreende das propostas indicadas, percebe-se que o trabalho constitui uma espécie de *survey* sobre o domínio dos cubos icebergue, no processo de optimização de selecção de vistas a materializar. Na verdade, a extensão do problema a solucionar e o conjunto de áreas de conhecimento envolvidas, permite criar um suporte documental de base que pode alargar, em muito, novos trabalhos sobre a problemática de selecção de estruturas multidimensionais.

Especialmente na vertente das soluções de distribuição das estruturas multidimensionais muito foi aqui avaliado e estudado, mas muito há ainda por fazer. Se as soluções propostas são já abrangentes, podendo ser aplicadas em arquitecturas diversificadas e heterogéneas na optimização das estruturas multidimensionais, importa avaliá-las melhor, para compreender mais cabalmente as suas especificidades de aplicação e, também, incluir os algoritmos de optimização já previstos e outros, a surgir neste domínio em constante evolução. De facto, as arquitecturas actuais a nível de optimização disponibilizam já um conjunto alargado de opções de fácil utilização e num leque alargado de abordagens possíveis dos sistemas de processamento analítico, de modo a permitir uma escalabilidade sustentada e simplificada, a preços controlados. A colaboração futura, mais alargada, permitirá, decerto, aprofundar o conhecimento em alguns quadrantes do problema, porventura, até agora, menos tratados, mas permitidos pela infra-estrutura existente. Na verdade, a flexibilidade das estruturas analíticas modernas irá permitir acomodar novas ideias e prosseguir por novos rumos de investigação nesta área. O próprio desenvolvimento icebergue será uma área de evolução e novos modelos conceptuais de criação de cubos serão certamente potenciados.

Bibliografia

[Agarwal et al., 1996] S. Agarwal, R. Agarwal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. “On the computation of multidimensional aggregates”. In Proc. of the 22nd VLDB Conf., pp. 506-521, 1996.

[Agrawal & Srikant, 1994] R. Agrawal and R. Srikant. “Fast algorithms for mining association rules”. In Proc. of the 20th VLDB Conf., pp. 487-499, Chile, 1994.

[Bae & Lee, 2000] J. Bae and S. Lee. "Partitioning Algorithms for the Computation of Average Iceberg Queries", DAWAK, 2000.

[Baralis et al., 1997] E. Baralis, S. Paraboschi and E. Teniente. “Materialized View Selection in a Multidimensional Database”. In: Proceedings of the 23rd International Conference on Very Large Data Base (VLDB), Greece, pp. 156-165, 1997.

[Barbara et al., 1997] D. Barbara and M. Sullivan. “Quasi-cubes: Exploiting approximation in multidimensional database”. SIGMOD Record, pp. 12-17, 1997.

[Bauer & Lehner, 2003] A. Bauer and W. Lehner. “On Solving the View Selection Problem in Distributed Data Warehouse Architectures”. In Proceedings of the 15th International Conference on Scientific and Statistical Database Management, pp. 43-51, 2003.

[Belo, O., 2000] O. Belo. "Putting Intelligent Personal Assistants Working on Dynamic Hypercube Views Updating". Proceedings of 2nd International Symposium on Robotics and Automation (ISRA), México, 2000.

[Beyer & Ramakrishnan, 1999] K. Beyer and R. Ramakrishnan. "Bottom-up computation of sparse and iceberg cubes". SIGMOD, 1999.

[Blum & Dorigo, 2004] C. Blum and M. Dorigo, "Deception in ant colony optimization", Proc. ANTS 2004, Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science, Vol. 3172, Springer, Germany, pp. 119-130, 2004.

[Chaudury & Dayall, 1997] S. Chaudury and U. Dayall: "An Overview of Data Warehouse and OLAP Technology". ACM SIGMOD Record 26(1), pp. 65-74, 1997.

[Chen, 1976] P. P-S. Chen. "The Entity Relationship Model – Towards a Unified View of Data", in ACM TODS Vol. 1, No. 1, 1976.

[Chen & Roussopoulos, 1994] C. Chen and N. Roussopoulos. "The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching". Proceedings of the International Conference on Extending Database Technology, 1994.

[Chen et al., 2005] Y. Chen, F. Dehne, T. Eavis and A. Chaplin, "PnP: Parallel and External Memory Iceberg Cube Computation", In Proc. 21st Int. Conf. on Data Engineering (ICDE), 2005.

[Chirkova et al., 2001] R. Chirkova, A. Y. Halevy and D. Suciu. "A formal perspective on the view selection problem". Proc. of VLDB, pp. 59–68, 2001.

[Cho et al., 2005] M. Cho, J. Pei, D. Cheung, "Cross Table Cubing: Mining Iceberg Cubes from Data Warehouses", SIAM, 2005.

[Codd. et al., 1993] E. F. Codd, S. B. Codd and C. T. Sulley: “Providing OLAP (On-Line Analytical Processing) to User Analysts: An IT Mandate”. Technical Report, 1993.

[Derakhshan et al., 2006] R. Derakhshan, F. Dehne et al. "Simulated Annealing for Materialized View Selection in Data Warehousing Environment", IIIS, 2006.

[Desphand et al., 1998] P. Desphand, K. Ramasamy, A. Shukla et al. “Caching Multidimensional Queries Using Chunks”. Proceedings of ACM SIGMOD, 1998.

[DeWitt & Gray, 1992] D. DeWitt and J. Gray. "Parallel database systems: the future of high performance database systems". Communication ACM 35(6), pp. 85–98, 1992.

[Diosan & Oltean, 2006] L. Diosan and M. Oltean. "Evolving the structure of the particle swarm optimization algorithms," in Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization, vol. 3906, pp. 25-36, 2006.

[Dorigo et al., 1996] M. Dorigo, M. Maniezzo and A. Colorni: “The Ant System: Optimization by a Colony of Cooperating Agents”. In IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26(1), pp. 29-41, 1996.

[Eberhart & Kennedy 1995] R. Eberhart and J. Kennedy. “A new Optimizer Using Particle Swarm Theory”. In Proc. Os the Sixth International Symposium on Micro Machine and Human Science, IEEE Service Center, Japan, pp. 39-43, 1995.

[Fang et al., 1997] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. “Computing iceberg queries efficiently”. In Proc. of the 24th VLDB Conf., New York, pp. 299-310, 1997.

[Garcia-Molina et al., 2000] H. Garcia-Molina, J. Ulman and J. Widom. "Database System Implementation", Prentice Hall, 2000.

[Gilbert et al., 2001] A. Gilbert et al., "Surfing wavelets on streams: one-pass summaries for approximate aggregate queries", VLDB, 2001.

[Graefe, 1993] G. Graefe, “Query Evaluation Techniques for Large Databases”, ACM Computing Survey, 25, 2, pp. 73-170, 1993.

[Gray et al., 1996] J. Gray, A. Bosworth, A. Layman and H. Pirahesh: “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals”. Proceedings of the 12th International Conference on Data Engineering, IEEE, 1996.

[Griffin & Likkin, 1995] T. Griffin and L. Likkin. “Incremental Maintenance of Views with Duplicates”. In Proceeding of the ACM SIGMOD, International Conference on Management of Data, CA, pp. 23-25, 1995.

[Gupta & Mumick, 1999] H. Gupta and I. Mumick. “Selection of Views to Materialize under a Maintenance-Time Constraint”. In: Proceedings of the International Conference on Database Theory, 1999.

[Gupta, A. et al., 1995] A. Gupta, V. Harinarayan and D. Quass. “Aggregate-Query Processing in a Data Warehousing Environments”. In Proceedings of the 21st International Conference on Very Large Databases (VLDB), Switzerland, 1995.

[Gupta, H. et al., 1997] H. Gupta, V. Harinarayan, A. Rajaraman and J. Ullman: “Index Selection for OLAP”. In: Proceedings of the Intl. Conf. on Data Engineering, UK, pp. 208-219, 1997.

[Gupta, 1995] A. Gupta and I. Mumick. "Maintenance of Materialized Views: Problems, Techniques, and Applications". Data Engineering Bulletin, Vol. 18, No. 2, 1995.

[Gupta, A. et al., 1993] A. Gupta, T. Mumick and V. Subrahmanian: "Maintaining Views Incrementally". In Proceedings of ACM SIGMOD, International Conference on Management of Data, Washington, 1993.

[Gupta, H., 1997] H. Gupta: "Selection of Views to Materialize in a Data Warehouse". In Proceedings of the 6th International Conference on Database Theory, Lecture Notes in Computer Science, vol. 1186, pp. 98-112, 1997.

[Han & Kamber, 2006] J. Han and M. Kamber, "Data Mining Concepts and Techniques 2nd edition", Morgan Kaufmann, 2006.

[Han et al., 2000] J. Han, J. Pei and Y. Yin. "Mining Frequent Patterns without Candidate Generation". SIGMOD, 2000.

[Han et al., 2001] J. Han, J. Pei, G. Dong and K. Wang. "Efficient computation of iceberg cubes with complex measures", SIGMOD, pp. 1-12, 2001.

[Harinarayan et al., 1996] V. Harinarayan, A. Rajaraman and J. Ullman. "Implementing Data Cubes Efficiently". Proceedings of ACM SIGMOD, Canada, pp. 205-216, 1996.

[Holland, 1992] J. Holland: "Adaptation in Natural and Artificial Systems". MIT Press, Cambridge, (2nd edition), 1992.

[Horng et al., 1999] J. Horng, Y. Chang, B. Liu and C. Kao: "Materialized View Selection Using Genetic Algorithms in a Data Warehouse". In Proceedings of World Congress on Evolutionary Computation, D.C., 1999.

[Imielinski et al., 2002] T. Imielinski, L. Khachiyan and A. Abdulghani. “Cube grades: Generalizing association rules. Data Mining and Knowledge Discovery”, pp. 219–258, 2002.

[Informatica, 1997] Informatica: “Enterprise-Scalable Data Marts: A New Strategy for Building and Deploying Fast, Scalable Data Warehousing Systems”. White Paper, 1997.

[Inmon, 1996] W. H. Inmon: “Building the Data Warehouse”, second edition. Wiley Computer Publishing, 1996.

[Ioannidis & Poosala, 1995] Y. Ioannidis and V. Poosala. "Histogram-Based Solutions to Diverse Database Estimation Problems", IEEE Data Engineering, Vol. 18, No. 3, pp. 10-18, 1995.

[Jamil & Modica, 2001] H. Jamil and G. Modica. “A View Selection Tool for Multidimensional Databases”. Proceedings on the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Hungary, Lecture Notes in Computer Science 2070, Springer, Berlin, pp. 237-246, 2001.

[Jian et al., 2007] F. Jian, J. Pei and A. Fu, "IX-Cubes: Iceberg Cubes for Data Warehousing and OLAP on XML Data", 2007.

[Kalnis & Papadias, 2001] P. Kalnis and D. Papadias: “Proxy-Server Architectures for OLAP”. Proceedings of the ACM SIGMOD, International Conference on Management of Data, CA, pp. 367-378, 2001.

[Kalnis et al., 2002a] P. Kalnis, N. Marnoulis and D. Papadias, “View Selection Using Randomized Search”. Data Knowledge Engineering, vol. 42, number 1, pp. 89-111, 2002.

[Kalnis et al., 2002b] P. Kalnis, W. Ooi, D. Papadias and K. Tan: "An Adaptive Peer-to-Peer Network for Distributed caching of OLAP Results". Proceedings of ACM SIGMOD International Conference on Management of Data, Wisconsin, 2002.

[Karayannidis & Sellis, 2001] N. Karayannidis and T. Sellis: "SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes". Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW), Switzerland, 2001.

[Kaser & Lemire, 2005] O. Kaser and D. Lemire, "Attribute Value Reordering For Efficient Hybrid OLAP", Information Sciences, Vo. 176, No 16, pp. 2279-2438, 2005.

[Kennedy & Eberhart, 1995] J. Kennedy and R. Eberhart: "Particle Swarm Optimization". In Proceedings of IEEE International Conference on Neural Networks, IEEE Service Center, Australia, pp. 1942-1948, 1995.

[Kennedy & Eberhart, 1997] Kennedy, J. and Eberhart, R.C. "A Discrete Binary Version of the Particle Swarm Optimization Algorithm". In Proc. of the 1997 Conference on Systems, Man and Cybernetics (SMC), pp. 4104-4109, 1997.

[Kimball, 1996] R. Kimball: "Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses". John Wiley & Sons, 1996.

[Kotidis & Roussopoulos, 1999] Y. Kotidis and N. Roussopoulos: "Dynamat: A Dynamic View Management System for Data Warehouses". Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 371-382, 1999.

[Lakshmanan et al., 2002] L. Lakshmanan et al. "Quotient Cubes: How to Summarize the Semantics of a Data Cube". VLDB, 2002.

[Lakshmanan et al., 2003] L. Lakshmanan et al. “QC-Trees: An Efficient Summary Structure for Semantic OLAP”. SIGMOD, 2003.

[Lazaridis & Mehrotra, 2001] I. Lazaridis and S. Mehrotra. "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure", SIGMOD, 2001.

[Leela et al., 2002] K. Leela, P. Tolani and J. Haritsa. "On Incorporating Iceberg Queries in Query Processors", Dept. of Computer Science & Automation, Indian Institute of Science, Bangalore, India, 2002.

[Li et al., 2008] X. Li, H. Hamilton et al., "The Multi-Tree Cubing algorithm for computing iceberg cubes", Department of Computer Science, University of Regina, Regina, Saskatchewan, S4S-0A2, Canada, 2008.

[Liang et al. 2001] W. Liang, H. Wang and Orlowska. “Materialized View Selection under the Maintenance Cost Constraint”. In Data and Knowledge Engineering, 37(2), pp. 203-216, 2001.

[Lin & Kuo 2004] Y. Lin and Kuo, I-C. “A Genetic Selection Algorithm for OLAP Data Cubes”. In Knowledge and Information Systems, Springer-Verlag London Ltd, Volume 6, Number 1, pp. 83-102, , 2004.

[Loureiro & Belo, 2006] J. Loureiro and O. Belo, "Swarm Intelligence in Cube Selection and Allocation for Multi-Node OLAP Systems". To appear in Proceedings of the International Conference on Systems, SCSS, 2006.

[Loureiro & Belo, 2007a] J. Loureiro and O. Belo, “Metamorphosis Algorithm for the Optimization of Multi-Node OLAP Systems”, Universidade do Minho, Portugal, 2007.

[Loureiro & Belo, 2007b] J. Loureiro and O. Belo, "Optimization of distributed OLAP Cubes with an Adaptive Simulated Annealing Algorithm", Universidade do Minho, Portugal, 2007.

[Maniezzo et al., 2001] V. Maniezzo, A. Carboraro, M. Golfarelli and S. Rizzi: “An ANTS Algorithm for Optimizing the Materialization of Fragmented Views in Data Warehouses: Preliminary Results”. Proceedings of the EVO Workshops on Applications of Evolutionary Computing, vol. 2037, Springer-Verlag, pp. 80-89, 2001.

[Maniezzo et al., 2004] V. Maniezzo, L. Gambardella and F. Luigi, "Ant Colony Optimization", Optimization Techniques in Engineering. Springer-Verlag, 2004.

[Maniezzo, 1999] V. Maniezzo: “Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. INFORMS Journal on Computing, 11(4), pp. 358-369, 1999.

[Matias & Segal, 1999] Y. Matias and E. Segal, "Approximate iceberg queries", Technical reports, Department of Computer Science, Tel Aviv University, Israel, 1999.

[Mumick et al., 1997] I. Mumick, D. Quass and B. Mumick: “Maintenance of Data Cubes and Summary Tables in a Warehouse”. Proceedings of ACM SIGMOD International Conference of Management of Data, Arizona, pp. 100-111, 1997.

[Munneke et al., 1999] D. Munneke, K. Wahlstrom and M. Mohania: “Fragmentation of Multidimensional Databases”. Proceedings of 10th Australasian Database Conference, Auckland, pp. 153-164, 1999.

[Nadeau & Teorey, 2004] T. Nadeau and T. Teorey. "OLAP Query Optimization in the Presence of Materialized Views", Proceedings of HICCS, 2004.

[O'Neil & Graefe, 1995] P. O'Neil and G. Graefe. "Multi-Table Joins through Bitmapped Join Indices" SIGMOD Record, 1995.

[Park et al., 2003] C. Park, M. Kim and Y. Lee: "Usability-based Caching of Query Results in OLAP Systems". Journal of Systems and Software, Vol. 68, pp. 103-119, 2003.

[Parsopoulos & Vrahatis, 2005] K. Parsopoulos and M. Vrahatis. "Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization". Natural Computing, 1 (2-3), pp. 235-306, 2005.

[Ramakrishnan & Gehrke, 2000] R. Ramakrishnan and J. Gehrke, "Database Management Systems", McGraw Hill Book Company, 2000.

[Ross & Srivastava, 1997] K. Ross and D. Srivastava. "Fast computation of sparse data cubes". In Proc. of the Int. Conf. on Very Large Databases, pp. 116-125, 1997.

[Roussopoulos, 1982] N. Roussopoulos, "The Logical Access Path Schema of a Database", IEEE Trans. on Software Engineering. N. Roussopoulos, "View Indexing in Relational Databases", ACM TODS, 1982.

[Rozin & Margaliot, 2007] V. Rozin and M. Margaliot, "The Fuzzy Ant", IEEE Computational Intelligence Magazine, 2007.

[Sapia et al., 1999] C. Sapia, M. Blaschka, G. Höfling and B. Dinter: "Extending the E/R Model for the Multidimensional Paradigm". In Advances in Database Technologies (ER Workshop Proceedings), Springer-Verlag, pp. 105-116, 1999.

[Sapia, 2000] C. Sapia: "PROMISE: Modeling and Predicting User Query Behavior in Online Analytical Processing Environments". Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DAWAK), 2000.

[Sarawagi et al., 1996] S. Sarawagi, R. Agrawal and A. Gupta. "On computing the data cube". Technical Report RJ10026, IBM Almaden Research Center, California, USA, 1996.

[Scheuermann et al., 1996] P. Scheuermann, J. Shim and R. Vingralek. "WATCHMAN: A Data Warehouse Intelligent Cache Manager". Proceedings of 22nd International Conference on Very Large Data Base (VLDB), India, pp. 51-62, 1996.

[Sedgewick, 1990] R. Sedgewick. "Algorithms in C". Chapter 8, pp. 112. Addison-Wesley Publishing Company, 1990.

[Shao et al., 2004] Z. Shao, J. Han and D. Xin. "MM-Cubing: Computing Iceberg Cubes by Factorizing the Lattice Space". SSDBM, 2004.

[Shim et al., 1999] J. Shim, P. Scheuermann and R. Vingralek: "Dynamic Caching of Query Results for Decision Support Systems". Proceedings of 11th International Conference on Scientific and Statistical Database Management, Ohio, 1999.

[Shukla et al., 1998] A. Shukla, P. Deshpande and J. Naughton: "Materialized View Selection for Multidimensional Datasets". In: Proc. of Very Large Data Bases, VLDB, 1998.

[Shukla et al., 2000] A. Shukla, P. Deshpande and J. Naughton: "Materialized View Selection form Multicube Data Models". Proceedings of Advances in Database Technology (EDBT), 7th International Conference on Extended Database Technology, pp. 269-284, 2000.

[Sismanis et al., 2002] Y. Sismanis, A. Deligiannakis, N. Roussopoulos and Y. Kotids. "Dwarf: Shrinking the PetaCube". SIGMOD, 2002.

[Sismanis et al., 2004] Y. Sismanis and N. Roussopoulos. "The Polynomial Complexity of Fully Materialized Coalesced Cubes". Proceedings of the Thirtieth international conference on Very large data bases - Volume 30. VLDB, 2004.

[Soutyina & Fotouhi, 1997] E. Soutyina and F. Fotouhi: "Optimal View Selection form Multidimensional Database Systems". In Proceedings of International Database Engineering and Applications Symposium, pp. 309-318, 1997.

[Stonebraker et al., 1994] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah and C. Staelin: "An Economic paradigm for Query Processing and Data Migration in Mariposa". In Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, USA, 28-30, pp. 58-68, 1994.

[Taniar, 2009] D. Taniar, Progressive Methods in Data Warehousing and Business Intelligence: Concepts and Competitive Analytics, pp. 132- 152, 2009.

[Theodoratos & Bouzeghoub, 2000] D. Theodoratos and M. Bouzeghoub: "A General Framework for the View Selection Problem for Data Warehouse Design and Evolution". Proceedings 3rd ACM International Workshop on Data Warehouse and OLAP, pp. 1-8, 2000.

[Theodoratos & Selis, 1997] D. Theodoratos and T. Sellis: "Data Warehouse Configuration". Proceedings of the 23rd International Conference on Very Large Databases, Greece, pp. 126-135, 1997.

[Theodoratos, 2004] D. Theodoratos, "Constructing search spaces for materialized view selection, in DOLAP". Proceedings of the 7th ACM international workshop on Data warehousing and OLAP. USA. pp. 112-121, 2004.

[Timos, 1988] S. Timos, "Multiple-Query Optimization". ACM Trans. Database Syst. 13(1): pp. 23-52, 1988.

[Timos & Kyuseok, 1994] S. Timos and S. Kyuseok, "Improvements on a Heuristic Algorithm for Multiple-Query Optimization", Knowledge Engineering, 1994.

[Valduriez, 1987] P. Valduriez. “Join indices: ACM Transactions on Database Systems”. TODS, 12(2), pp. 218-246, 1987.

[Wagner & Yin, 2001] A. Wagner and Y. Yin. “Iceberg-cube computation with PC clusters”. Proceedings of 2001 ACM SIGMOD Conference on Management of Data, 2001.

[Wang et al., 2002] W. Wang, J. Feng, H. Lu and J. X. Yu. “Condensed Cube: An Effective Approach to Reducing Data Cube Size”. ICDE. 2002.

[Xin et al., 2003] D. Xin, J. Han, X. Li and B. Wah. “Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration”. VLDB. 2003.

[Xin et al., 2006] D. Xin, Z. Shao, J. Han and H. Liu. “C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking”. Technical Report UIUCDCS-R-2005-2648, Department of Computer Science, UIUC, 2006.

[Yang et al., 1997] J. Yang, K. Karlapalem and Q. Li: “Algorithm for Materialized View Design in Data Warehouse Environment”. Proceedings of the 23rd International Conference on Very Large Databases, Greece, pp. 136-145, 1997.

[Yu et al., 2004] J. Yu, C. Choi, G. Gou and H. Lu: “Selecting Views with Maintenance Cost Constraints: Issues, Heuristics and Performance”. Journal of Research and Practice in Information Technology, Vol. 36, No. 2, 2004.

[Zhang et al., 2001] C. Zhang, X. Yao and J. Yang: “An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment”. IEEE Trans. on Systems, Man and Cybernetics, Part C, Vol. 31, 2001.

[Zhang et al., 1999] C. Zhang, X. Yao and J. Yang: “Evolving Materialized Views in Data Warehouses”. Proceedings of the IEEE Congress on Evolutionary Computation (CEC), USA, pp. 823-829, 1999.

[Zhao et al., 1997] Y. Zhao, P. Desphande and J. F. Naughton: “An Array-Based Algorithm for Simultaneous Multidimensional Aggregates”. In Proc. of the ACM SIGMOD Conf., pp. 159-170, 1997.

[Zharkov, 2008] A. Zharkov, "On Using Materialized Views for Query Execution in Distributed Real-Time Database Management Systems", SYRCoDIS, 2008.

[Zhuge et al., 1995] Y. Zhuge, H. Garcia-Molina, J. Hammer and J. Widom: “View Maintenance in a Warehousing Environment”. In: Proceeding of the ACM SIGMOD, International Conference on Management of Data, San Jose, 1995.

Referências WWW

[1] http://wikipedia.org

Enciclopédia on-line, com conteúdos de acesso livre e escritos de forma colaborativa por pessoas de todo o mundo. Em cada artigo há hiperligações para outros o que permite um rápido conhecimento de todos os assuntos relacionados com tema em pesquisa. Pretende-se que o conteúdo da Wikipedia seja factual, notável, verificável através de fontes externas, e apresentado de forma neutral, com citações a fontes externas. Um conjunto de políticas e guias de orientação são aplicados com esse objectivo.

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

Portal da *Transaction Processing Performance Council* (TPC). A TPC é uma empresa sem fins lucrativos fundada para definir *benchmarks* relativos a processamento e bases de dados e a disseminar dados de desempenho TPC objectivos e verificáveis para a indústria. Neste site encontram-se recomendações acerca das bases de dados e a forma de efectuar os testes que são padrão e, que permite avaliar comparativamente os diversos servidores ou bases de dados. O site disponibiliza também a lista actualizada comparativa dos servidores e bases de dados, relativa aos diversos *benchmarks* definidos, que avaliam um servidor ao executar: 1) o processamento transaccional típico sobre uma base de dados (TPC-C), centrado sobre as actividades principais (transacções) num ambiente de vendas; 2) o processamento típico num sistema de suporte à decisão (TPC-H), envolvendo grandes volumes de dados e execução de interrogações com um alto grau de complexidade, dando resposta a questões críticas de

negócio; 3) as aplicações e serviços Web, que simulam as actividades típicas de uma aplicação B2B num servidor de aplicações transaccional a operar em ambiente contínuo.

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

Biblioteca digital e motor de busca com centenas de milhares de artigos científicos disponíveis para consulta, incluindo referências e outras ligações a fontes de metadados como a DBLP e o portal ACM. O objectivo da *CiteSeer* é melhorar a disseminação e acesso a literatura científica e académica. Trata-se de um serviço que pode ser acedido por todos, sendo considerado como fazendo parte do movimento de acesso aberto que tenta alterar a forma de publicação de artigos científicos e académicos, com o objectivo de permitir um maior acesso à literatura científica.

[4] http://citeseerx.ist.psu.edu

Biblioteca digital e motor de busca com centenas de milhares de artigos científicos, vocacionado principalmente para a literatura em informática e ciência de informação. O *CiteSeerX* visa melhorar a divulgação da literatura científica e proporcionar melhorias na funcionalidade, uso, disponibilidade, custo, abrangência, eficiência e rapidez no acesso ao conhecimento científico e académico. Ao invés de criar apenas "mais uma" biblioteca digital, o *CiteSeerX* pretende oferecer recursos para acesso a algoritmos, dados, metadados, serviços, técnicas e SW que podem ser utilizados para promover a continuidade do trabalho científico nas áreas das ciências e tecnologias de informação.

[5] http://www.dw-institute.com

O *Data Warehousing Institute* (TDWI) é dedicado a ajudar as organizações a melhorar a sua compreensão e uso da inteligência para o negócio através da educação dos agentes de decisão e profissionais de SI, no desenvolvimento adequado de estratégias e tecnologias de *Data Warehousing*. Promove a partilha de informação acerca das melhores práticas e lições recolhidas do dia-a-dia. Organiza conferências internacionais e cursos nos domínios do *Data Warehousing*, para além de publicar o *Journal of Data Warehousing* e muitas outras iniciativas, tal como o muito popular *Flash Report*. Também promove a edição de livros,

compila casos de estudo e *white papers*, empreende pesquisas e suporta programas de investigação. No site do instituto, é disponibilizada muita informação e hiperligações sobre a temática do *Data Warehousing*.

[6] http://ieeexplore.ieee.org/

Trata-se de um portal de acesso a literatura técnica acerca de engenharia electrotécnica, computação e electrónica. Disponíveis artigos de revistas e colóquios (um pouco abaixo dos 1.5 milhões). Muitos artigos são de acesso geral, outros de acesso restrito. Muitos dos artigos aqui disponíveis podem ser acedidos, de forma gratuita, através do portal de informação relatado na referência [8].

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

É um portal de acesso a artigos publicados pela ACM (todos os artigos publicados pela ACM, incluindo os 50 anos de arquivos). Também permite aceder a livros, artigos, colóquios, teses de doutoramento, de mestrado e relatórios técnicos de outras proveniências, perfazendo um total de quase 1 milhão de itens. Os resumos das obras são de acesso livre para qualquer utilizador, mas a visualização do conteúdo, é de acesso restrito.

[8] http://b-on.pt

É uma biblioteca portuguesa de conhecimento online que reúne as principais editoras de revistas científicas internacionais, de modo a oferecer um conjunto vasto de artigos científicos (disponíveis on-line) que abranjam a maior parte das áreas científicas e estimulem as condições de acesso universal ao saber, por parte da comunidade científica e académica, procurando gerar economias de escala e promovendo as condições de universalidade de acesso à produção científica. Permitiu o acesso, em 2004, a mais de 3500 publicações electrónicas de seis editoras de referência internacional, nas principais áreas de investigação científica e académica. Pretende-se com a iniciativa criar condições para alavancar as condições de acesso, utilização e difusão desse conhecimento, esperando-se que venha a ser um grande contributo para aumentar a produção e inovação da comunidade científica portuguesa em todo o mundo.

[9] http://books.google.com/

O *Google Book Search* é um serviço de busca da empresa Google que permite fazer pesquisas em milhares de livros digitalizados. Parte da ideia de publicar na internet grande parte dos livros existentes no planeta. A ferramenta foi lançada no final do ano de 2004. Pode ser utilizado da mesma forma que o motor de busca na internet, ou seja, títulos, frases ou palavras-chave podem ser pesquisados na base de dados de obras disponíveis. De forma a evitar problemas de direitos de autor, apenas são mostradas na integra as obras do domínio público. Mesmo assim, as pesquisas podem ser efectuadas tanto em obras registadas como públicas, para servir de referência.

[10] http://illimine.cs.uiuc.edu

O *IlliMine* é um projecto *open source* que disponibiliza várias ferramentas para mineração de dados e operação de sistemas inteligentes. Actualmente, o projecto incorpora diversos algoritmos nas seguintes áreas de investigação: *Data Cubing*, *Association Mining* e *Sequential Mining*. Para além disso, o site disponibiliza um conjunto de demonstrações que permite gerar cubos de dados com a aplicação de diversos algoritmos (incluindo cubos icebergue). Adicionalmente, a geração de cubos pode ser totalmente personalizada, através do controlo dos parâmetros de *input* indicados pelo utilizador. Isto permite ter uma ideia real de quanto tempo demora a executar cada algoritmo, de acordo com os dados escolhidos. Outro dos recursos deste site é a disponibilização de artigos e *white papers* sobre as temáticas relacionadas com estruturas multidimensionais de dados.

Anexos

A1. Pseudocodigo de Algoritmos Icebergue

De modo a compilar os algoritmos icebergue mais predominantes, em ambas as perspectivas de processamento, veja-se o pseudocodigo de cada uma das lógicas de selecção.

Perspectiva Centralizada: Algoritmos Icebergue da família *STAR*

Algoritmo 1: Top-down Computation [24]

```
Function TDC(arrOrder, size)
Inputs:
        arrOrder[]: attributes to order the select statement by
        size: number of attributes in arrOrder
Globals:
        minsup: minimum support
         numTuples: number of tuples in the database table
        results: table that contains results of TDC()
Locals:
        arrNext[]: hold the tuple being examined
        arrHold[]: the last unique tuple examined
        strOrder: string containing ordering of fields
        i, j: counters
        curTable: temporary table that holds results of order by query
Method:
1. strOrder = arrOrder[0];
```

[24] O pseudocodigo do algoritmo original *MultiWay* foi extraído do artigo [Zhao et al., 1997].

```
2. for i = 1 to size - 1
3.          strOrder = strOrder & "," & arrOrder[i]
4. end for
5. Run ("select " & strOrder & " from " & tableName & "  order by " & strOrder);
6. for i = 0 to size - 1
7.          arrHold[i] = curTable[0, i];
8.end for
9.for i = 0 to numTuples
10.         arrNext[0] = curTable[i , 0];            // first attribute of current tuple
11.for j = 0 to size - 1
12.         if j > 0 then
13.                   arrNext[j] = arrNext[j - 1] & "-" & curTable[i , j];
14.         end if
15.         if arrNext[j] != arrHold[j] then
16.                   if count[j] >= minsup then
17.                             Run ("insert into results (Combination, Count)
18.                             values (" & arrHold[j] & "," & count[j];
19.                   end if
20.         count[j] = 0;
21.         arrHold[j] = arrNext[j];
22.         end if
23.         count[j]++;
24.         end for
25. end for
26. return(0);
```

Algoritmo 2: Bottom-up Computation [25]

Inputs:

Input: The relation to aggregate.
Dim: The starting dimension for this iteration.

Globals:

Constant numDims: The total number of dimensions.
Constant cardinality[numDims]: The cardinality of each dimension.
Constant minsup: The minimum number of tuples in a partition for it to be output.
OutputRec: The current output record.
DataCount [numDims]: Stores the size of each partition.
DataCount[i] is a list of integers of size
Cardinality[i].

Outputs:

One record that is the aggregation of input.
Recursively, outputs CUBE (dim, . . . , numDims) on input (with minimum support).

Method:

```
Procedure BottomUpCube (input, dim)
1.Aggregate(input); // Places result in outputRec
```

[25] O pseudocodigo do algoritmo original BUC foi extraído do artigo [Beyer & Ramakrishnan, 1999].

```
2. if input.count() == 1 then // Optimization
        WriteAncestors(input[0], dim); return;
3. write outputRec;
4. for d = dim ; d < numDims ; d++ do
5.      let C = cardinality[d];
6.      Partition(input, d, C, dataCount[d]);
7.      let k = 0;
8.      for i = 0 ; i < C ; i++ do // For each partition
9.              let c = dataCount[d][i]
10.             if c >= minsup then // The BUC stops here
11.                     outputRec.dim[d] = input[k].dim[d];
12.                     BottomUpCube(input[k . . . k+c], d+1);
13.             end if
14.             k += c;
15.     end for
16.     outputRec.dim[d] = ALL;
17. end for
```

Algoritmo 3: Star-Cubing [26]

Inputs:

(1) A relational table R

(2) an iceberg condition, min sup (taking count as the measure)

Outputs:

The computed iceberg cube.

Each star-tree corresponds to one cube-tree node, and vice versa.

Method:

```
Begin
        scan R twice, create star-table S and star-tree T;
        output count of T:root;
        call starcubing(T; T:root);
End
Procedure starcubing(T; cnode)// cnode: current node {
1. for each non-null child C of T's cube-tree
2.      insert or aggregate cnode to the corresponding position or node in C's star-tree;
3. if (cnode:count ¸ min sup) {
4.      if (cnode 6= root)
5.              output cnode.count;
6.      if (cnode is a leaf)
7.              output cnode.count;
8.      else { // initiate a new cube-tree
9.              create CC as a child of T's cube-tree;
10.             let TC as CC's star-tree;
11.              TC:root's count = cnode:count;
12.     }
```

[26] O pseudocodigo do algoritmo STAR-CUBING foi extraído do artigo [Xin et al., 2003].

```
13. }
14. if (cnode is not a leaf)
15.         call starcubing(T; cnode:first child);
16. if (C_C is not null) {
17.         call starcubing(T_C; T_C:root);
18.         remove C_C from T's cube-tree; g
19. if (cnode has sibling)
20.         call starcubing(T; cnode:sibling);
21. remove T;
}
```

Perspectiva Distribuída: Algoritmos Icebergue da família *PnP*

Algoritmo 4: Pipe' n Prune sequencial, na memória principal [27]

Input:

R[1..*n*]: a table representing the raw data set consisting of *n* rows *R*[*i*], *i* = 1, . . . , *n*; *v*: identifier for a group-by of *R*; *pv*: a prefix of *v*.

Output:

The iceberg data cube.

Locals:

k = |*v*| − |*pv*|; *Rj* : tables for storing rows of *R*;
b[1..*k*]: a buffer for storing *k* rows, one for each group-by *v*1 … *vk* ; *h*[1..*k*]: *k* integers;
i, *j* : integer counters.

Method:

```
1. h[1..k] = [1..1]; b[1..k] = [null..null].
2: for i = 1..n do
3:         for j = k..1 do
4:                 if (b[j] = null) OR (the feature values of b[j ] are a prefix of R[i]) then
5:                 Aggregate Rj [i] into b[j ].
6:         else
7:                 if b[j ] has minimum support then
8:                         Output b[j ] into group-by vj .
9:                         if j ≤ k −2 then
10:                                 Create a table Rj = ^R j+1[h[j]] …^R j+1[i −1].
11:                                 Sort and aggregate Rj .
12:                                 Call PnP-1(Rj , ^vj+1, vj ).
13:                         end if
14:                 end if
15:                 Set b[j] = null and h[j] = i.
16:                 end if
17:         end for
18: end for
```

[27] O pseudocodigo do primeiro algoritmo PnP, referente ao processamento em memória principal, foi extraído do artigo [Chen et al., 2005].

19: Create a table *R* [1..*n*1] by sorting and aggregating ^*R*1[1] …^*R*1[*n*].
20: Call PnP-1(*R*, ^*v* 1,∅).

Algoritmo 5: Pipe' n Prune sequencial, em memória externa [28]

Input:
R[1..*n*]: a table (stored on disk) representing the raw data set consisting of *n* rows *R*[*i*], *i* = 1, . . . , *n*; *v*: identifier for a group-by of *R*; *pv*: a prefix of *v*.
Output:
The iceberg data cube (stored on disk).
Locals:
k = |*v*| − |*pv*|; *Rj* : tables for storing rows of *R* (called *partitions*);
Fj : disk files for storing multiple partitions *Rj* ; *b*[1..*k*]: a buffer for storing *k* rows, one for each group-by *v*1, . . . , *vk* ; *h*[1..*k*]:
k integers; *i*, *j* : integer counters.
Method:
1. b[1..*k*] = [null..null]; *h*[1..*k*] = [1..1].
2. for *i* = 1..*n* (while reading *R*[*i*] from disk in streaming mode. . .) do
3. for *j* = *k*..1 do
4. if (*b*[*j*] = *null*) OR (the feature values of *b*[*j*] are a prefix of *R*[*i*]) then
5. Aggregate *Rj* [*i*] into *b*[*j*].
6. else
7. if *b*[*j*] has minimum support then
8. Output *b*[*j*] into group-by *vj* . Flush to disk if *vj* 's buffer is full.
9. if *j* ≤ *k* −2 then
10. Create a table *Rj* = ^*R j*+1[*h*[*j*]] …^*R j*+1[*i* −1].
11. Sort and aggregate *Rj* (using external memory sort if necessary).
12. Write the resulting *Rj* and an "end-of-partition" symbol to file *Fj* .
13. end if
14. end if
15. Set *b*[*j*] = *null* and *h*[*j*] = *i*.
16. end if
17. end for
18. end for
19. for *j* = *k*..1 do
20. for each partition *Rj* written to disk file *Fj* in line 11 do
21. Call PnP-2(*Rj* , ^*vj*+1, *vj*).
22. end for
23. end for
24. Create a table *R* [1..*n*1] by sorting and aggregating ^*R* 1[1] … ^*R* 1[*n*] (using external memory sort if necessary).
25. Call PnP-2(*R*, ^*v* 1, ∅).

[28] O pseudocodigo do segundo algoritmo PnP, referente ao processamento em memória externa, foi extraído do artigo [Chen et al., 2005].

Algoritmo 6: Pipe' n Prune distribuído [29]

Input:

R: a table representing a *d*-dimensional raw data set consisting of *n* rows,
stored on *p* processors. Every processor *Pi* stores (on disk) a table *Ri* of *n p* rows of *R*

min_sup: the minimum support.

Output:

The iceberg data cube (distributed over the disks of the *p* processors)

Variables:

On each processor *Pi* a set of *d* tables $T_i1 \ldots T_id$.

Method:

1.for *j* = 1..*d* do
2. Each processor P_i : Compute T_ij from T_i^{j-1} *via sequential sort* ($T_i 1 = R_i$)
3. Perform a parallel global sort on $T_1j \cup T_1j \cup \ldots \cup T_pj$
4.end for
5.for *j* = 1..*d* do
6. Each processor *Pi* : Apply Algorithm 4 to T_ij.
7. end for

[29] O pseudocodigo do terceiro algoritmo PnP, referente ao processamento de cubos icebergue em ambiente distribuído, foi extraído do artigo [Chen et al., 2005]. Repare-se que este algoritmo é uma extensão do algoritmo 4 em versão distribuída.

A2: Adequação das propostas icebergue face a diferentes cenários

Algoritmo Icebergue	Tipo de Processamento *	Agregação Simultânea	Particionamento e Filtragem	Distribuições Densas	Distribuições Esparsas	Distribuições Enviesadas	Dimensionalidade	Cardinalidade	Patamar Suporte Mínimo
1) MultiWay	Top-down	Sim	Não	Forte	Fraco	Fraco	Baixa	Baixa	Baixo
2) BUC	Bottom-up	Não	Sim	Fraco	Forte	Fraco	Alta	Alta	Alto
3) H-Cubing	Top-down / Bottom-up	Sim	Sim	Forte	Fraco	Forte	Baixa	Baixa	Alto
4) Star-Cubing	Top-down / Bottom-up	Sim	Sim	Forte	Médio	Forte	Baixa	Média	Indiferente
5) MM-Cubing	Factorização	Sim	Sim	Forte	Forte	Forte	Alta	Alta	Alta
6) Pipe 'n Prune	Top-down / Bottom-up	Sim	Sim	Forte	Forte	Forte	Média / Alta	Média / Alta	Indiferente
7) C-Cubing	Top-down / Bottom-up	Sim	Sim	Forte	Forte	Forte	Média / Alta	Média / Alta	Alto
8) IX-Cubing	XML Tree / Bottom-up	Não	Sim	Fraco	Forte	Fraco	Média	Média	Alto
9) MT-Cubing	Top-down / Bottom-up	Sim	Sim	Forte	Forte	Forte	Alta	Alta	Alto
10) CT-Cubing	Top-down / Bottom-up	Sim	Sim	Forte	Forte	Forte	Alta	Alta	Alto

* Método de processamento do Lattice multidimensional de dependências

Tabela 9. Comparativo dos métodos icebergue em diferentes cenários