Documentos

Outubro, 2005 52

ISSN 1677-9274

Evolução e Desafios na *Twing Query* em XML

Embrapa

ISSN 1677-9274
Outubro, 2005

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Documentos 52

Evolução e Desafios na *Twing Query* em XML

Carla Geovana do Nascimento Macário

Campinas, SP
2005

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. André Tosello, 209
Cidade Universitária "Zeferino Vaz" – Barão Geraldo
Caixa Postal 6041
13083-970 – Campinas, SP
Telefone (19) 3789-5743 – Fax (19) 3289-9594
URL: http://www.cnptia.embrapa.br
e-mail: sac@cnptia.embrapa.br

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel Fugisawa Souza*
Editoração eletrônica: *Área de Comunicação e Negócios (ACN)*

1ª. edição on-line - 2005

Macário, Carla Geovana do Nascimento.
Evolução e desafios na *twing query* em XML / Carla Geovana do Nascimento Macário. — Campinas: Embrapa Informática Agropecuária, 2005.
21 p. : il. — (Documentos / Embrapa Informática Agropecuária ; 52).

ISSN 1677-9274

1. *Twing query*. 2. Árvores de consulta. 3. Algoritmos de consulta. I. Título. II. Série.

CDD 005.741 (21st. ed.)

Autora

Carla Geovana do Nascimento Macário
M.Sc. em Engenharia Elétrica e de Computação, Pesquisadora da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo
13083-970 - Campinas, SP
Telefone (19) 3789-5795
e-mail: carla@cnptia.embrapa.br

Apresentação

A linguagem XML é uma linguagem simples e bastante flexível, baseada em marcações. Originalmente projetada para atingir certos desafios de publicação eletrônica, seu uso se tornou tão intenso que atualmente consiste em um padrão universal para a troca de informação (dados), desempenhando um papel importante em aplicações da *web*.

Apesar de sua simplicidade, esta linguagem é capaz de representar informações de diversas fontes de dados, incluindo documentos estruturados e semi-estruturados, bases de dados relacionais e repositórios de dados. Em decorrência, os documentos XML podem apresentar uma estrutura complexa.

Dada sua potencialidade e aplicação, torna-se essencial a existência de mecanismos que permitam a recuperação destes documentos pelos usuários. Vários esforços vêm sendo realizados na busca por algoritmos que realizem esta operação, resultando em avanços constantes nesta área de pesquisa. Este trabalho apresenta o estado da arte destes algoritmos, fornecendo informações básicas que permitam ao leitor obter o conhecimento genérico deste assunto.

Eduardo Delgado Assad
Chefe-Geral

Sumário

Evolução e Desafios na *Twing Query* em XML

Carla Geovana do Nascimento Macário

Introdução

A linguagem XML (Extensible Markup Language), por sua capacidade de representar informações de diversas fontes de dados, incluindo documentos estruturados e semi-estruturados, bases de dados relacionais e repositórios de dados, tornou-se um padrão para a troca de dados entre aplicações da *web*.

Com isso, torna-se essencial a existência de mecanismos eficientes de consultas a dados em XML. Um linguagem de consulta que usa a estrutura XML deve ser capaz de expressar de maneira inteligente consultas sobre todos os tipos de dados estando eles armazenados em XML ou apenas vistos como XML via *middleware*. Por esta razão, muitas das linguagens disponíveis para este tipo de consulta expressam os documentos XML por meio de árvores, utilizando também esta estrutura para representar as consultas a serem realizadas.

Encontrar as ocorrências destas árvores de consulta, ditas *twig*, em uma base de dados XML é uma operação central no processamento de consultas XML, sejam elas em implementações relacionais de bases de dados XML ou em XML nativas. Diferente de consultas em bancos de dados relacionais, consultas *twig* requerem o reconhecimento dos relacionamentos entre os elementos envolvidos.

Vários estudos vêm sendo realizados na busca por algoritmos eficientes em termos de geração mínima de resultados intermediários. Com isso, grandes avanços têm sido obtidos no processamento dessas consultas. Este trabalho apresenta o estado da arte destes algoritmos, fornecendo ao leitor informações básicas desta área de pesquisa.

Inicialmente é apresentado o modelo de dados XML utilizado por estes algoritmos, fazendo uma breve descrição do que vem a ser uma consulta *twig*. Em seguida são descritos os principais algoritmos desenvolvidos até o momento nesta área, comentando suas principais contribuições. Por fim são apresentados novos desafios que vêm sendo propostos, indicando prováveis linhas de pesquisa futura.

Modelo de Dados e Consultas *Twig*

A linguagem XML - Extensible Markup Language é uma linguagem simples e bastante flexível, baseada em marcações. Originalmente projetada para atingir certos desafios de publicação eletrônica, seu uso se tornou tão intenso que atualmente tornou-se um padrão na troca de informação (dados), desempenhando um papel importante em aplicações da *web*. Sua importância fez com que fossem criados grupos de estudo de padronização da linguagem. Em especial, o XML Query (XQuery) Working Group, tem como missão desenvolver um padrão para consultas em documentos Web (World Wide Web Consortium, 2005).

Esta linguagem é capaz de representar informações de diversas fontes de dados, incluindo documentos estruturados e semi-estruturados, bases de dados relacionais e repositórios de dados. Então é fácil pensar que os documentos XML podem apresentar uma estrutura complexa. Isto não deixa de ser verdade, mas é possível a sua representação genérica por meio de árvores ordenadas, onde os nós representam elementos, atributos e textos e os relacionamentos do tipo pai-filho indicam pares elemento-subelemento, elemento-atributo e elemento-texto. A Fig. 1 ilustra uma representação em árvore de um documento XML. É então correto afirmar que uma base de dados XML pode ser vista como uma floresta de árvores com raízes, ordenadas e nomeadas.

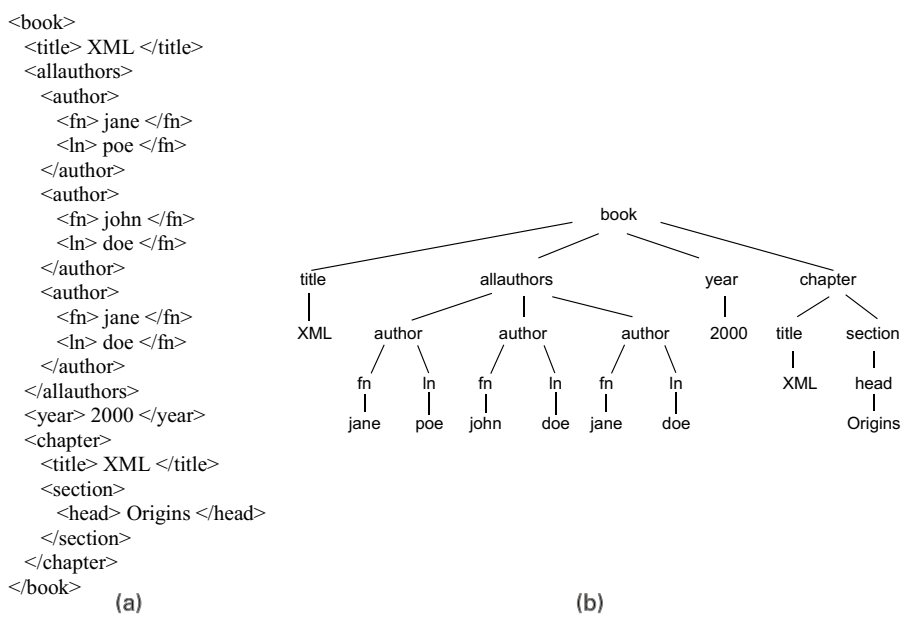

Fig. 1. Representação em árvore de documento XML.

Várias das linguagens de consultas a dados XML fazem uso desta estrutura de árvore para representar o documento, como também a consulta a ser realizada, a qual recebe o nome de *twig* padrão (pequena árvore). Os nós da *twig* padrão representam elementos, atributos e textos de uma consulta. Já suas arestas representam os relacionamentos básicos pai-filho (denotados por /) e ancestral-descendente (denotados por //). Se o número de filhos de um nó é maior que 1, trata-se de um nó ramificado. Caso contrário é um nó não-ramificado.

Por exemplo, a expressão author[fn = 'jane' and ln = 'doe'] que busca por elementos author que tenham um filho fn com conteúdo jane e um filho ln com conteúdo doe é representada pela *twig* padrão da Fig. 2 (a). Neste acaso apenas relacionamentos pai-filho foram usados. A expressão book[title = 'XML']//author[fn = 'jane' and ln = 'doe] é representada pela *twig* padrão da Fig. 2 (b).

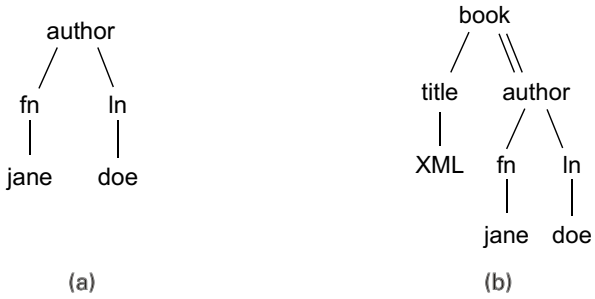

Fig. 2. Exemplos de *Twig* padrão.

Casamentos de *twig* padrão com base de dados em XML

Dada uma *twig* padrão T e uma base de dados XML D, um casamento de T em D é feito mapeando-se os nós de T para os elementos em D tal que: os nós predicados são satisfeitos pelos elementos da base de dados correspondente e os relacionamentos entre estes elementos são os mesmos que os expressos na *twig* padrão. Ou seja, um casamento de T em D corresponde a uma ocorrência da subárvore T em D. Exemplificando, considerando o documento XML da Fig. 1 (b) e a twig padrão da Fig. 2 (a), existe um casamento, que corresponde à terceira subárvore do elemento author.

Uma consulta *twig* tem como objetivo encontrar todas as ocorrências da *twig* padrão em uma base de dados XML. Tal operação é essencial no processamento de consultas XML, tanto em implementações relacionais de bases de dados XML como em XML nativas (Zhang et al., 2001).

Representando a posição dos elementos e de seus valores de dados na base XML

Existe um mecanismo chave para tornar eficiente a realização da consulta *twig*, ou seja, para buscar as ocorrências da *twig* padrão em um documento XML. Este mecanismo é a representação posicional dos elementos XML e de valores de dados na base XML, estendendo a estrutura de índice invertido usada normalmente em recuperação de informação. Esta representação foi proposta por Zhang et al. (2001) e tornou-se clássica, podendo ser considerada com o primeiro grande marco na evolução de consultas *twig*. Desde então, a maioria dos algoritmos propostos baseia-se nesta representação.

Basicamente, a posição de uma ocorrência de um elemento ou valor é representada por uma tupla (docid, início:fim, nível), onde docid é o identificador do documento; início corresponde à posição da palavra no documento XML; fim é a posição de término do elemento e nível é a profundidade do elemento no documento. Para elementos folha o valor de início e o fim é o mesmo e por esta razão, há a representação de apenas um deles. Apesar da proposta de Zhang et al. (2001) não incluir o identificador do documento, trabalhos subseqüentes ao seu propuseram esta extensão que vem sendo adotada freqüentemente. A Fig. 3 ilustra a representação posicional para o documento XML da Fig. 1.

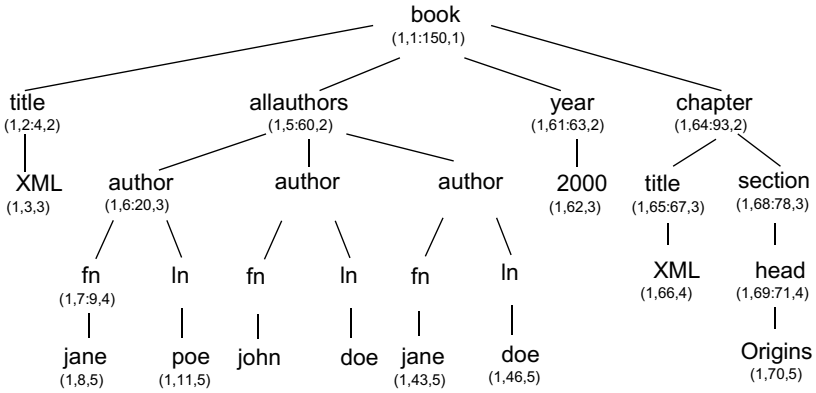

Fig. 3. Representação posicional para um documento XML.

A principal vantagem desta codificação é permitir identificar facilmente relacionamentos ancestral-descendente (A-D) e pai-filho (P-F), bem como a proximidade entre nós, tornando-a bastante eficiente para a avaliação de relacionamentos estruturais.

Dados dois nós *u* e *v*, *u* é ancestral de *v*, ou seja existe um relacionamento A-D entre eles se *u.início* < *v.início* < *u.fim*. Já *u* é pai de *v*, ou seja existe um relacionamento P-F entre ele se além da propriedade de ancestral-descendente, *u.nivel* = *v.nivel* -1. Note-se que é mais fácil checar relacionamentos A-D do que os P-F.

Evolução dos Algoritmos para Consultas *Twig*

A busca por algoritmos que permitam a execução eficiente de consultas *twig* tem motivado a realização de várias pesquisas nesta área. Isto é comprovado pelos resultados obtidos, todos relativamente recentes, sendo o primeiro deles em 2001, com o trabalho de Zhang et al. (2001). De lá para cá pode-se dizer que mais três resultados relevantes foram produzidos, os quais serão discutidos a seguir.

Basicamente a consulta *twig* divide-se em dois passos. No primeiro deles encontram-se resultados intermediários que de alguma forma satisfazem partes da *twig* padrão. No segundo passo, usando um algoritmo com base nos relacionamentos existentes na *twig* padrão e nos resultados intermediários obtidos, faz-se uma junção de parte destes resultados de maneira a obter a solução final.

Em seus trabalhos, Zhang et al. (2001) buscavam provar que, por meio de algumas adaptações, seria possível que os sistemas gerenciadores de banco de dados relacionais (SGBD) suportassem consultas em bases de dados XML de maneira eficiente. A motivação para sua pesquisa foi a tendência de que os SGBD passem a armazenar os documentos XML, tornando-se essencial a existência de mecanismos para realizar as consultas a estes dados.

Zhang et al. (2001) investigaram o uso de suporte nativo de SGBD comerciais como suporte para consultas baseadas em relacionamentos estruturais (*containement*). Para isso, propôs o uso de listas invertidas como uma extensão do mecanismo de índice invertido normalmente utilizado em recuperação de informações. Nesta extensão, adicionou informações suficientes para mapear a localização de um dado elemento em um documento XML, tornando possível responder a uma consulta *twig*, sem a necessidade de acessar os documentos originais.

Além do mecanismo posicional, propôs também um algoritmo de junção chamado Multi-predicate Merge-join (MPMGJN) a ser usado pelas listas invertidas, para encontrar todas as ocorrências de relacionamentos estruturais básicos em um documento XML.

Os autores compararam a implementação de consultas estruturais usando suporte nativo em dois sistemas comerciais e a lista invertida de propósito especial com o algoritmo MPMGJN. Os resultados mostraram que o algoritmo proposto poderia

superar o de junção padrão de um SGBD em ordem de mais de uma magnitude em se tratando de consultas estruturais. A chave para esta eficiência é a representação posicional dos elementos XML e valores *strings*, que sucintamente captura os relacionamentos estruturais entre elementos na base XML, tornando possível verificar a presença de relacionamentos (A-D e P-F) entre os elementos a partir da checagem das condições de desigualdade que estabelecem os relacionamentos estruturais.

Em seus estudos, Al-Khalifa et al. (2002) mostraram que o algoritmo proposto por Zhang et al. (2001) poderia gerar uma quantidade de operações de E/S e processamentos desnecessários, propondo assim uma nova classe de algoritmos: os de junção estrutural.

Numa evolução do trabalho proposto anteriormente, Al-Khalifa et al. (2002) tiraram vantagem da representação posicional para projetar algoritmos ótimos em termos de I/O e CPU para a realização de consultas *twig* em documentos XML. Para isso, decompôs a *twig* padrão em várias relações binárias, como ilustrado na Fig. 4, que decompõe a *twig* padrão da Fig. 2 (b). Em seguida por meio de algoritmos de junção estrutural (*tree-merge* e *stack-merge*) faz o casamento das relações binárias com a base de dados XML, para depois promover a junção destes casamentos para obtenção dos resultados finais.

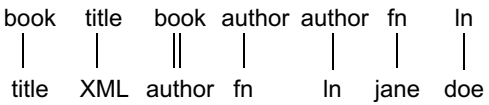

Fig. 4. Decomposição de *twig* padrão em relações binárias.

Apesar de ter proposto dois algoritmos, o primeiro deles, o *tree-merge* é basicamente uma extensão dos algoritmos tradicionais e do MPMGJN. Já o segundo, o *stack-tree* não tinha precedentes nos algoritmos de junção tradicionais, mostrando ser muito mais eficiente que eles.

Considere dois nós n1 e n2 num relacionamento A-D e suas listas posicionais ordenadas por início, Ln1, com as ocorrências dos ascendentes e Ln2 com as ocorrências dos descendentes. O algoritmo *stack-tree* faz um *merge* estrutural entre elas. À medida que avança, é checada a relação A-D entre os elementos, verificando sua existência entre o elemento corrente no topo da pilha e o próximo nó das listas a ser feito o *merge*, ou seja o elemento da lista que tiver o menor valor de início. Com esta comparação a pilha é manipulada apropriadamente e a saída é produzida.

Em qualquer momento do algoritmo é garantido que a pilha tem uma seqüência de nós ancestrais, cada nó da pilha sendo um descendente daquele que está abaixo

dele. Quando um novo nó na lista de ascendentes é encontrado para ser um descendente do topo corrente da pilha, ele é simplesmente colocado na pilha. Quando um nó da lista de descendentes é encontrado para ser um descendente do topo da pilha, sabe-se que ele é descendente de todos os nós da pilha. Também é garantido que ele não será um descendente de qualquer outro nó da lista de ascendentes. Então a junção resulta em envolver este elemento da lista de descendentes com cada um dos nós da lista de ancestrais que estão na pilha. Se o novo nó na lista de *merge* não é um descendente do topo corrente da pilha então é garantido que nenhum nó futuro desta lista será descendente do topo corrente da pilha. Com isso, pode ser executado uma operação de pop na pilha e ser repetido o teste com o novo elemento no topo. Nenhuma saída é gerada com um elemento no qual foi executado o comando pop.

Em seus estudos, Bruno et al. (2002) observaram que os algoritmos anteriores geravam uma quantidade grande de resultados intermediários desnecessários, já que muitos deles não contribuíam para o resultado final. Propôs então um algoritmo de junção estrutural holístico, o *TwigStack*, que avalia a consulta como um todo, numa tentativa de reduzir o número de resultados intermediários.

Este novo algoritmo, que vem a ser uma evolução dos anteriores, usa a mesma notação posicional, com listas invertidas associadas a cada nó. O seu diferencial é: decomposição da *twig* padrão em caminhos completos no sentido raiz-folha e o uso de uma cadeia de pilhas ligadas, uma para cada elemento da *twig* padrão, usadas para representar de maneira compacta os resultados intermediários parciais da consulta.

Este algoritmo consiste de duas fases: produção de uma lista de caminhos solução raiz-folha como resultados intermediários, onde cada saída corresponde a um casamento no documento de dados originais; e junção estrutural, onde todos os caminhos gerados na primeira fase são mesclados de maneira a produzir uma resposta final.

O seu funcionamento básico considera as listas invertidas associadas a cada elemento da consulta *twig*, que são percorridos uma única vez sem volta, durante o processamento do algoritmo. Enquanto as listas associadas aos elementos das folhas não estiverem vazias, ou seja uma solução pode ser encontrada, seleciona, entre todos os elementos, aquele com menor início, colocando-o na sua respectiva pilha. Caso este elemento colocado na pilha seja uma folha, imprime a solução. Uma vez computados os resultados intermediários, executa a sua junção para geração da resposta.

O diferencial deste algoritmo é que antes de um nó ser colocado na pilha, uma função recursiva GetNext garante que este nó tem um descendente n_i em cada uma das listas Ln_i para os n_i que são seus filhos e que cada um dos nós n_i recursivamente satisfazem a mesma condição. Caso as condições não sejam satisfeitas, o algoritmo avança para o próximo elemento das listas com menor valor de início.

Experimentos mostraram que o algoritmo *TwigStack* é ótimo para *twig* padrão cujos relacionamentos são todos do tipo A-D, com aproveitamento total dos resultados intermediários gerados. Entretanto, para *twig* padrão com um ou mais relacionamentos do tipo P-F, o algoritmo não é capaz de controlar a geração de resultados intermediários. Segundo Choi et al. (2003), nenhum algoritmo conseguirá determinar se um nó é útil na solução sem avançar na sua lista invertida. Entretanto, esta ação pode levar à perda de informação que poderia ser considerada num outro caminho, pois no avanço da lista, o que fica para trás não pode ser mais utilizado.

Como tentativa de superar esta restrição, Lu et al. (2004) propuseram um novo algoritmo, o *TwigStackList*, que vem a ser uma variação do *TwigStack*. A principal diferença é usar outra estrutura de dados associada a cada elemento da *twig* padrão, além da pilha e da lista invertida: uma lista para armazenar alguns elementos que encontram-se na lista invertida e já foram avaliados. Com isso, é possível avançar e retroceder nos elementos desta lista, permitindo uma melhor decisão se o caminho em análise é útil ou não para a solução da consulta, mesmo em casos de *twig* padrão com relacionamentos do tipo P-F.

Este algoritmo, igualmente ao seu anterior, mostrou-se ótimo, em termos de E/S, em *twig* padrão apenas com relacionamentos A-D. Mas, diferente de seu anterior, mesmo em consultas contendo relacionamentos P-F, o conjunto de resultados intermediários no *TwigStackList* é garantido ser um subconjunto dos algoritmos anteriormente propostos.

Avaliação dos Algorimos Apresentados

A junção estrutural é uma operação crucial em qualquer processador de consulta XML, sendo normalmente a razão de alto custo. A quantidade de resultados intermediários desnecessários gerados com certeza tem forte impacto na eficiência de um algoritmo que realiza a consulta *twig*.

Sob este aspecto, os dois primeiros algoritmos apresentados na seção anterior, apesar das inovações que trouxeram consigo, geram muitos resultados intermediários desnecessários, tornando-os ineficientes do ponto de vista de E/S.

Já o *TwigStack*, mostrou-se ótimo para consultas com *twig* padrão cujos relacionamentos sejam todos do tipo A-D. Entretanto, para *twig* padrão com um ou mais relacionamentos do tipo P-F, o algoritmo continua não controlando a geração de resultados intermediários. Isto se deve ao fato de que em sua análise, este algoritmo considera elementos descendentes, sem levar em conta informação de seu nível.

A Fig. 5 ilustra um exemplo de uma *twig* padrão que tem um relacionamento P-F para <a,d>. Apesar de não existir este caminho no documento XML, o algoritmo

TwigStack consideraria o nó a, pois existe o relacionamento A-D para <a,d>.Com isso todos os caminhos da árvore seriam gerados como resultados intermediários da primeira fase, mesmo não existindo solução para a consulta.

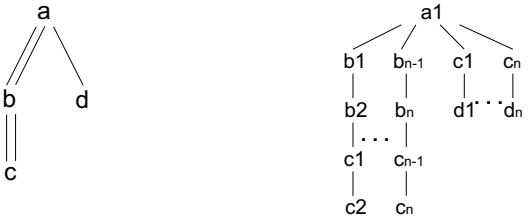

Fig. 5. Consulta *twig* com relacionamento P-F.

Lu et al. (2004) realizaram experimentos com o algoritmo *TwigStack* usando dados XML reais do TreeBank da Universidade de Washington. Este documento tem profundidade máxima de 36, vários elementos com recursão e mais de 2.5 milhões de nós. A Tabela 1 mostra os números obtidos com este experimento, na realização de consultas cuja *twig* padrão apresentava um ou mais relacionamentos do tipo P-F.

Tabela 1. Resultados de consultas em ***T**wigStack* com relacionamento P-F.

Consulta	*Caminhos parciais*	*Caminhos usados na resposta final*	*% de caminho inúteis*
VP[/DT]//PRP_DOLLAR	10663	5	99,9
S[/JJ]/NP	70988	10	99,9
S[//VP/IN]NP	702391	22565	96,8

Por estes dados, percebe-se que em casos de relacionamentos P-F o algoritmo *TwigStack* realmente gera muitos resultados intermediários.

O algoritmo *TwigStackList* proposto por Lu et al. (2004) implementa uma solução para o problema do *TwigStack*, baseando-se numa sugestão de Choi et al. (2003), que dá uma indicação de uso de memória para guardar informações que podem ser descartadas erroneamente. Para isso, Lu faz uso de um estrutura de lista, armazenando informações adicionais dos elementos percorridos. A Tabela 2 apresenta os resultados obtidos com a execução deste algoritmo nos dados XML do TreeBank, comparando-os à execução do *TwigStack*.

Tabela 2. Resultados de consultas em *TwigStack* e *TwigStackList*.

Consulta	*Caminhos parciais TwigStack*	*Caminhos parciais TwigStackList*	*Caminhos usados na resposta final*	*% de redução*
S[//MD]//ADJ	35	35	35	0
VP[/DT]//PRP_DOLLAR	10663	11	5	99,9
S[//VP/IN]NP	702391	22565	22565	96,8
S[/JJ]/NP	70988	30	10	99,9

Tais resultados mostram uma grande diferença de eficiência deste algoritmo em relação ao *TwigStack*, o que pode representar mais um grande marco na evolução das consultas *twig*.

Novos Desafios

Apesar da evolução dos algoritmos que tornaram a consulta *twig* mais eficiente, ainda existem desafios a serem vencidos, visando melhorar ainda mais a eficiência das consultas *twig* em dados XML.

Um ponto a ser estudado é a otimização das consultas. A junção é a operação mais cara na avaliação de consultas. Por isso, selecionar adequadamente a junção a ser realizada é uma tarefa chave para o otimizador, mesmo em consultas XML.

A junção em consultas XML geralmente foca nos relacionamentos A-D e P-F dos elementos relacionados, sendo assim, a condição de junção especificada em termos da sua posição relativa no documento XML.

Usando modelos simples de custo para algoritmos de junção estrutural, Wu et al. (2003) desenvolveram cinco algoritmos de otimização para junção estrutural. Estes algoritmos tentam eliminar planos de consultas desnecessárias para a geração da resposta final a um custo menor. Para isso usa heurísticas, como limite de número de resultados intermediários a serem considerados, e considera escolhas normalmente feitas por otimizadores de sistemas relacionais. Este trabalho pode servir de base para a expansão de estudos focando em algoritmos mais novos.

Um outro ponto de estudo é a melhoria nos índices empregados nas consultas *twig*. Yang et al. (2004) propõem o uso combinado de índices de caminho (path index) e de informação de ancestrais como forma de substituir o cursor físico usado na junção estrutural. Com o uso de cursores virtuais, torna-se possível eliminar operações de E/S sobre os índices, o que pode melhorar a performance das junções estruturais na ordem de uma magnitude ou mais. Ainda apresenta modificações para acomodar o uso de cursores virtuais no algoritmo *TwigStack*. Entretanto, todos os exemplos utilizados consideram apenas relacionamentos do tipo A-D. Uma pesquisa interessante seria o uso destes cursores no algoritmo *TwigStackList* já discutido anteriormente.

Já Li et al. (2004) fazem um estudo comparativo com as três soluções de índices mais usadas em junções estruturais: árvore B^+, árvore XB e árvore XR, apresentando e comentando os resultados obtidos. Estes resultados também podem ser utilizados para pesquisas futuras de maneira a contribuir na evolução dos algoritmos existentes.

Por fim, Jiang et al. (2004) colocam que os algoritmos *twig* holísticos provaram ser um grande avanço em consultas *twig*. Entretanto, estes algoritmos normalmente tratam de consultas com relacionamento entre irmão do tipo AND e não do tipo OR. Como é natural o uso do OR em consultas, Jiang et al. (2004) sugerem um tratamento para estes relacionamentos diferente do mecanismo normalmente utilizado, que traduz as consultas OR para a forma normal conjuntiva, combinando depois os resultados obtidos. Esta decomposição tem a desvantagem de fazer vários acessos ao mesmo elemento, podendo fazer a consulta crescer de maneira exponencial. Como solução, os autores propõem um novo *framework* para tratar consultas OR baseando-se no conceito de OR-Block. Consiste, com certeza, em mais um ponto para estudos futuros.

Considerações Finais

Encontrar todas as ocorrências de uma *twig* padrão em um documento XML não é uma operação trivial, constituindo no ponto central de processamento de consultas XML. O algoritmo usado nesta operação desempenha papel importante, sendo capaz de promover ganhos em eficiência ou de gerar uma quantidade enorme de resultados intermediários desnecessários.

Este trabalho apresentou os principais algoritmos já desenvolvidos nesta área, comentando suas contribuições e evoluções. Também foram apresentados novos desafios que vem sendo propostos, provendo ao leitor uma visão geral do estado da arte desta área de pesquisa.

Conclusões

1. A linguagem XML é um padrão estabelecido para a troca de dados em aplicações para a *web*. Por isso, a existência de mecanismos de consulta eficientes a dados em XML é essencial.
2. Consultas *twig* vem mostrando ser um ponto central do processamento de consultas em XML, objeto de várias linhas de estudo.
3. As principais contribuições obtidas até o momento nesta área de pesquisa são: sistema de codificação posicional baseada em sistema de índice invertido, algoritmo para junção estrutural e consultas holísticas.
4. Ainda há geração de muitos resultados intermediários, o que motiva a contínua busca por algoritmos mais eficientes, senão ótimos.
5. Além disso, novos desafios vêm sendo colocados como a otimização de consulta, melhoria no uso de índices e tratamento de consulta do tipo OR, provando que esta é uma área de pesquisa em constante evolução, com novos pontos de estudos e obstáculos a vencer.

Referências Bibliográficas

AL-KHALIFA, S.; JAGADISH, H. V.; KOUDAS, N.; PATEL, J. M.; SRIVASTAVA, D.; WU, Y. **Structural joins:** a primitive for efficient XML query pattern matching. 2002. Disponível em: <http://www.eecs.umich.edu/~jignesh/publ/xmljoin-ICDE.pdf>. Acesso em: jun. 2005.

BRUNO, N.; KOUDAS, N.; SRIVASTAVA, D. Holistic *twig* joins: optimal XML pattern matching. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2002, Madison. **SIGMOD 2002**: proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002. p. 310-321.

CHOI, B.; MAHOUI, M.; WOOD.; D. On the optimality of holistic algorithms for *twig* queries. In: MARÍK, V.; RETSCHITZEGGER, W.; STEPÁNKOVÁ, O. (Ed.). **Database and expert systems applications:** proceedings of the 14th International Conference: DEXA 2003. [Prague]: Springer-Verlag, 2003. p. 28-37. (Lecture Notes in Computer Science, v. 2736).

JIANG, H.; LU, H.; WANG, H. Efficient processing of XML *twig* queries with or-predicates. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2004, Paris. **SIGMOD 2004**: proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2004.

LI, H.; LEE, M. L.; HSU, W.; CHEN, C. An evaluation of XML indexes for structural join. **ACM SIGMOD Record**, v. 33, n. 3, p. 28-33, Sept. 2004. Disponível em: <http://www.sigmod.org/record/issues/0409/5.indexes2.pdf>. Acesso em: jun. 2005.

LU, J.; CHEN, T.; LING, T. W. Efficient processing of XML *twig* patterns with parent child edges: a look-ahead approach. In: ACM CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 13., 2004, Washington, D.C. **Proceedings of the thirteenth...** New York: ACM Press, 2004. p. 533-542. Disponível em: <http://portal.acm.org/ft_gateway.cfm?id=1031272&type=pdf&coll=GUIDE&dl=GUIDE&CFID=56419879&CFTOKEN=47234855>. Acesso em: jun. 2005.

WORLD WIDE WEB CONSORTIUM. **Query 1.0:** an XML query language. W3C working draft 04 Apr. 2005. Disponível em: <http://www.w3.org/TR/2005/WD-xquery-20050404/>. Acesso em: jun. 2005.

WU, Y.; PATEL, J.M.; JAGADISH, H. V. Structural join order selection for XML query optimization. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE'03), 19., 2003, Bangalore. **Proceedings ...** Washington, D. C.: IEEE Computer Society, 2003. p. 443-454. Disponível em: <http://www.eecs.umich.edu/~jignesh/publ/XML-opt.pdf>. Acesso em: jun. 2005.

YANG, B.; FONTOURA, M.; SHEKITA, E.; RAJAGOPALAN, E.; BEYER, K. Virtual cursors for XML joins. In: ACM CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 13., 2004, Washington, D.C. **Proceedings of the thirteenth...** New York: ACM Press, 2004. p. 523-532. Disponível em: <http://portal.acm.org/ft_gateway.cfm?id=1031271&type=pdf&coll=GUIDE&dl=GUIDE&CFID=56419879&CFTOKEN=47234855>. Acesso em: jun. 2005.

ZHANG, C.; NAUGHTON, J.; DEWITT, D.; LUO, Q.; LOHMAN, G. On supporting containment queries in relational database management systems. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2001, Santa Barbara. **SIGMOD 2001:** proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2001. p. 425-436.

Embrapa
Informática Agropecuária

Ministério da Agricultura,
Pecuária e Abastecimento

BRASIL
UM PAÍS DE TODOS
GOVERNO FEDERAL