

Melhorando o Desempenho do Processamento de Consultas Drill-Across em Ambientes de Data Warehousing

Diogo Tuler Forlani¹, Cristina Dutra de Aguiar Ciferri², Ricardo Rodrigues Ciferri³

¹Departamento de Informática – Universidade Estadual de Maringá
87.020-900 – Maringá – PR – Brasil

²Departamento de Ciências de Computação – Universidade de São Paulo/São Carlos
13.560-970 – São Carlos – SP – Brasil

³Departamento de Computação – Universidade Federal de São Carlos
13.565-905 – São Carlos – SP – Brasil

dforlani@din.uem.br, cdac@icmc.usp.br, ricardo@dc.ufscar.br

Abstract. *In this paper we propose two algorithms aimed at increasing the performance of drill-across queries in data warehousing environments. The JM-G algorithm is used when numeric measures of different data warehouses are usually required together. The EG-JM algorithm extends the first one also considering storage requirements. The performance tests carried out using the TPC-H benchmark showed a huge improvement on the query performance with a reduced additional storage requirement.*

Keywords. *data warehouse, drill-across query processing.*

Resumo. *Neste artigo são propostos dois algoritmos que enfocam o aumento no desempenho do processamento de consultas drill-across em ambientes de data warehousing. O algoritmo JM-G é usado para agrupar medidas numéricas de grafos de derivação diferentes que são comumente requisitadas conjuntamente. O algoritmo EG-JM estende o primeiro algoritmo considerando requisitos de espaço. Os testes realizados com o benchmark TPC-H mostraram uma grande melhoria no desempenho do processamento das consultas frente a um espaço de armazenamento adicional bem reduzido.*

Palavras-chave: *data warehouse, processamento de consultas drill-across.*

1. Introdução

Um ambiente de *data warehousing* transforma dados operacionais em informação voltada à tomada de decisão estratégica. Para tanto, oferece um conjunto de funcionalidades que possibilita que dados de diferentes provedores de informação que compõem o ambiente operacional da corporação sejam extraídos, traduzidos, filtrados, integrados e armazenados no *data warehouse* (DW). Este conjunto de funcionalidades também permite que usuários de sistemas de suporte à decisão manipulem com flexibilidade e eficiência os dados do DW, por meio de visões multidimensionais desses dados [Chaudhuri and Dayal 1997].

O DW é um banco de dados especialmente organizado para armazenar dados orientados a assunto, históricos e não-voláteis, além de integrados. Ademais, visando

uma melhoria no desempenho do processamento de consultas OLAP (*on-line analytical processing*), o DW pode armazenar visões referentes não somente aos dados detalhados coletados diretamente do ambiente operacional, mas também referentes aos dados agregados originados a partir destes dados detalhados [Monteiro and Campos 2000]. Esta forma de organização do DW em diferentes níveis de agregação garante melhores tempos de resposta [Becker and Ruiz 2004], principalmente para consultas *drill-down* e *roll-up*, as quais referenciam dados em níveis cada vez mais e menos detalhados, respectivamente. Assim, em seu projeto inicial, os dados do DW são frequentemente organizados em diferentes níveis de agregação.

Com a maturidade no uso do ambiente de *data warehousing*, novos tipos de consulta passam a ser comumente executadas contra o DW, dentre os quais este artigo enfoca consultas *drill-across*. Estas consultas comparam medidas numéricas distintas que são relacionadas entre si por dimensões em comum, sendo classificadas como consultas complexas que demandam uma grande quantidade de operações de varredura, junção e agregação. Entretanto, desde que o DW não foi inicialmente projetado para oferecer suporte para consultas *drill-across*, os tempos de resposta destas consultas tornam-se críticos. Surge, então, a necessidade de se avaliar as metas de desempenho do ambiente de *data warehousing* em uso frente a este novo tipo de consulta.

Este artigo enfoca a melhoria no desempenho do processamento de consultas *drill-across*, sem contudo degenerar o desempenho das demais consultas submetidas a ambientes de *data warehousing* em uso. Mais especificamente, o artigo enfoca a diminuição do tempo de resposta para consultas *drill-across* em ambientes com alta incidência não somente destas consultas, mas também de consultas *drill-down* e *roll-up*.

As principais contribuições do artigo são:

- a proposta do algoritmo JM-G, que visa melhorar o desempenho do processamento de consultas *drill-across* em ambientes nos quais os dados do DW são organizados em diferentes níveis de agregação; e
- a proposta do algoritmo EG-JM, que estende o algoritmo JM-G levando também em considerando restrições de espaço de armazenamento.

Os algoritmos JM-G e EG-JM foram validados por meio da realização de testes de desempenho utilizando o *benchmark* TPC-H.

Este artigo está estruturado da seguinte forma. A seção 2 resume trabalhos correlatos, enquanto que a seção 3 descreve a fundamentação teórica. Os algoritmos propostos, JM-G e EG-JM, são apresentados nas seções 4 e 5, respectivamente. A seção 6 discute os resultados de desempenho. O artigo é concluído na seção 7.

2. Trabalhos Correlatos

Este artigo enfoca a geração de grafos de derivação que possuem a junção de medidas numéricas de DW distintos. Esta geração apresenta diversos desafios, desde que deve considerar as características intrínsecas do DW, como a organização dos dados em diferentes níveis de agregação, a multidimensionalidade dos dados e as consultas OLAP frequentemente submetidas ao ambiente de *data warehousing*. Sob este aspecto, os algoritmos propostos neste artigo estendem os algoritmos de fragmentação horizontal dos dados do DW de Ciferri and Souza (2002) oferecendo suporte a consultas *drill-across*.

Cada grafo de derivação gerado pelos algoritmos propostos neste artigo pode ser considerado um fragmento dos DW originais. A produção de fragmentos em ambientes de *data warehousing* tem sido investigada no contexto de fragmentação horizontal e vertical. Com relação à fragmentação horizontal, além de Ciferri and Souza (2002), destacam-se os trabalhos de Costa and Madeira (2004), Furtado (2004a, 2004b) e Aguilar-Saborit *et al.* (2005). Em oposição a esses trabalhos, os fragmentos produzidos pelos algoritmos propostos neste artigo são fragmentos verticais. Já os trabalhos de Datta *et al.* (1998) e Golfarelli *et al.* (2004) visam a fragmentação vertical dos dados do DW. Diferentemente dos algoritmos propostos neste artigo, Datta *et al.* não enfocam a organização dos dados do DW em níveis de agregação. O algoritmo de Golfarelli *et al.* identifica, no projeto inicial do DW, quais fragmentos verticais do nível inferior do DW devem ser armazenados, incluindo a possibilidade de junção de medidas numéricas de diferentes níveis inferiores. Entretanto, Golfarelli *et al.* não vislumbram o cenário no qual podem existir vários níveis de agregação já armazenados em um DW em uso, além de enfocarem detalhadamente a análise da carga de trabalho para determinar fragmentos verticais. Neste artigo, os algoritmos propostos visam principalmente atacar a questão de como os dados dos fragmentos podem ser obtidos a partir dos DW originais em uso.

3. Fundamentação Teórica

3.1. Esquema Estrela do Benchmark TPC-H

O TPC-H é um *benchmark* voltado à tomada de decisão, sendo formado por dados sintéticos e por uma carga de trabalho composta por 22 consultas OLAP [Poess and Floyd 2000]. Este *benchmark* define uma aplicação de *data warehousing* que armazena dados históricos relativos a pedidos e vendas de uma companhia.

A definição do TPC-H foi adaptada neste trabalho para representar um esquema estrela. Um esquema estrela reflete a forma como os dados do DW são armazenados em sistemas gerenciadores de banco de dados (SGBD) relacionais [Kimball 2002]. No esquema estrela adaptado, *Lineitem* e *Partsupp* são tabelas de fatos, desde que contêm as medidas numéricas de interesse. As medidas de *Lineitem* são *quantity*, *extendedprice*, *discount* e *tax*, ao passo que as medidas de *Partsupp* são *availqty* e *supplycost*. Já *Supplier*, *Part* e *Orders* são tabelas de dimensão. Cada tabela de dimensão armazena os atributos daquela dimensão, além da chave primária. Os atributos de uma dimensão podem se relacionar por meio de hierarquias de relacionamento de atributos, as quais especificam a granularidade dos dados [Hurtado and Mendelzon 2002]. Neste trabalho, são consideradas as hierarquias *part* (p) → *brand* (b) → *MFGR* (m) e *supplier* (s) → *nation* (n) → *region* (r) para as dimensões *Part* e *Supplier*. Na segunda hierarquia, *supplier* é o atributo de menor granularidade, enquanto *region* é o atributo de maior granularidade.

3.2. Organização dos Dados em Diferentes Níveis de Agregação

O esquema estrela do TPC-H ilustra o nível de agregação inferior de um DW. Em geral, os dados do DW são organizados em diferentes níveis de agregação, desde um nível inferior que possui dados detalhados, até um nível superior que possui dados muito resumidos. Também podem existir vários níveis intermediários entre estes dois níveis que representam graus de agregação crescentes e que são determinados com base nas granularidades dos atributos das dimensões [Chaudhuri and Dayal 1997]. Assim, a

definição do TPC-H também foi adaptada neste trabalho para gerar um DW organizado em diferentes níveis de agregação. A Figura 1 mostra esta adaptação, ilustrando dois grafos de derivação: G_1 para as agregações geradas a partir de *Lineitem* e G_2 para as agregações geradas a partir de *Partsupp*.

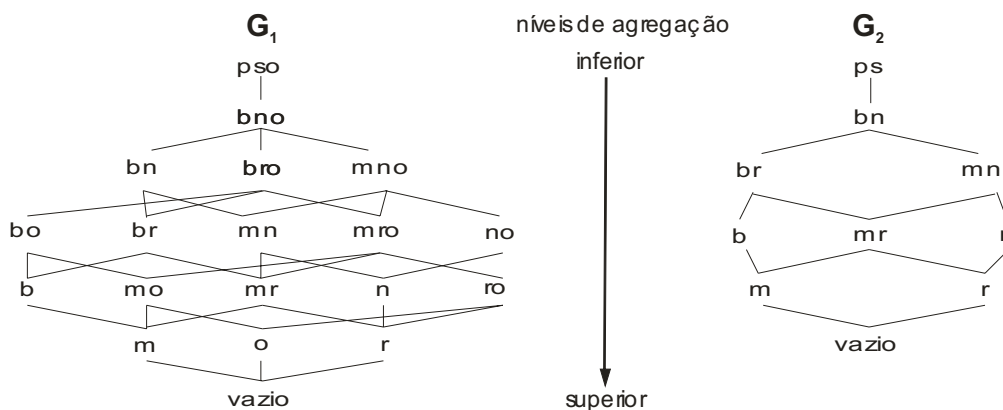


Figura 1. Grafos de derivação para *Lineitem* (G_1) e *Partsupp* (G_2).

Um grafo de derivação é um par (V, E) de conjuntos disjuntos de vértices V e arestas E . $V(G)$ representa um conjunto de agregações (i.e., visões), enquanto $E(G)$ representa um conjunto de relações de dependência entre estas agregações. Cada vértice do grafo agrega medidas numéricas sobre as dimensões presentes naquele vértice, sendo nomeado com base na granularidade do atributo de cada uma dessas dimensões. Por exemplo, o vértice $\{ps\} \in V(G_2)$ agrega as medidas numéricas *availqty* e *supplycost* contextualizadas pelos atributos de menor granularidade das dimensões *Part* e *Supplier*.

Os vértices de um grafo de derivação seguem as seguintes características de *lattice* de visões [Harinarayan *et al.* 1996, Baralis *et al.* 1997]. Uma agregação pode ser definida por meio dos dados contidos em outra agregação. Para as agregações $\{mn\}$ e $\{mno\} \in V(G_1)$, existe uma ordenação parcial que indica que $\{mn\}$ pode ser obtida por meio dos dados de $\{mno\}$. Neste trabalho, $\{mno\}$ é considerado um possível vértice gerador de $\{mn\}$. Ademais, o *lattice* de visões possui uma agregação a partir da qual todas as demais podem ser obtidas (e.g., $\{pso\} \in V(G_1)$), a qual é chamada neste artigo de agregação derivante total. Por fim, o *lattice* de visões pode possuir uma agregação completamente resumida, que pode ser calculada a partir de qualquer outra agregação, a qual é chamada de agregação vazia (e.g., $\{vazio\} \in V(G_1)$). Na Figura 1, não são representados todos os vértices que podem ser gerados a partir de $\{pso\} \in V(G_1)$ e $\{ps\} \in V(G_2)$ visando diminuir o tamanho de cada grafo e, desta forma, simplificar o entendimento deste trabalho.

3.3. Consultas Drill-Down, Roll-Up e Drill-Across

A organização dos dados do DW em diferentes níveis de agregação possibilita que os usuários de sistemas de suporte à decisão iniciem suas análises no nível superior para obter uma visão geral do negócio, podendo percorrer a hierarquia de agregação até o nível inferior à medida que dados mais específicos são necessários. Este tipo de análise ilustra o uso de consultas *drill-down*. Consultas *roll-up* representam o inverso, possibilitando que a análise dos dados seja realizada em níveis de agregação progressivamente menos detalhados. Além disso, o compartilhamento de dimensões em

comum por tabelas de fatos diferentes permite a realização de consultas *drill-across*, as quais comparam medidas numéricas distintas que são relacionadas entre si por pelo menos uma dimensão em comum [Chaudhuri and Dayal 1997].

Usando como base a Figura 1, um usuário pode solicitar as seguintes consultas: *discount, tax, availqty* por *nation* (i.e., agregação {n}); a seguir *discount, tax, availqty* por *MFGR* por *nation* (i.e., agregação {mn}); e a seguir *discount, tax, availqty* por *brand* por *nation* (i.e., agregação {bn}). Esta análise ilustra o uso de consultas *drill-down* desde que estas consultas acessam diferentes níveis de agregação e também ilustra o uso de consultas *drill-across* desde que estas consultas solicitam medidas numéricas de grafos distintos relacionados entre si pelas dimensões *Part* e *Supplier*. Esta análise corresponde às consultas C⁶ a C⁸ da seção 5.3.

4. O Algoritmo Proposto JM-G

Nesta seção é proposto o algoritmo JM-G (junção de **m**edidas numéricas sobre **g**rafos de derivação), o qual é utilizado em situações nas quais existe a necessidade de se agrupar medidas numéricas de grafos de derivação diferentes que são comumente requisitadas conjuntamente. O algoritmo JM-G produz um novo grafo de derivação que contém a junção destas medidas numéricas em termos dos atributos em comum dos grafos de derivação originais, considerando a granularidade desses atributos.

4.1. Entradas para o Algoritmo JM-G

As entradas requeridas pelo algoritmo JM-G são:

- k esquemas de DW representados por grafos de derivação G_i ($k \geq 2$ e $1 \leq i \leq k$);
- para cada atributo presente na agregação derivante total de cada G_i , o grafo que representa a hierarquia de relacionamento de atributos de sua dimensão;
- para cada dois atributos diretamente ligados na hierarquia de relacionamento, a função de mapeamento f_map que associa valores do domínio do primeiro atributo com um valor do domínio do segundo atributo; e
- o conjunto de medidas numéricas CMG_i presente na agregação derivante total de cada G_i e a função de agregação f_ag de cada medida numérica.

Como exemplos de entrada para o algoritmo JM-G, a Figura 2 ilustra dois grafos de entrada baseados no TPC-H: (i) G_3 , com os atributos *brand* (b), *region* (r) e *orders* (o) na agregação derivante total; e (ii) G_4 , com os atributos *MFGR* (m) e *nation* (n) na agregação derivante total. G_3 e G_4 , mesmo tendo iniciado suas agregações com granularidades diferentes, utilizam as mesmas hierarquias *brand* (b) \rightarrow *MFGR* (m) e *nation* (n) \rightarrow *region* (r) para as dimensões *Part* e *Supplier*, respectivamente. Para estes grafos: $CMG_3 = \{quantity, extendedprice, discount, tax\}$, $CMG_4 = \{availqty, supplycost\}$, $f_agG_3(quantity) = soma$, $f_agG_3(extendedprice) = soma$, $f_agG_3(discount) = soma$, $f_agG_3(tax) = soma$, $f_agG_4(availqty) = soma$ e $f_agG_4(supplycost) = soma$. Já exemplos de funções de mapeamento são: $f_map(brand, MFGR) = \{“Brand\#11”\} \rightarrow \{“Manufact\#1”\}$ e $f_map(nation, region) = \{“Brazil”, “Argentina”\} \rightarrow \{“America”\}$.

Foi desenvolvida uma ferramenta que semi-automatiza o processo de geração destas entradas. Esta ferramenta, a partir de uma análise dos dados de um DW mantido no SGBD Oracle®, identifica as tabelas que compõem o DW, os atributos de cada

tabela de dimensão e as medidas numéricas presentes nas tabelas de fatos. A interação com o usuário é necessária para determinar: (i) quais são tabelas de dimensão e quais são tabelas de fatos; (ii) qual a função de agregação de cada medida numérica de cada tabela de fatos; (iii) quais as hierarquias de relacionamento de atributos. De posse das informações, a ferramenta gera automaticamente o grafo de derivação correspondente a cada tabela de fatos, juntamente com seu conjunto de medidas numéricas, e também o grafo correspondente a cada uma das hierarquias de relacionamento de atributos e as respectivas funções de mapeamento.

4.2. Detalhamento do Algoritmo JM-G

A explicação do algoritmo proposto é realizada em termos da Figura 2. Nesta figura, G_3 e G_4 representam grafos de derivação de entrada, ao passo que GS representa o grafo de derivação de saída. Desde que G_3 e G_4 têm como dimensões em comum *Part* e *Supplier* e que os atributos de maior granularidade destas dimensões são respectivamente *MFGR* (*m*) e *region* (*r*), o algoritmo JM-G gera o grafo GS com as medidas $CMGS = \{quantity, extendedprice, discount, tax, availqty \text{ e } supplycost\}$ para estes atributos em comum.

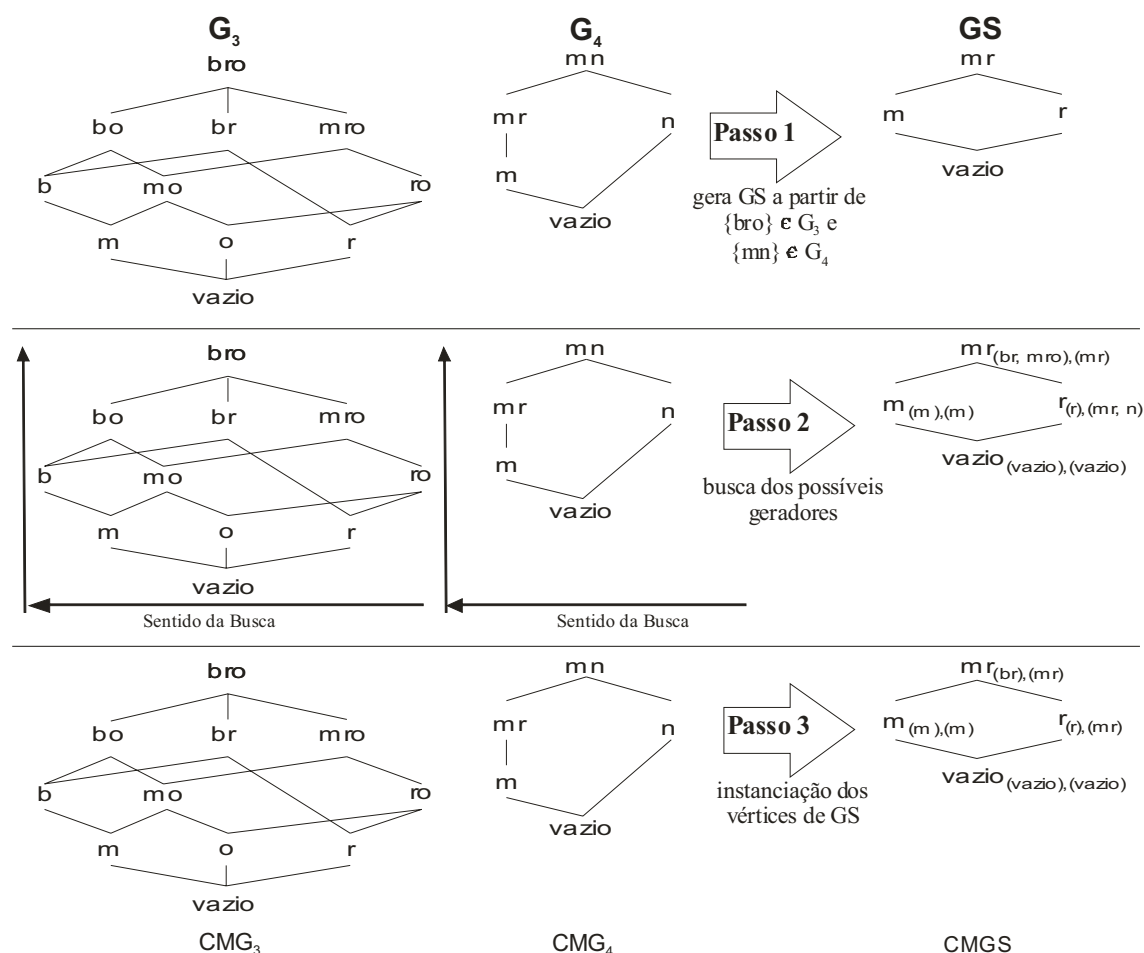


Figura 2. Passos do algoritmo JM-G.

As primeiras atividades realizadas pelo algoritmo JM-G consistem na geração de funções de mapeamento adicionais quando necessário e na criação de um esboço de grafo de derivação de saída (passo 1). Para três atributos a^1 , a^2 e a^3 de uma mesma

hierarquia de relacionamento, uma função de mapeamento adicional associa valores do domínio de um atributo a^1 a um valor do domínio de um atributo a^3 , desde que: (i) a^1 e a^3 não estejam diretamente ligados; e (ii) existam funções de mapeamento entre a^1 e a^2 e entre a^2 e a^3 . Por exemplo, se um dos grafos de entrada fosse definido a partir da hierarquia *supplier* \rightarrow *nation* \rightarrow *region*, a função adicional $f_map(\text{supplier}, \text{region})$ deveria ser criada. As funções de mapeamento adicionais são utilizadas em conjunto com as funções de mapeamento fornecidas como entrada, sem distinção, para a transformação de um vértice de menor granularidade em um de maior granularidade.

Ainda no passo 1, é criado o esboço do grafo de derivação de saída GS. Este grafo é considerado um esboço porque representa apenas os seus vértices, porém não indica como esses vértices podem ser instanciados a partir dos vértices dos grafos originais. O esboço de GS é criado com base nos atributos de maior granularidade das dimensões em comum presentes nas agregações derivantes totais de cada grafo G_i ($1 \leq i \leq k$), os quais são associados como a agregação derivante total de GS. Por exemplo, para $\{bro\} \in V(G_3)$ e $\{mn\} \in V(G_4)$ é formado o vértice $\{mr\} \in V(GS)$. Este vértice possui os atributos *MFGR* e *region*, que são os atributos de maior granularidade em comum entre $\{bro\}$ e $\{mn\}$, e não representa a dimensão *Orders* porque ela não é compartilhada por G_3 e G_4 . A partir de $\{mr\}$ e das hierarquias fornecidas como entrada, são gerados os demais vértices de GS, ou seja, $\{m\}$, $\{r\}$ e $\{\text{vazio}\}$.

No passo 2 é feita a busca dos possíveis geradores de cada vértice $vGS \in V(GS)$. Esta busca é realizada em cada $V(G_i)$ ($1 \leq i \leq k$), percorrendo-se G_i do nível de agregação superior para o nível de agregação inferior (i.e., agregação derivante total) e associando-se a vGS os vértices $v \in V(G_i)$ de forma que v seja um possível gerador de vGS e que v possua o maior nível de agregação possível. Esta busca é realizada visando-se identificar os vértices geradores de vGS com maior granularidade. Assim, a busca é finalizada ao passar de um nível de agregação menos detalhado com possíveis geradores para um nível de agregação mais detalhado. Pode ocorrer de mais do que um vértice gerador pertencente a $V(G_i)$ ser associado à vGS , uma vez que diferentes vértices geradores podem possuir o mesmo nível de agregação. Como exemplo, os possíveis geradores do vértice $\{mr\} \in V(GS)$ são $\{br\}$ e $\{mro\} \in V(G_3)$ e $\{mr\} \in V(G_4)$, fato este representado na Figura 2 por $mr_{(br, mro), (mr)}$. Esta nomenclatura indica que, para a criação do vértice $\{mr\} \in V(GS)$ no passo 3, poderá ser feita uma junção entre os vértices $\{br\}$ e $\{mr\}$ ou entre os vértices $\{mro\}$ e $\{mr\}$ dos grafos originais.

No processo de instanciação dos vértices de GS (passo 3), primeiramente é escolhido, para cada grafo, um único vértice gerador $vG_i \in V(G_i)$ ($1 \leq i \leq k$) para cada vGS , denominado $vG_i\text{escolhido}$. Para isto, é escolhida a combinação entre possíveis geradores que apresenta o menor custo de junção entre seus vértices. Este custo é determinado utilizando-se a função de custo de junção de laço aninhado indexada descrita em Elmasri and Navathe (2003), denominada neste artigo de *CJunção*.

O processo de instanciação é feito da seguinte forma para cada $vG_i\text{escolhido}$. Se as granularidades de $vG_i\text{escolhido}$ e de vGS forem diferentes, uma ou duas das transformações a seguir podem ser necessárias. Na primeira delas, se $vG_i\text{escolhido}$ possuir mais atributos que vGS , $vG_i\text{escolhido}$ deve ser transformado utilizando as funções de agregação. Por exemplo, caso $\{mr\} \in V(G_4)$ seja o vértice gerador de $\{r\} \in V(GS)$, $\{mr\}$ deve ser agregado em $\{r\}$ usando $f_agG_4(\text{availqty})$ e $f_agG_4(\text{supplycost})$

para depois ser utilizado no processo de instanciação. Já na segunda transformação, se v_{G_i} escolhido possuir atributos com menor granularidade que alguns atributos de v_{GS} , os atributos de v_{G_i} escolhido devem ser transformados utilizando primeiro as funções de mapeamento e em seguida as funções de agregação para depois ser realizado o processo de instanciação. Como exemplo, caso $\{br\} \in V(G_3)$ seja o vértice gerador de $\{mr\} \in V(GS)$, deve-se usar $f_map(\text{brand}, MFGR)$ sobre $\{br\}$ e aplicar $f_agG_3(\text{quantity})$, $f_agG_3(\text{extendedprice})$, $f_agG_3(\text{discount})$ e $f_agG_3(\text{tax})$.

Após a agregação de cada v_{G_i} escolhido ($1 \leq i \leq k$) é feita a junção destes com base nas suas dimensões em comum. Desta junção obtém-se um vértice com todas as medidas presentes em cada v_{G_i} escolhido, o qual é associado à v_{GS} . Por exemplo, tomando-se $v_{GS} = \{mr\}$, v_{G_3} escolhido agregado = $\{mr\}$, v_{G_4} escolhido = $\{mr\}$, $CMG_3 = \{\text{quantity}, \text{extendedprice}, \text{discount}, \text{tax}\}$ e $CMG_4 = \{\text{availqty}, \text{supplycost}\}$, a junção de v_{G_3} escolhido com v_{G_4} escolhido resulta em um vértice $\{mr\}$, o qual é associado à v_{GS} , com $CMGS = \{\text{quantity}, \text{extendedprice}, \text{discount}, \text{tax}, \text{availqty}, \text{supplycost}\}$.

O algoritmo JM-G produz todas as agregações que podem ser geradas a partir da agregação derivante total do grafo de saída. Uma otimização no algoritmo é a geração de apenas as agregações correspondentes às consultas *drill-down*, *roll-up* e *drill-across* que são mais frequentemente submetidas ao ambiente. Esta extensão é tratada pelo algoritmo EG-JM, que também leva em consideração requisitos de espaço.

5. O Algoritmo Proposto EG-JM

Nesta seção é proposto o algoritmo EG-JM (escolha de grafos de junção de medidas numéricas), o qual tem como objetivo a melhoria no desempenho do processamento da maior quantidade possível de consultas *drill-down*, *roll-up* e *drill-across* levando em consideração requisitos de espaço. Para isto, o algoritmo cria esboços do grafo de derivação de saída cujos vértices respondem a estas consultas de forma mais eficiente.

Os esboços do grafo de derivação de saída são analisados para verificar qual deles proporcionaria o maior ganho de desempenho para as consultas se o seu grafo existisse. O esboço escolhido é instanciado pelo algoritmo JM-G. Após a instanciação, o algoritmo EG-JM repete o processo realizando a busca pelo segundo melhor esboço a ser gerado, agora considerando a existência do grafo já produzido. O processo é repetido até que o espaço disponível não seja suficiente para armazenar outros grafos.

A função usada para calcular o espaço requerido por um esboço do grafo de saída é detalhada na seção 5.1. A seção 5.2 descreve a função usada para determinar o ganho de desempenho proporcionado por um esboço. As entradas do algoritmo EG-JM são descritas na seção 5.3, enquanto que o algoritmo é detalhado na seção 5.4.

5.1. Espaço de Armazenamento

A função f_arm (Equação 1) analisa cada vértice v de um esboço do grafo de saída com o objetivo de determinar um valor aproximado para a quantidade de bytes necessária para a geração do seu grafo de derivação EG. Esta função é baseada: (i) na quantidade de tuplas dos possíveis vértices geradores de v , ou seja, $tuplas(vG)$; (ii) na quantidade de tuplas das dimensões utilizadas nas junções, ou seja, $tuplas(d)$; (iii) na quantidade de bytes de cada atributo e medida numérica dos possíveis vértices geradores de v , denotada por $Bytes(\text{atributo})$ ou $Bytes(\text{medida})$. Esta quantidade é então multiplicada

por um valor aproximado da quantidade de tuplas de v para definir o tamanho de v (Equação 2); e (iv) na seletividade da junção (i.e., sel) entre os vértices geradores.

$$f_arm(EG) = \sum_{v=cada\ vértice\ de\ EG} tam(v) \quad (1)$$

$$tam(v) = \left(\sum_{a=cada\ atributo\ de\ v} Bytes(a) + \sum_{m=cada\ medida\ de\ v} Bytes(m) \right) * aprox_tuplas(v) \quad (2)$$

$$aprox_tuplas(v) = \prod_{vG=cada\ vértice\ gerador\ de\ v} tuplas(vG) * \prod_{d=cada\ dimensão\ utilizada\ nas\ junções} tuplas(d) \quad (3)$$

$$* sel(vG_1 \bowtie (vG_2 \bowtie d_1) \bowtie \dots \bowtie \dots vG_k)$$

A determinação da quantidade de tuplas aproximadas de v (Equação 3) é calculada pela quantidade máxima de tuplas que v pode possuir multiplicada pela seletividade da junção sel . Esta quantidade máxima de tuplas é obtida pelos produtórios da Equação 3 e indica a quantidade de tuplas do produto cartesiano entre os vértices geradores e as dimensões em questão. Já a seletividade indica a fração de tuplas que são retornadas na junção entre vértices geradores e dimensões para se obter vértices mais agregados, e entre diferentes vértices geradores para se obter v .

5.2. Ganho de Desempenho

A escolha de qual esboço do grafo de derivação de saída será gerado é determinada pelo ganho de desempenho que o seu grafo EG proporciona. Segundo a Equação 4, este ganho de desempenho é obtido pelo somatório da razão do custo de cada consulta C sem a existência de EG (i.e., $Custo(C, \emptyset)$) pelo custo de C com a existência de EG (i.e., $Custo(C, EG)$) multiplicada pela frequência de utilização de C (i.e., $freq(C)$). A função $freq(C)$ é considerada para que consultas freqüentes sejam privilegiadas com maior possibilidade de geração do grafo que as respondam mais eficientemente.

$$Ganho(EG) = \sum_{\substack{\text{para cada consulta C} \\ \text{da carga de trabalho}}} \frac{Custo(C, \emptyset)}{Custo(C, EG)} freq(C) \quad (4)$$

Como as consultas sob análise neste artigo não requerem seleções de dados, o custo de submissão de cada consulta é determinado pela função CJunção citada na seção 4.2. Ademais, desde que vários vértices de um grafo de derivação podem responder a uma mesma consulta, o custo considerado deve ser o custo da consulta submetida ao(s) vértice(s) em que a consulta produza o melhor desempenho. Na Equação 4, podem ocorrer dois casos para a razão entre os custos: (i) $Custo(C, \emptyset) = Custo(C, EG)$, indicando que a criação de EG não proporciona nenhum ganho no desempenho do processamento da consulta C; e (ii) $Custo(C, \emptyset) > Custo(C, EG)$, indicando que a criação de EG proporciona algum ganho no desempenho do processamento da consulta C.

5.3. Entradas para o Algoritmo EG-JM

Além das entradas para o algoritmo JM-G, o algoritmo EG-JM requer como entradas:

- o espaço de armazenamento disponível, em bytes, que pode ser utilizado para a criação dos grafos de derivação de saída (i.e., Esp_Disp);
- o conjunto CCT que representa as consultas da carga de trabalho. A representação de cada consulta é dividida em dois subconjuntos, um contendo

as projeções de medidas numéricas (i.e., PM) e outro contendo as projeções de atributos (i.e., PA); e

- a freqüência de submissão de cada consulta, a qual especifica a média de submissão desta consulta em um determinado período de tempo.

Os grafos de derivação G_1 e G_2 da Figura 1 e as entradas listadas na seção 4.1 ilustram algumas das entradas requeridas pelo algoritmo EG-JM. Exemplos de outras entradas, particulares deste algoritmo, são o valor de Esp_Dis (e.g., 900.000.000), o conjunto $CCT = \{\{PM^1, PA^1\}, \{PM^2, PA^2\}, \{PM^3, PA^3\}, \{PM^4, PA^4\}, \{PM^5, PA^5\}, \{PM^6, PA^6\}, \{PM^7, PA^7\}, \{PM^8, PA^8\}\}$ e as freqüências das consultas, de forma que:

- C^1 : $PM^1 = \{\text{extendedprice, quantity, supplycost}\}$; $PA^1 = \{\text{ps}\}$; $\text{freq}(C^1) = 10$;
- C^2 : $PM^2 = \{\text{extendedprice, quantity, supplycost}\}$; $PA^2 = \{\text{bn}\}$; $\text{freq}(C^2) = 15$;
- C^3 : $PM^3 = \{\text{extendedprice, quantity, supplycost}\}$; $PA^3 = \{\text{br}\}$; $\text{freq}(C^3) = 17$;
- C^4 : $PM^4 = \{\text{extendedprice, quantity, supplycost}\}$; $PA^4 = \{\text{mr}\}$; $\text{freq}(C^4) = 14$;
- C^5 : $PM^5 = \{\text{extendedprice, quantity, supplycost}\}$; $PA^5 = \{\text{m}\}$; $\text{freq}(C^5) = 12$;
- C^6 : $PM^6 = \{\text{discount, tax, availqty}\}$; $PA^6 = \{\text{mn}\}$; $\text{freq}(C^6) = 6$;
- C^7 : $PM^7 = \{\text{discount, tax, availqty}\}$; $PA^7 = \{\text{n}\}$; $\text{freq}(C^7) = 10$; e
- C^8 : $PM^8 = \{\text{discount, tax, availqty}\}$; $PA^8 = \{\text{bn}\}$; $\text{freq}(C^8) = 9$.

5.4. Detalhamento do Algoritmo EG-JM

A explicação do algoritmo proposto é realizada em termos das Figuras 1 e 3. G_1 e G_2 representam grafos de derivação de entrada, ao passo que G_{CR}^1 e G_{CR}^2 consistem em esboços do grafo de derivação de saída obtidos por meio do conjunto CCT destacado como exemplo na seção 5.3.

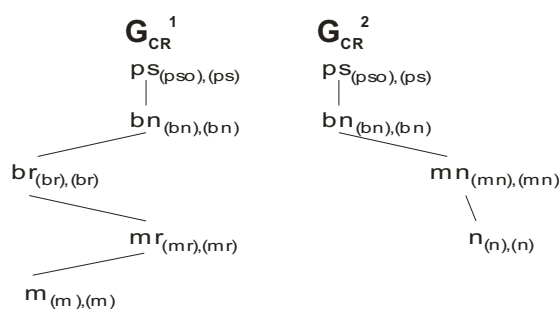


Figura 3. Exemplos de esboços criados pelo algoritmo EG-JM.

O algoritmo EG-JM executa três passos. O primeiro deles determina os possíveis esboços a serem gerados. O segundo passo cria e preenche um vetor de ganho de desempenho para estes esboços. No terceiro passo o esboço é escolhido com base neste vetor, sendo seu grafo de derivação de saída gerado pelo algoritmo JM-G.

No primeiro passo, o algoritmo EG-JM analisa o conjunto CCT fornecido como entrada, com o objetivo de agrupar as consultas em conjuntos de consultas relacionadas CR^1, CR^2, \dots, CR^t . Para que duas consultas $\{PM^i, PA^i\}$ ($1 \leq i \leq u$) e $\{PM^j, PA^j\}$ ($1 \leq j \leq u$ e $i \neq j$) estejam relacionadas, as medidas numéricas de PM^i e PM^j devem ser obtidas com a junção das medidas numéricas de dois ou mais grafos de derivação e $PM^i = PM^j$. Para o exemplo em questão, é possível determinar os seguintes conjuntos de consultas

relacionadas a partir de CCT: (i) $CR^1 = \{\{PM^1, PA^1\}, \{PM^2, PA^2\}, \{PM^3, PA^3\}, \{PM^4, PA^4\}, \{PM^5, PA^5\}\}$; (ii) $CR^2 = \{\{PM^6, PA^6\}, \{PM^7, PA^7\}, \{PM^8, PA^8\}\}$. Assim, os conjuntos de medidas numéricas de G_{CR^1} e G_{CR^2} são, respectivamente, $CMG_{CR^1} = \{\text{extendedprice, quantity, supplycost}\}$ e $CMG_{CR^2} = \{\text{discount, tax, availqty}\}$.

Cada CR^p ($1 \leq p \leq t$) é utilizado como base para a criação de um esboço do grafo de derivação de saída G_{CR^p} . Cada vértice de cada esboço corresponde a um conjunto PA referente a um conjunto PM que tem suas medidas numéricas contidas no esboço sob análise. A Figura 3 ilustra os esboços G_{CR^1} e G_{CR^2} gerados respectivamente a partir de CR^1 e de CR^2 . Ademais, cada esboço possui a agregação derivante total gerada pela junção dos respectivos grafos de derivação de entrada, possibilitando que qualquer consulta *drill-across* referente ao esboço possa ser respondida.

No segundo passo, o algoritmo de EG-JM cria um vetor de ganho de desempenho VG para determinar qual grafo de derivação de saída mais contribui para a melhoria no desempenho do processamento das consultas. Este vetor tem tamanho máximo t e possui como índice de suas colunas cada grafo G_{CR^p} ($1 \leq p \leq t$) que pode ser armazenado de acordo com o espaço disponível. Cada célula p de VG é preenchida com o ganho de desempenho (obtido pela Equação 4) proporcionado pela geração de G_{CR^p} .

Por fim, no terceiro passo, é feita a escolha do G_{CR^p} ($1 \leq p \leq t$) a ser gerado. Esta escolha seleciona o G_{CR^p} com maior valor no vetor de ganho de desempenho, o qual é então gerado pelo algoritmo JM-G (i.e., G_{CR^1}). O algoritmo EG-JM possui dois critérios de desempate, caso necessário. O primeiro calcula a soma das freqüências das consultas relativas a cada grafo que proporciona o mesmo ganho, e gera o grafo correspondente à maior soma das freqüências. Se o empate persistir, no segundo critério o algoritmo EG-JM escolhe o grafo que requer menor espaço de armazenamento. Em adição aos grafos de derivação originais utilizados como entrada, o algoritmo EG-JM também especifica como entrada para o algoritmo JM-G o grafo de derivação de saída que deve ser criado. Como resultado, o primeiro passo do algoritmo JM-G não é mais realizado (seção 4.2).

Após a geração de um grafo G_{CR^p} (e.g., G_{CR^1}), o algoritmo EG-JM executa novamente o segundo passo, com a diferença que: (i) G_{CR^p} também faz parte do DW; (ii) as consultas de CR^p são eliminadas de CCT; e (iii) o espaço disponível é diminuído da quantidade necessária à criação de G_{CR^p} . Caso haja espaço disponível, o algoritmo EG-JM refaz os cálculos do vetor de ganho VG e escolhe outro grafo a ser gerado.

6. Testes de Desempenho

6.1. Ambiente de Teste

As vantagens do uso dos algoritmos JM-G e EG-JM foram investigadas por meio de testes de desempenho realizados com o *benchmark* TPC-H. Os grafos de derivação usados nos testes são os mostrados na Figura 1, os quais seguem as especificações das seções 3.1 e 3.2. A Figura 4 define as consultas da carga de trabalho geradas a partir da consulta Q9 do TPC-H, além de suas respectivas freqüências. Estas consultas ilustram, conjuntamente, consultas *roll-up* e *drill-down* e, individualmente, consultas *drill-across*. O espaço de armazenamento disponível considerado foi de 0,05GB (i.e., $Esp_Disp = 53.687.091$ bytes). As funções de mapeamento, apesar de serem usadas nos testes, não são listadas neste artigo por serem muito longas.

Foram definidas cinco configurações de teste: (i) configuração 1: armazenamento de $G_1_completo$ e $G_2_completo$ usando várias tabelas de fatos; (ii) configuração 2: armazenamento de $G_1_incompleto$ e $G_2_incompleto$ usando várias tabelas de fatos; (iii) configuração 3: armazenamento de $G_1_completo$ e $G_2_completo$ usando uma tabela de fatos única; (iv) configuração 4: armazenamento de $G_1_incompleto$ e $G_2_incompleto$ usando uma tabela de fatos única; e (v) configuração 5: armazenamento do grafo G_{CR}^1 , em adição aos grafos originais (i.e., completos ou incompletos).

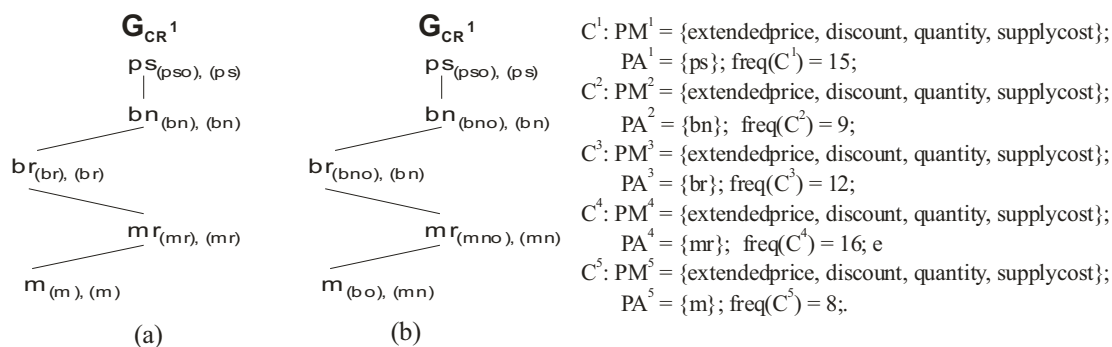


Figura 4. Consultas e Grafo G_{CR}^1 para grafos completos (a) e incompletos (b).

Para estas configurações, foram definidos três aspectos de projeto. No primeiro, os valores das medidas de cada agregação de cada grafo de entrada foram armazenados em uma tabela de fatos particular (i.e., abordagem de várias tabelas de fatos) ou todos os valores das medidas de todas as agregações de cada grafo de entrada foram armazenados em uma única tabela de fatos (i.e., abordagem de tabela de fatos única). O segundo aspecto considerou o uso de grafos de entrada completos e incompletos, os quais foram povoados com dados sintéticos gerados a partir da especificação do TPC-H para o nível inferior com 1GB. $G_1_completo$ e $G_2_completo$ são denominados completos por representarem, respectivamente, os grafos G_1 e G_2 da Figura 1. Já $G_1_incompleto$ e $G_2_incompleto$ são denominados incompletos por representarem apenas algumas agregações de G_1 e G_2 , respectivamente. $G_1_incompleto$ é composto pelos vértices $\{pso, bno, mno, bo\} \in V(G_1)$, enquanto $G_2_incompleto$ é composto pelos vértices $\{ps, bn, mn\} \in V(G_2)$. O terceiro aspecto considerou a existência apenas dos grafos de entrada (i.e., configurações 1 a 4) ou o existência conjunta dos grafos de entrada e do grafo G_{CR}^1 produzido pelos algoritmos propostos (i.e., configuração 5). Existe somente uma configuração relativa aos algoritmos propostos porque eles adotam a abordagem de várias tabelas de fatos e geram as agregações de acordo com o conjunto CCT. As Figuras 4a e 4b ilustram o grafo G_{CR}^1 produzido a partir dos vértices dos grafos completos e incompletos, respectivamente.

Em especial, para $G_1_incompleto$ e $G_2_incompleto$, as consultas C^1 a C^5 foram executadas nos vértices mais apropriados para respondê-las quando o vértice do conjunto PA da consulta não estava armazenado. Foram utilizadas as seguintes combinações de consultas e vértices de $G_1_incompleto$ e de $G_2_incompleto$: ($\{C^1\}$, $\{pso\}$, $\{ps\}$), ($\{C^2, C^3\}$, $\{bno\}$, $\{bn\}$), ($\{C^4\}$, $\{mno\}$, $\{mn\}$) e ($\{C^5\}$, $\{bo\}$, $\{mn\}$).

Os testes foram executados no SGBD Oracle9i® em um computador Pentium IV de 1.8 GHz. Os resultados coletados foram medidos em termos do tempo de resposta, do número de acessos a disco e do espaço requerido para armazenar os grafos de derivação.

6.2. Resultados de Desempenho

Considerando-se as medidas tempo de resposta e número de acessos a disco, o uso do grafo G_{CR}^1 proporcionou um expressivo ganho de desempenho no processamento das consultas C^1 a C^5 quando comparado com as demais configurações (Tabelas 1, 2, 3).

A Tabela 1 ilustra o tempo de resposta requerido pelas configurações 1 a 5 para realizar o processamento das consultas C^1 a C^5 . Enquanto a configuração 5 requereu apenas 32,066 segundos no total, as demais configurações requereram um tempo de resposta muito maior variando entre uma e duas ordens de grandeza superior. Desta forma, o uso do grafo G_{CR}^1 proporcionou uma redução no tempo de resposta entre 96,11% e 99,50% com relação às configurações 1 a 4 (Tabela 2). O desempenho é bem inferior nos grafos originais devido à necessidade de se realizar a junção de medidas contidas em diferentes grafos no momento da execução das consultas *drill-across*.

Analisando-se individualmente cada consulta, verifica-se que o tempo de resposta produzido pelas configurações 2 a 4 é muito maior, em algumas ordens de grandeza, do que o tempo de resposta produzido pela configuração 5. Desta forma, conclui-se que a escolha de grafos originais incompletos e/ou especialmente com tabelas de fatos única degenerou significativamente o desempenho no processamento de cada uma das consultas. O uso de uma tabela de fatos única mostra-se custoso por causa da necessidade de processamento de um grande número de tuplas para qualquer consulta, independentemente do nível de agregação ao qual a consulta se refere. Já o uso de grafos incompletos requer o processamento da consulta em um vértice mais detalhado do grafo quando o vértice alvo não existe no grafo, sendo, portanto, mais custoso.

Tabela 1. Tempo de resposta (em segundos) no processamento das consultas.

configuração	C^1	C^2	C^3	C^4	C^5	total: C^1 a C^5
1	823,7326	0,2300	0,1900	0,2000	0,1800	824,5326
2	823,7326	81,9180	81,4220	78,0920	71,0430	1.136,2076
3	1.958,8770	1.192,8950	1.089,0160	1.087,0830	1.090,8890	6.418,7600
4	1.333,4570	674,1780	669,3920	666,9290	666,5680	4.010,5240
5	31,5550	0,1700	0,1200	0,1300	0,0910	32,0660

Tabela 2. Redução no tempo de resposta: configuração 5 para configurações 1 a 4.

consultas	conf ₅			
	redução: conf 1	redução: conf 2	redução: conf 3	redução: conf 4
C^1	96,169267%	96,169267%	98,389128%	97,633594%
C^2	26,086957%	99,792475%	99,985749%	99,974784%
C^3	36,842105%	99,852620%	99,988981%	99,982073%
C^4	35,000000%	99,833530%	99,988041%	99,980508%
C^5	49,444444%	99,871909%	99,991658%	99,986348%
total: C^1 a C^5	96,111009%	97,177804%	99,500433%	99,200454%

Já a configuração 1 mostrou-se mais competitiva quando comparada com a configuração 5. Com exceção de C^1 , as demais consultas foram processadas na configuração 1 com tempo de resposta na mesma (ou quase na mesma) ordem de grandeza do tempo obtido com a configuração 5. No entanto, o uso da configuração 5 ainda mostrou-se bastante satisfatório desde que sempre produziu um melhor desempenho, com uma redução no tempo de resposta de C^2 a C^5 entre 26,09% e 49,44% e com uma redução muito maior no tempo de resposta de C^1 de 96,17% com relação à configuração 1 (Tabela 2). Em especial, C^1 é uma consulta realizada nos vértices do nível inferior dos grafos, os quais são muito volumosos.

Considerações similares às realizadas para a medida tempo de resposta aplicam-

se também à medida número de acessos a disco (Tabela 3). Além disto, o uso de G_{CR}^1 requereu um reduzido espaço adicional. No pior caso, o espaço adicional para o armazenamento de G_{CR}^1 em adição ao armazenamento dos grafos de entrada G_1 _incompleto e G_2 _incompleto foi de 2,238812% (Tabela 4).

Tabela 3. Redução nos acessos a disco: configuração 5 para as configurações 1 a 4.

consultas	conf ₅			
	redução: conf 1	redução: conf 2	redução: conf 3	redução: conf 4
C ¹	97,415450%	97,415450%	99,055600%	98,313589%
C ²	36,363636%	99,975336%	99,998301%	99,997098%
C ³	50,000000%	99,992953%	99,999514%	99,999171%
C ⁴	50,000000%	99,992865%	99,999514%	99,999171%
C ⁵	50,000000%	99,991691%	99,999514%	99,999171%
total: C ¹ a C ⁵	97,409257%	98,321413%	99,763675%	99,591894%

Tabela 4. Espaço de armazenamento.

configuração	bytes	GB	grafo G_{CR}^1 com relação aos grafos originais G_1 e G_2
1	1.948.204.336	1,8144	1,733213%
2	1.508.234.145	1,4047	2,238812%
3	3.024.660.218	2,8169	1,116374%
4	1.768.197.510	1,6468	1,909658%
5	33.766.534	0,0314	tamanho base

7. Conclusões

Este artigo enfocou a melhoria no desempenho do processamento de consultas *drill-across* em ambientes de *data warehousing* caracterizados pela alta incidência de consultas *drill-down* e *roll-up*. Contribuiu, desta forma:

- propondo o algoritmo JM-G, o qual é utilizado em situações nas quais existe a necessidade de se agrupar medidas numéricas de grafos de derivação diferentes que são comumente requisitadas conjuntamente; e
- propondo o algoritmo EG-JM, o qual provê a melhoria no desempenho do processamento da maior quantidade possível de consultas *drill-down*, *roll-up* e *drill-across* levando em consideração requisitos de espaço.

O aumento de desempenho, para as consultas *drill-down* e *roll-up*, é obtido pela geração de visões em diferentes níveis de agregação presentes no grafo de derivação de saída. Isto permite que consultas mais ou menos detalhadas sejam processadas nas visões do grafo que as respondam com o menor custo. Já a melhoria no desempenho do processamento de consultas *drill-across* é obtida pela criação de um grafo de saída que contém as possíveis junções entre as visões de diferentes grafos originais e, por conseguinte, de suas medidas numéricas. Isto possibilita que consultas *drill-across* sejam remanejadas para as visões do grafo obtido pela junção, portanto sem a necessidade de se realizar a junção das medidas numéricas no momento de execução destas consultas.

Os algoritmos JM-G e EG-JM foram validados por meio da realização de testes de desempenho utilizando o *benchmark* TPC-H. Os testes de desempenho mostraram um ganho de desempenho muito expressivo no processamento das consultas *drill-down*, *roll-up* e *drill-across* quando comparado com o uso apenas dos grafos de derivação originais. Este ganho variou entre 26,09% e 99,99%. O armazenamento do grafo de derivação de saída requereu um aumento no espaço de armazenamento entre 1,12% e 2,24%. Pode-se concluir, portanto, que este aumento é muito reduzido e justifica-se frente ao grande ganho de desempenho obtido no processamento das consultas.

O algoritmo EG-JM está sendo estendido para incorporar os algoritmos de fragmentação vertical propostos pelos autores deste artigo e também para determinar automaticamente o melhor algoritmo a ser usado (i.e., JM-G ou fragmentação vertical). Outra extensão consiste na análise da carga de trabalho visando identificar os subconjuntos de medidas numéricas e de atributos a serem utilizados como base para a geração dos grafos produzidos. Neste artigo, a geração do conjunto CCT definido como entrada do algoritmo EG-JM foi baseada no uso de técnicas convencionais.

Agradecimentos

Os autores agradecem o apoio financeiro das seguintes agências de fomento à pesquisa do Brasil: FAPESP (proc. N° 2005/04272-9), CNPq, CAPES e FINEP.

Referências Bibliográficas

- Aguilar-Saborit, J. *et al.* (2005) “Ad Hoc Star Join Query Processing in Cluster Architectures”, In *Proc. 7th DaWaK*, p. 200-209.
- Baralis, E. *et al.* (1997) “Materialized View Selection in a Multidimensional Database”, In *Proc. 23rd VLDB Conference*, p. 25-29.
- Becker, K. and Ruiz, D.D.A. (2004) “An Aggregate-Aware Retargeting Algorithm for Multiple Fact Data Warehouses”, In *Proc. 6th DAWAK*, p. 118-128.
- Chaudhuri, S. and Dayal, U. (1997) “An Overview of Data Warehousing and OLAP Technology”, In *SIGMOD Record*, Vol. 26, No. 1, p. 65-74.
- Ciferri, C.D.A. and Souza, F.F. (2002) “Focusing on Data Distribution in the WebD²W System”, In *Proc. 4th DaWaK*, p. 265-274.
- Costa, M. and Madeira, H. (2004) “Handling Big Dimensions in Distributed Data Warehouses using the DWS Technique”, In: *Proc. 7th DOLAP*, p. 31-37.
- Datta, A., Moon, B. and Thomas, H. (1998) “A Case for Parallelism in Data Warehousing and OLAP”, In *Proc. 9th DEXA*, p. 226-231.
- Elmasri, R. and Navathe, S.B. (2003) “Fundamentals of Database Systems”, 4th edition.
- Furtado, P. (2004a) “Workload-based Placement and Join Processing in Node-Partitioned Data Warehouses”, In *Proc. 6th DaWaK*, p. 38-47.
- Furtado, P. (2004b) “Experimental evidence on partitioning in parallel data warehouses”, In *Proc. 7th DOLAP*, p. 23-30.
- Golfarelli, M. *et al.* (2004) “Materialization of Fragmented Views in Multidimensional Databases”, In *Data & Knowledge Engineering*, Vol. 49, No.2, p. 325-351.
- Harinarayan, V. *et al.* (1996) “Implementing Data Cubes Efficiently”, In: *Proc. SIGMOD*, p. 205-216.
- Hurtado, C.A. and Mendelzon, A.O. (2002) “OLAP Dimension Constraints”, In *Proc. 21st PODS*, p. 169-179.
- Kimball, R. (2002) “The Data Warehouse Toolkit”, 2nd edition.
- Monteiro, R.R. and Campos, M.L.M. (2000) “Modelo de Custos para Seleção Dinâmica de Agregados a Materializar em Data Warehouses”, In *Proc. XV SBBD*, p. 331-345.
- Poess, M. and Floyd, C. (2000) “New TPC Benchmarks for Decision Support and Web Commerce”, In *ACM SIGMOD Record*, Vol. 29, No. 4, p. 64-71.