Processamento Eficiente de Junções Espaçotemporais

Geraldo Zimbrão
Jano Moreira de Souza
Victor Teixeira de Almeida

Computer Science Department, Graduate School of Engineering
Federal University of Rio de Janeiro
PO Box 68511, ZIP code: 21945-970 Rio de Janeiro - Brazil,
Email: {zimbrao,jano,valmeida}@cos.ufrj.br

Resumo

É um fato conhecido que as novas aplicações de Sistemas de Informações Geográficas (SIG) necessitam armazenar informações temporais. Entre outras operações, um SGBD Espaçotemporal deve eficientemente suportar a consulta de junção espaçotemporal. A melhor estrutura de indexação de dados espaciais conhecida, a R-Tree (e suas variantes) não armazena a evolução dos MBRs. Novas estruturas de indexação têm sido propostas na literatura que permitem o armazenamento e recuperação de estados presentes e passados dos dados e, em sua maioria, são baseadas na estrutura da R-Tree. Este artigo apresenta um estudo pioneiro no processamento de junções espaçotemporais utilizando algumas destas estruturas, particularmente a R-Tree parcialmente persistente chamada TR-Tree e a 2+3D R-Tree. Partindo de algoritmos de junção espacial, serão apresentados algoritmos para o processamento de junções espaçotemporais para instante e intervalo de tempo em ambas as estruturas. Foram propostas algumas melhorias nos algoritmos e tentamos mostrar a corretude dos mesmos. Finalmente, implementamos e testamos os novos algoritmos com alguns conjuntos de dados espaçotemporais. Nossos experimentos mostraram que a performance dos algoritmos é boa até mesmo em casos extremos, como conjuntos de dados com muitas versões, mostrando sua boa escalabilidade, especialmente para a TR-Tree. Além disto, com algumas adaptações, as idéias principais dos nossos algoritmos podem ser utilizadas para a avaliação de junções utilizando-se outras estruturas de dados parcialmente persistentes, como a MVB-Tree.

Abstract

It's a well-known fact that the new GIS applications need to keep track of temporal information. Among other operations, a spatiotemporal DBMS should efficiently answer the spatiotemporal join. The best-known spatial index structure, the R-Tree (and its variants), does not preserve the MBRs' evolution. New indexing structures were proposed in the literature that allows the retrieving of present and past states of data, and most of them are R-Tree based. This paper presents a first study of spatiotemporal join processing using these new structures, particularly a partially persistent R-Tree called Temporal R-Tree and the 2+3D R-Tree. Starting from spatial join algorithms, we present algorithms for processing spatiotemporal joins over time instants and intervals on both spatiotemporal structures. Then, we propose some improvements that lead to a better performance and try to show the correctness of our algorithms. Finally, we implement and test these new algorithms with some spatiotemporal data sets. Our experiments shows that our algorithms performance is good even on extreme cases, like datasets with many changes, showing its good scalability – especially for the TR-Tree. In addition, with minor adaptations, the main ideas of our algorithm can be used for evaluating joins using other partially persistent structures, like the MVB-Tree.

1. Introdução

Uma das principais funções de um Sistema Gerenciador de Banco de Dados Espaçotemporal (SGBDET) é o armazenamento e a recuperação eficiente de objetos espaçotemporais, ou seja, objetos cuja extensão espacial evolui com o tempo. Segundo [TSPM98], existem dois tipos principais de consultas espaçotemporais: seleções e junções. Este trabalho apresenta um estudo pioneiro no processamento de junções espaçotemporais usando estruturas de dados parcialmente persistentes baseadas em R-Trees [G84], como a Temporal R-Tree [ZAS00] e a Partially Persistent R-Tree [K+01]. Um algoritmo refinado para a realização de junções espaçotemporais específico para estruturas de dados parcialmente persistentes é apresentado. Sua acurácia é verificada e o seu desempenho é testado realizando-se várias consultas em diversos conjuntos de dados. Além disso, as idéias principais apresentadas neste trabalho podem ser utilizadas para desenvolver algoritmos de junção em estruturas de dados parcialmente persistentes, tanto sobre dados espaçotemporais como dados apenas temporais.

Conforme é apontado em [TSPM98], SGBDs Espaçotemporais devem oferecer tipos de dados apropriados e linguagens de consultas que suportem dados espaciais que evoluem com o tempo; devem prover métodos de indexação e recuperação eficientes; e numa etapa posterior, devem também explorar modelos de custos das operações espaçotemporais para fins de otimização. Relevantes aplicações que um SGBD espaçotemporal deve suportar incluem Sistemas de Informações Geográficas e Temporais (SIGTs), Sistemas de Multimídia, Bancos de Dados Estatísticos e Científicos, tais como bancos de dados para a área médica, meio ambiente, previsão de tempo, evolução do clima etc.

Um SGBD Espaçotemporal é um sistema que mantém registro dos estados passados dos dados espaciais. Em tais sistemas, cada alteração nos dados cria uma nova versão dos mesmos. O sistema nunca apaga um objeto: apenas marca o seu tempo de remoção. Desta forma, a quantidade de dados é sempre crescente. A necessidade de se indexar bases de dados espaçotemporais é óbvia: quanto maior o volume de dados, maiores são os ganhos de desempenho obtidos ao se usar esquemas sofisticados de indexação. Contudo, isto irá demandar novos algoritmos e estruturas para se indexar simultaneamente dados espaciais e temporais.

Uma das operações fundamentais e mais caras em bancos de dados é a operação de junção. Uma junção espaçotemporal é uma operação usada para combinar objetos espaçotemporais de dois conjuntos que possuem uma determinada propriedade. De grande interesse prático é a junção de interseção espaçotemporal, que significa: “encontrar todos os pares de objetos que em algum momento de sua existência possuíram uma área comum”. Além disso, a consulta pode ser restringida a uma região específica e/ou um determinado intervalo de tempo.

Este trabalho está organizado da seguinte forma. A Seção 2 define o problema e os termos usados. A Seção 3 apresenta um resumo das novas estruturas de dados espaçotemporais e seus algoritmos básicos. A Seção 4 discute a junção espaçotemporal e apresenta os nossos algoritmos. A Seção 5 contém a descrição dos dados e das consultas, bem como os resultados experimentais. Finalmente, a Seção 6 apresenta as conclusões deste trabalho e as futuras direções do mesmo.

2. Definindo o Problema

Um Método de Acesso Espaçotemporal (MAET) é um índice que organiza dados espaçotemporais tanto por uma chave espacial (MBR – *Minimum Bounding Rectangle*) como temporal (intervalo de tempo de vida) dos objetos. O principal objetivo de um MAET é ser um suporte eficiente ao processamento de consultas espaçotemporais. Quanto mais abrangente o conjunto de consultas suportado, mais importante e útil torna-se o método de acesso. Segundo [TSPM98], um conjunto fundamental de consultas espaçotemporais é:

- *Consultas de Seleção*: são consultas da forma "encontrar todos os objetos que possuem ou possuíram interseção com uma determinada área (ou com um determinado ponto), durante um determinado intervalo de tempo (ou em um determinado instante de tempo)".
- *Consultas de Junção*: são consultas da forma "encontrar todos os pares de objetos que possuem interseção entre si durante um determinado intervalo de tempo (ou em um determinado instante de tempo)".

No contexto de bancos de dados espaciais, as consultas de seleção são também conhecidas como consultas por janela, e as consultas de junção são, por sua vez, conhecidas como junções espaciais [BKS93]. Neste artigo iremos adotar uma convenção para nomenclatura análoga. Por motivos de desempenho, iremos definir ainda um operador de junção mais poderoso combinando a operação primitiva de junção com uma janela de seleção. Desta forma, nossa definição de junção será: "encontrar todos os pares de objetos que possuem interseção entre si e com uma determinada área (ou com um determinado ponto) durante um determinado intervalo de tempo (ou em um determinado instante de tempo)". Chamaremos este novo operador de junção espaçotemporal por janela. Acreditamos que este tipo de consulta é o mais utilizado por usuários de um SGBDET. A Tabela 1 mostra os tipos de consultas que um MAET deve suportar.

	Consulta por janela	**Junção espacial por janela**
Instante de tempo	encontrar todos os objetos que possuem interseção com uma determinada janela em um determinado tempo.	encontrar todos os pares de objetos que possuem interseção entre si em um determinado tempo.
Intervalo de tempo	encontrar todos os objetos que possuem interseção com uma determinada janela durante um intervalo de tempo.	encontrar todos os pares de objetos que possuem interseção entre si durante um intervalo de tempo.

Tabela 1 – Tipos de consultas que um MAET deve suportar.

Em Brinkhof [BKS93], temos uma forma de realizar a junção entre dois conjuntos de dados para os quais tenhamos uma R-Tree. Nossa proposta consiste em adaptar aquele algoritmo para lidar com estruturas temporais parcialmente persistentes – e neste caso, a adaptação não é apenas lidar com mais uma dimensão, o tempo, mas levar em conta também o fato de que as estruturas parcialmente persistentes possuem duplicação de dados.

3. Trabalhos Relacionados

Nesta seção iremos resumir alguns trabalhos correlatos. Embora existam alguns MAETs propostos na literatura, não há nenhum trabalho que mencione a existência de um método de processamento de junção espaçotemporal utilizando-se de alguns destes MAETs.

3.1 Métodos de Acesso Temporais

Nos últimos anos, uma série de métodos de acesso temporal foi proposta na literatura. Uma revisão bastante abrangente da literatura pode ser encontrada em [ST94]. A maioria das abordagens propostas assume que as alterações ocorrem ao longo do tempo em ordem crescente, logo, são apropriadas para o armazenamento da evolução do tempo de transação. Estas abordagens não removem fisicamente os objetos, mas apenas os marcam como "mortos", ficando os objetos ainda não removidos marcados como "vivos". Alguns destes métodos de acesso temporais podem ser adaptados ao domínio espacial dos dados.

A Multiversion B-Tree (MVB-Tree) apresentada em [BGO+96] é uma boa abordagem para o tratamento de dados temporais, mas não espaciais. A principal idéia deste trabalho é a de manter os dados removidos nos nós da estrutura e executar o que é chamado de divisão de versão (*version split*) sempre que um nó torna-se cheio. Nesta operação apenas os dados vivos são copiados para um novo nó evitando assim a replicação de dados antigos. Nosso trabalho estende as idéias da MVB-Tree para o contexto de dados espaçotemporais utilizando R-Trees, e tentamos manter suas convenções de nomes. Embora muitas das suposições de desempenho para B-Trees não sirvam para as R-Trees, nossos experimentos mostraram bons resultados em termos de utilização de espaço, operações de E/S e tempo de resposta para consultas.

A Bitemporal R-Tree proposta em [KTF95, KTF97], no contexto de bancos de dados bitemporais, é uma implementação de uma R-Tree parcialmente persistente. A Bitemporal R-Tree é utilizada para indexar dados não espaciais, seus tempos de validade e de transação. O tempo de transação é mantido utilizando-se uma R-Tree parcialmente persistente para indexar os valores das chaves não espaciais e do tempo de validade, um em cada eixo em duas dimensões. Algumas decisões de projeto da Bitemporal R-Tree impedem que a mesma seja diretamente convertida para suportar dados espaçotemporais, alguns ajustes devem ser feitos.

3.2 Métodos de Acesso Espaçotemporais

Segundo [TSPM98], na época de sua publicação, apenas quatro métodos de acesso espaçotemporais apareciam na literatura: 3D R-trees [TVS96], MR-trees e RT-trees [XHL90], e HR-trees [NS98]. Estes métodos têm as seguintes características:

- 3D R-trees tratam o tempo como uma nova dimensão utilizando R-Trees [G84, BKS+90] como método de acesso espacial.
- RT-trees acoplam o intervalo de tempo à informação espacial dentro dos nós da árvore adotando idéias das R-trees e TSB-trees [LS89].
- MR-trees e HR-trees usam a técnica de sobreposição de árvores com R-trees para representar os estados sucessivos da base de dados [MK90], a última utilizando-se de Hilbert R-Trees [KF94].

A esta lista, devemos adicionar quatro novos métodos de acesso: 2+3D R-Trees, R^{ST}-Trees [SJ99], PPR-Trees [K+01] e Temporal R-Trees [ZAS00] que será discutida em maiores detalhes na seção 3.3.

A 2+3D R-Tree é uma melhoria significativa implementada na 3D R-Tree para resolver o problema do tempo atual que gera MBRs crescendo indeterminadamente. A 2+3D R-Tree utiliza uma 2D R-Tree para objetos vivos e uma 3D R-Tree para objetos com tempo de morte determinado. Com isso evita-se que o tempo de morte indeterminado dos objetos faça com que os retângulos cresçam, uma vez que esta informação temporal não faz parte dos MBRs da 2D R-Tree. Esta estrutura cria um novo problema que é o de que as buscas devem percorrer duas ao invés de apenas uma árvore, como nas 3D R-Trees.

A R^{ST}-Tree proposta em [SJ99], tenta resolver o problema do tempo atual crescente no contexto de bancos de dados espaciais e bitemporais. O problema é resolvido deixando que os MBRs dos objetos também cresçam ao longo do tempo formando uma escada e possuindo aspecto parecido com um triângulo, o que reduz o espaço de busca de forma considerável. Antes deste trabalho, as estruturas só eram capazes de lidar com o tempo "agora" para tempos de transação, o que pôde ser estendido também para o tempo de validade.

A TR-Tree é uma especialização da MVB-Tree para o tratamento de dados espaciais, ou seja, altera a estrutura e os algoritmos da MVB-Tree para que a mesma possa utilizar uma estrutura de dados espacial, a R-Tree [G84], e ainda estende seus algoritmos para dar suporte aos algoritmos da R*-Tree. Conseqüentemente, a TR-Tree é uma versão espacial da MVB-Tree capaz então de armazenar e consultar eficientemente dados espaçotemporais. Uma descrição completa dos algoritmos bem como uma avaliação de desempenho pode ser encontrada em [ZAS00].

A PPR-Tree proposta em [K+01], é bastante semelhante à TR-Tree. É uma tentativa de se estender as técnicas propostas em [BGO+96] para a MVB-Tree para dados espaciais utilizando R-Trees. Este trabalho é bastante semelhante a Bitemporal R-Tree [KTF95, KTF97], apenas trocando a dimensão do tempo de validade por mais uma dimensão dos dados tornando-os assim espaciais. Não foi proposto, em [K+01], uma forma de se efetuar a reinserção de entradas devido à dificuldade de implementação de tais métodos e, assim, quando de uma remoção um nó viola a restrição de número mínimo de entradas o mesmo é juntado com algum nó vizinho segundo cinco tipos de heurísticas. Também não é implementado em [K+01] o mecanismo de reinserção forçada de nós da R*-Tree, o que é mostrado em [BKS+90] que melhora substancialmente o desempenho de tais estruturas. A TR-Tree, embora tenha surgido antes da PPR-Tree, implementa tanto a reinserção forçada como a divisão (*split*) por eixo, e outras otimizações características da R*-Tree.

Existe também uma gama de propostas para a indexação de objetos em movimento. Não iremos mencioná-las aqui uma vez que estão fora do escopo do nosso trabalho. Estamos assumindo aqui que as geometrias dos objetos não mudam de forma contínua, mas apenas em passos discretos, e portanto não estamos mencionando objetos em movimento. Segundo [TSPM98], este caso é de grande interesso para bancos de dados espaçotemporais.

3.3 Temporal R-Tree

Apresentaremos aqui um breve resumo da estrutura da TR-Tree. Para maiores detalhes, o leitor deve consultar [ZAS00] ou [ZAS99]. A estrutura de índice da TR-Tree é bastante similar à da R-Tree com algumas alterações para que possa tratar de informações temporais. Cada entrada na TR-Tree possui um MBR, um identificador e um intervalo de tempo na forma [t_nascimento,t_morte). O tempo agora (*now*) é representado por "*". As entradas na TR-Tree ficam então na forma <MBR, ponteiro, t_nascimento, t_morte>. Ainda inserimos no nó a informação de seu tempo de nascimento para as que as alterações em bloco, ou seja, um conjunto de alterações em uma mesma versão, pudessem se tornar possíveis.

Toda vez que um novo MBR de um objeto é inserido na base de dados no tempo ti seu intervalo de tempo de vida torna-se [t_i, *). Um MBR permanece vivo até ser removido ou alterado. Uma alteração de um objeto no tempo t_i é implementada como uma remoção lógica (mudança do atributo t_morte de "*" para t_i) seguida de uma inserção do objeto com suas modificações. Um objeto é dito vivo no tempo t_i se t_nascimento $\leq t_i <$ t_morte, ou seja, $t_i \in$ [t_nascimento, t_morte). Desta forma, entradas com tempos de diferentes versões na base de dados podem coexistir em um mesmo nó.

Outra importante modificação na TR-Tree é que pode conter mais de um nó raiz apontando para R-Trees, isto é, existem mais de uma árvore na estrutura da TR-Tree. Entretanto, em um determinado instante de tempo, existe uma e apenas uma árvore correspondente. Como conseqüência deste fato, a TR-Tree não deve ser vista como uma árvore, mas como um grafo acíclico direcionado. Ainda, deve haver uma estrutura em forma de vetor para a indexação dos nós raízes da TR-Tree e seus respectivos intervalos de tempo de vida. Chamaremos esta estrutura de vetor de nós raízes.

Uma mudança estrutural para tratar um *overflow* de nó ou *underflow* é efetuada baseada na operação de cópia de nós, isto é, o nó é marcado como morto e suas entradas vivas na versão corrente são copiadas para um novo nó. Os algoritmos da TR-Tree foram implementados de forma a permitir um bloco de operações antes que uma versão seja criada. Desta forma a criação de novas versões da estrutura deve ser explicitamente executada. Esta é uma melhoria apropriada para vários casos como alterações em bloco, passos intermediários produzidos pela aplicação de restrição de integridades referenciais, e outras operações quaisquer que produzam dados inválidos que o sistema não precise manter armazenados, como reinserções de entradas em nós removidos, por exemplo. Finalmente, podemos dizer que as principais idéias propostas na TR-Tree [ZAS00] para tornar as R-Trees parcialmente persistentes foram as mesmas utilizadas na Bitemporal R-Tree [KTF95, KTF97] e posteriormente na PPR-Tree[K+01], provenientes das idéias propostas em [BGO+96] para a MVB-Tree.

3.4 Comparações entre os MAETs

As principais diferenças que podemos ressaltar entre a TR-Tree e a PPR-Tree são no tratamento de underflow de nós e na implementação dos algoritmos da R*-Tree. Os autores da PPR-Tree, os mesmos da Bitemporal R-Tree, não propuseram uma solução para a reinserção de entradas em nós com menos do que o número mínimo *m* de entradas. Sabe-se que a idéia da reinserção de entradas produz estruturas mais bem organizadas que as provenientes da fusão (*merge*) de nós devido à falta de uma ordenação total do espaço em mais de uma dimensão. A escolha do nó vizinho a ser juntado com o nó com menos de *m* entradas é bastante difícil e nem sempre ótima. São propostos cinco tentativas de se encontrar o melhor vizinho a se inserir as entradas do nó em underflow em [KTF95, KTF97]. Ainda sobre essa questão de reinserção de nós, os autores da PPR-Tree não propuseram também uma solução para a reinserção forçada proposta nos algoritmos de inserção da R*-Tree o que garante uma melhor reorganização da árvore de tempos em tempos. Espera-se, devido à estes motivos, que a estrutura da TR-Tree encontre-se mais bem organizada que a da PPR-Tree e com isso, os algoritmos de busca e junção possam ter um melhor desempenho.

Finalmente, podemos dividir as estruturas espaçotemporais em duas classes quanto a uma particularidade extremamente relevante para a realização de junções (e de seleções também): se a estrutura em questão duplica ou não entradas. Das estruturas até aqui mencionadas, podemos dizer que a 3D R-Tree, a 2+3D R-Tree e a RT-Tree não duplicam entradas, ao passo que a TR-Tree, a HR-Tree e a MR-Tree duplicam.

Na HR-Tree e na MR-Tree, a duplicação de entradas ocorre sempre que algum nó deve ser alterado: o nó inteiro é copiado e alterado, e uma nova entrada é criada em seu pai, repetindo-se o processo até a raiz. Com isto, a cada inserção ou remoção todo o caminho da raiz até a folha onde ocorreu a alteração é duplicado - decorre deste fato a pouca adequação destas árvores para lidar com conjuntos de dados onde a taxa de alterações é de média para muito freqüentes.

Já a TR-Tree duplica nós/entradas apenas quando ocorre uma mudança estrutural, ou seja, um overflow ou underflow, resultando em muito menos duplicação de dados. Apenas para ilustrar este fato, imagine uma HR-Tree com altura igual a três, e ocupação de cada nó entre 40 e 100 entradas. Suponha que queiramos realizar 100 inserções, cada uma com tempo diferente das outras. Cada uma destas inserções irá resultar em três novos nós a serem inseridos na árvore, resultando em trezentos novos blocos. Já no caso da TR-Tree, ao realizarmos a mesma operação, podemos esperar que algumas inserções causem overflow, porém é remota a possibilidade de que todas as inserções resultem em overflow e conseqüentemente em duplicação de entradas: após cada overflow, que é resolvido com a divisão de um nó em dois, iremos ter novos nós abaixo de sua capacidade máxima. Além disso, o mesmo processo irá se repetir recursivamente até a raiz da árvore, ocasionando alguns overflows nos nós intermediários, mas certamente não em todos os que forem alterados. Apenas como exemplo, considere o caso em que a árvore está com a ocupação dos blocos tão próxima da capacidade máxima que a cada dez inserções tenhamos um overflow. Neste caso, ao inserirmos cem novas entradas, teremos cerca de vinte novos blocos no nível mais baixo da árvore, e ao realizarmos a inserção nos níveis intermediários destes vinte novos blocos, teremos mais dois novos blocos, resultando em algo em torno de vinte e dois novos blocos - um número significativamente menor do que o obtido na HR-Tree.

4. A Junção Espaçotemporal

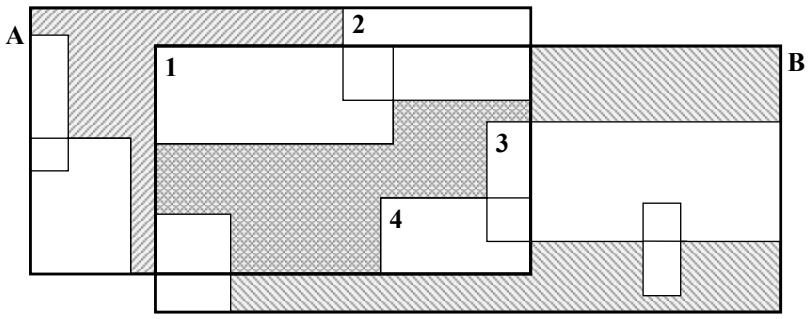
Nesta Seção nós iremos apresentar os algoritmos para a realização de junções espaçotemporais usando-se como estruturas de índices a 2+3D R-tree e a TR-Tree. Iremos assumir a pré-existência de índices nos dois conjuntos de dados a serem utilizados na junção. Escolhemos estas duas estruturas pelos motivos que se seguem:

- Em [ZAS00], torna-se evidente que a TR-Tree é a estrutura que se apresenta mais promissora para ser utilizada como índice espaçotemporal genérico, capaz de realizar bem tanto as consultas de instante como de intervalo de tempo, pois na média, em um apanhado de consultas, foi a que apresentou o melhor desempenho.
- Apenas a HR-Tree e a 2+3D R-Tree conseguiram superar o desempenho da TR-Tree nas consultas de instante e de intervalo de tempo, respectivamente. As outras estruturas avaliadas não apresentaram um desempenho satisfatório.
- a HR-Tree não é apropriada para indexar dados que possuem uma taxa de alterações média para alta, pois cria muita duplicação de dados resultando em índices realmente grandes. De fato, seu desempenho em consultas envolvendo intervalos degrada rapidamente a medida que o intervalo de tempo aumenta pelo fato das consultas percorrerem um número excessivo de R-Trees.
- Finalmente, a PPR-Tree, apesar de estruturalmente ser muito semelhante a TR-Tree, é baseada na R-Tree, ao passo que a TR-Tree é na verdade baseada na R*-Tree, que por sua vez comprovadamente apresenta um desempenho melhor que a R-Tree simples.

4.1 A Realização da Junção Espacial sobre os MBRs através de R-Trees

Sendo a R-Tree uma estrutura recursiva, o algoritmo que a utiliza para realizar a junção espacial também é recursivo. Assim, dadas duas árvores representadas pelos retângulos associados aos respectivos nós de mais alto nível (o nó raiz), iremos investigar os filhos de cada um desses nós, aos pares como em um produto cartesiano. Realizaremos junções nas sub-árvores representadas pelos pares de nós filhos cujos retângulos possuam interseção. Em

cada junção de sub-árvore iremos repetir o processo recursivamente até chegarmos às folhas, quando então iremos obter os pares de MBRs que formam o conjunto resposta da junção de MBRs. A Figura 1 ilustra esse processo. Note que em momento algum do processo é relevante a informação sobre a dimensão dos retângulos: este mesmo algoritmo funciona para qualquer número de dimensões, assim como a R-Tree em si. Em [BKS93] temos um estudo detalhado sobre a realização de junções espaciais utilizando-se R*-Trees. Neste trabalho, realizamos as otimizações apontadas em [BKS93] até o algoritmo SJ3, que é a junção utilizando um algoritmo de *plane-sweep* para comparar um nó com o outro. Os algoritmos seguintes obtém ganhos pouco significativos, pois lidam basicamente com otimizações que afetam o desempenho do cache de disco, pois envolvem somente trocar a ordem de comparações de nós. Decidimos não implementar estes melhoramentos pois o cache LRU utilizado apresentou resultados satisfatórios, de modo que estas otimizações resultariam em ganhos pouco significativos nas duas árvores.



A junção dos blocos **A** e **B** será efetivada mediante a realização de duas junções nas sub-árvores representadas pelos blocos **1** e **2** e pelos blocos **3** e **4**. Os demais blocos não serão avaliados pois os respectivos retângulos não se interceptam.

Figura 1 - Junção através de uma R-Tree

Uma das vantagens deste algoritmo é que cada par de MBRs somente será listado uma vez no conjunto de candidatos, pois cada MBR só pode ser inserido em um nó. A grande desvantagem do algoritmo é resultante da sobreposição dos retângulos correspondentes aos nós da R-Tree. Para investigarmos os MBRs que estão em determinada área onde haja sobreposição de dois ou mais nós teremos de percorrer mais de uma sub-árvore. Quando formos realizar uma junção, o problema irá se agravar, e de forma quadrática: se m retângulos de uma árvore possuem interseção com um ponto i e se n retângulos da outra árvore também possuem interseção com esse mesmo ponto i, então será necessário realizar $m \times n$ junções nas sub-árvores.

4.2 A Junção Espaçotemporal na 2+3D R-Tree

Em sua essência, a estrutura de cada uma das duas árvores que compõem a 2+D R-Tree é idêntica à R-Tree original: apenas uma delas possui duas dimensões (a que armazena os objetos ainda vivos no tempo corrente) e a outra possui três dimensões. Realizar uma junção sobre duas 2+3D R-Trees é portanto semelhante a realizar quatro junções em R-Trees: uma junção sobre um par de árvores com duas dimensões, outra sobre um par de árvores com três dimensões, e finalmente duas junções sobre árvores com duas e três dimensões. Apenas o último caso possui uma particularidade que o torna diferente do algoritmo original: realizar a junção sobre árvores com número de dimensões diferentes. Porém, cabe lembrar ao leitor que o número de dimensões é diferente apenas no que toca aos algoritmos estruturais da árvore, ou seja, de inserção e divisão de nós. Implicitamente temos, na árvore de duas dimensões, a informação da terceira dimensão armazenada também. Esta dimensão corresponde exatamente ao tempo de validade: as entradas nesta árvore possuem um campo com o seu

tempo de criação, e sabe-se que as mesmas estão todas vivas. Logo, a dimensão que faltava será sempre representada por [*a*,*), onde *a* é o tempo de criação da entrada (no caso dos nós, o tempo de criação da entrada mais antiga contida neste nó). Assim, estamos equiparados ao caso de realizar junções em árvores de MBRs de três dimensões. Cabe observar ainda que a estrutura da 2+3D R-Tree não duplica entradas, de forma que, embora tenhamos de realizar quatro junções completas, não é necessário eliminar pares de resposta duplicados.

4.3 A Junção Espaçotemporal na TR-Tree

Conforme descrito anteriormente, a TR-Tree é uma estrutura que duplica entradas, ou seja, um mesmo nó ou objeto espacial pode possuir mais de uma entrada na árvore. Isto irá nos remeter ao seguinte problema: se realizarmos a junção espaçotemporal de forma análoga à junção espacial em R*-Trees, iremos reportar pares de entradas duplicadas. O problema é um pouco mais grave pois devemos lembrar que uma entrada duplicada para um nó irá significar que este nó será visitado duas vezes, o que implica em todas as suas entradas serem reportadas duas vezes. A figura 2 ilustra este problema.

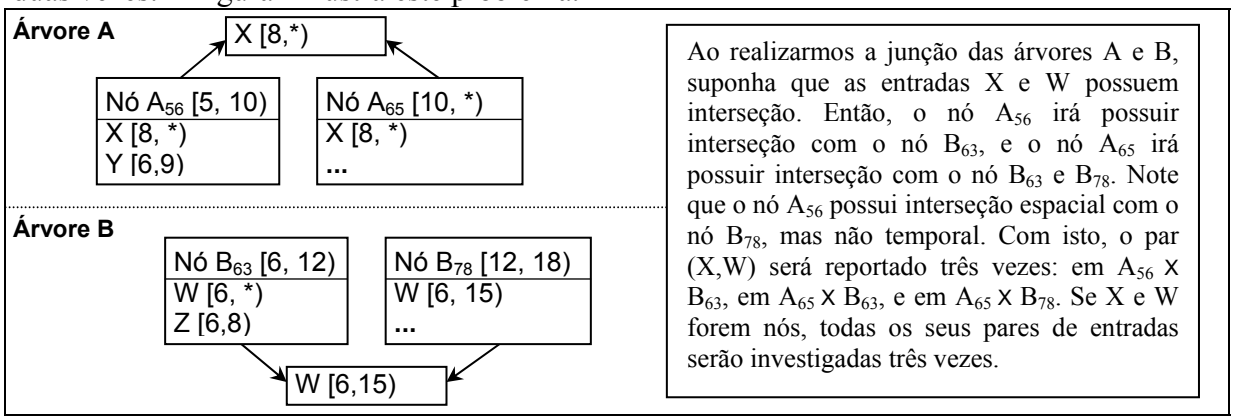


Figura 2 – Junção Espaçotemporal – problema dos pares duplicados

Pior do que isso, se elas forem reportadas em outro nó para o qual também existe uma entrada duplicada, elas serão reportadas duas vezes neste outro nó também, e assim por diante em cada nível da árvore até a raiz. Claramente esta não é uma boa abordagem.

4.4 Evitando Duplicatas

Em [ZAS00] foi proposto um algoritmo de busca por intervalo de tempo eficiente para a TR-Tree. Ele consiste em reescrever a consulta de seleção original, que era "encontre todos os MBRs que possuem sobreposição com o retângulo R e cujo tempo de vida intercepta o intervalo de consulta" para "encontre todos os MBRs que possuem sobreposição com o retângulo R e que, ou estão vivos no final do intervalo da consulta, ou foram mortos durante este intervalo". Esta pequena modificação nos sugere um algoritmo que consiste em percorrer a árvore, relatando as entradas que interceptam o retângulo de busca R mas, nos nós que já morreram, listando apenas as entradas mortas, pois as entradas vivas irão ser listadas mais adiante nos sucessores deste nó para cada uma das entradas vivas. O sucessor de um nó é o nó para onde este nó foi copiado no algoritmo de divisão de versão. Nos nós que estão vivos no final do intervalo de busca (ou seja, estão realmente vivos ou morreram em um instante depois do fim do intervalo de busca) devemos relatar todas as entradas, pois não iremos investigar os sucessores de deste nó para nenhuma entrada pois os mesmos terão tempo de

criação fora do intervalo de consulta. Este algoritmo foi de fato implementado e testado com bons resultados em [ZAS00].

Seguindo esta idéia, podemos estabelecer um algoritmo com estratégia semelhante para a junção e que não relate pares duplicados de entradas. Vamos supor que o algoritmo investigue cada par de nós que possuem interseção espacial e temporal apenas uma vez. Fatalmente, como vimos na figura 2, haverá pares que aparecerão mais de uma vez. Devemos portanto estabelecer um critério que permita decidir se um determinado par deve ou não ser listado. Uma boa abordagem seria listar os objetos no último tempo de sua interseção, pois como vimos, embora a árvore duplique entradas, apenas uma delas, a do nó que não possui sucessores, pode marcar a sua morte. A transformação da junção análoga à que foi feita para a seleção seria: "encontre todos os pares de MBRs com sobreposição que possuem o tempo final de sua interseção durante o intervalo da consulta, ou que estejam vivos no tempo final deste intervalo".

Usando a consulta acima podemos então estabelecer critérios para decidir se dois pares de entradas candidatas devem ou não ser relatadas quando encontradas. Chamaremos de pares candidatos de entradas aos pares de entradas que possuem interseção espacial e temporal, e sempre que mencionarmos se um nó ou entrada está morto estamos nos referindo ao seu estado no instante final do intervalo da consulta:

1) Se os nós onde o par candidato de entradas está são nós vivos no final do intervalo da consulta, temos que estes dois nós não possuem sucessores (ou seus sucessores não serão visitados, pois foram criados após o fim do intervalo da consulta), logo todos os pares candidatos de entradas devem ser relatados neste momento pois não iremos ter uma nova chance de relatá-los.
2) Se um dos nós está morto, e o outro está vivo no tempo final da consulta, devemos relatar apenas os pares candidatos em que as entradas que pertencem ao nó morto também estejam mortas, pois os pares candidatos de entradas que contém entradas vivas do nó morto irão aparecer de novo nos sucessores deste nó para cada uma destas entradas, e certamente cada um destes sucessores irá possuir interseção com o nó que está vivo.
3) Se ambos estão mortos, teremos um pouco mais de complicação: os pares candidatos em que estão ambas as entradas vivas certamente não devem ser relatadas, pois o mesmo irá aparecer novamente como par candidato nos respectivos sucessores. Da forma análoga, os pares candidatos que possuem apenas entradas mortas também devem ser relatados, pois esta é última vez que aparecerão na árvore (não possuem sucessores). Porém, os pares de entradas em que aparecem uma entrada morta e uma entrada viva apresentam a seguinte dificuldade: seja A o nó que contém a entrada viva e B o nó que contém a entrada morta. Se A morreu antes de B, o sucessor de A será comparado com B, e nele iremos novamente encontrar o mesmo par candidato de entradas. Note que este problema não irá surgir se os dois nós tiverem morrido no mesmo tempo, pois nenhum dos sucessores será comparado com o outro nó. Assim, não devemos listar os pares de entradas em que uma está viva e a outra está morta se o nó que contém a entrada viva morreu antes do nó que contém a entrada morta.

Com isto, teremos o seguinte algoritmo para a decisão de listar ou não um par candidato de entradas (ou seja, entradas que possuem interseção espacial e temporal):

```
function listaPar(eA, eB: Entrada; noA, noB: No; finalInt: Tempo): boolean;
begin
  if noA.vivo(finalInt) and noB.vivo(finalInt) then
    retorne true;
  else if (noA.vivo(finalInt) and noB.morto(finalInt) and eB.morto(finalInt)) or
          (noA.morto(finalInt) and noB.vivo(finalInt) and eA.morto(finalInt)) then
    retorne true;
  else // ambos os nós são mortos.
    if noA.tempoMorte = noB.tempoMorte and
       (eA.morta(finalInt) or eB.morta(finalInt)) then
      retorne true;
    else if (noA.tempoMorte < noB.tempoMorte and eA.morta(finalInt)) or
            (noA.tempoMorte > noB.tempoMorte and eB.morta(finalInt)) then
      retorne true;
    else
      retorne false;
end
```

Algoritmo 1: para decidir se um par de entradas candidatas deve ou nao ser listado.

Note que a função acima deve ser chamada durante o *plane-sweep* de um par de nós para cada entrada (em nosso algoritmo, chamamos listaPar no procedimento *internalLoop*). Algumas otimizações de chamadas de funções podem ser feitas, como por exemplo, se ambos os nós estão vivos devo relatar todos os pares candidatos, porém elas levariam a um código menos didático. Apenas para situar o leitor, mostraremos o código do *plane-sweep* de nós não otimizado (em nossos testes, utilizamos a versão otimizada). Em [BKS93] este algoritmo é chamado de *sortedIntersectionTest*. A variável "trocar" indica se a ordem dos pares deve ser trocada (para que não seja necessário escrever duas funções quase idênticas).

```
proc internalLoop( teste: Entrada; naoMarcado: integer; lista: ListaEntradas;
                   finalInt: Tempo; trocar: boolean; no_1, no_2: No;var saida: ListaEntradas )
begin
  k := naoMarcado;
  while k ≤ lista.tamanho and lista[ k ].min.x < teste.max.x do
  begin
    if teste.min.y ≤ lista[ k ].max.y and teste.max.y ≥ lista[ k ].min.y and
       teste.intervalo ∩ lista[ k ].intervalo ≠ ∅ and
       listaPar( teste, lista[ k ], no_1, no_2, finalInt ) then
      if trocar = false then
        saida.insere( Par( teste, lista[ k ] ) )
     else
        saida.insere( Par( lista[ k ], teste ) );
    k := k + 1;
  end
end
```

Algoritmo 2: para decidir se um par de entradas candidatas deve ou nao ser listado.

```
proc sortedIntersectionTest( noA, noB: No; rBusca: Retangulo; intBusca: Intervalo;
                             var saida: ListaEntradas )
begin
  saida := ListaVazia; i := 1; j := 1;
  listaA := lista de entradas de noA que possuem interseção espacial com rBusca e
            temporal com intBusca, ordenadas na menor cordenada x e na menor coordenada y;
  listaB := lista de entradas de B que possuem interseção espacial com rBusca e
            temporal com intBusca, ordenadas na menor cordenada x e na menor coordenada y;
  while i ≤ listaA.tamanho and j ≤ listaB.tamanho do
    if listaA[ i ].min.x < listaB[ j ].min.x then begin
      internalLoop(listaA[i], j, listaB, intBusca.final, false, noA, noB, output);
      i := i + 1;
    end
    else begin
      internalLoop(listaB[j], i, listaA, intBusca.final, true, noB, noA, output);
      j := j + 1;
    end;
end
```

Algoritmo 3: Procedimento para realizar a junção em dois nós da árvore.

Por motivos de espaço, não iremos mostrar aqui o algoritmo que percorre as TR-Trees de forma sincronizada, pois ele é análogo ao usado para percorrer a R*-Tree apresentado em [BKS93], e é o mesmo usado também para percorrer as árvores da 2+3D R-Tree. Apenas cabe ressaltar que, no nível do vetor de nós raízes, iremos realizar uma junção para cada par de nós

raiz que possui interseção espacial e temporal. Não há necessidade de melhoramentos neste nível, pois a cada instante cada R-Tree da TR-Tree possui um e somente um nó raiz, o que garante que cada par de nós raízes será investigado apenas uma vez.

5. Resultados Experimentais

5.1 Geração dos Dados

Para nossos testes de avaliação de desempenho utilizamos conjuntos de dados gerados pseudo-aleatoriamente através do gerador de dados espaçotemporais chamado GSTD (*Generate Spatio Temporal Data*) proposto em [TSN99]. Os objetos destes conjuntos de dados são todos retangulares com distribuição espaçotemporal uniformemente aleatória. Fizemos pequenas alterações no programa para escalar os retângulos de forma que estivessem contidos numa janela de (0, 0) a (1000, 1000). Esta escala é importante para que a dimensão do espaço encontre-se na mesma ordem de grandeza da dimensão do tempo. Variamos a quantidade de versões na base de dados para os valores de 250, 500, 750 e 1000. Em cada conjunto de dados inserimos 50.000 objetos inicialmente e procuramos efetuar, em média, este mesmo número de alterações nos objetos existentes. Para cada número de versões geramos cinco pares conjunto de dados variando a semente de geração dos números aleatórios, formando assim 20 pares de conjuntos de dados. Utilizamos páginas de 8 kbytes e um cache de disco LRU independente para cada árvore com 97 páginas de 8 kbytes, mais um cache com a altura da árvore (nos testes, 3) para guardar o caminho da raiz até a folha. A máquina utilizada foi um Pentium III 700 com 256Mb de memória principal, usando o Linux Conectiva 6.0 e gcc 2.95.

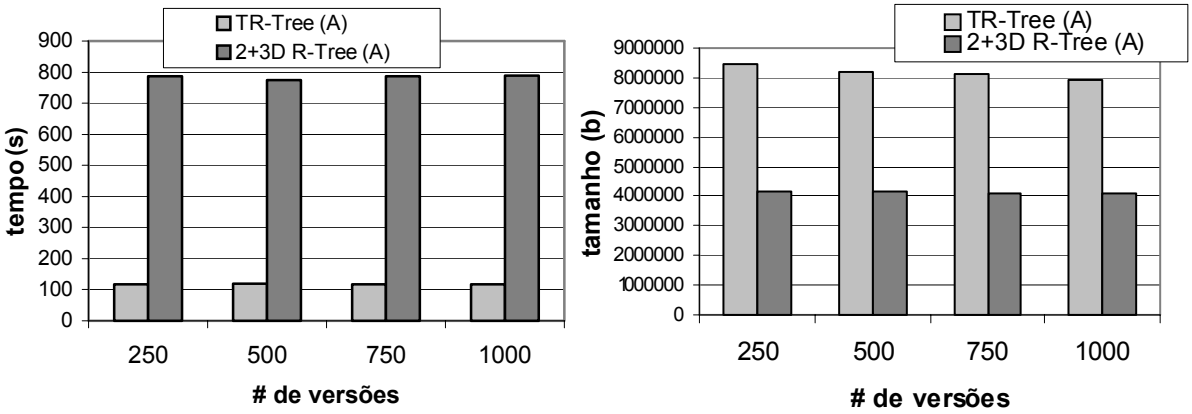


Figura 3: Gráficos do tempo (esquerda) de construção e tamanho (direita) dos índices.

O tempo de CPU foi medido usando-se o tempo relatado pelo sistema operacional como tempo de usuário, e o tempo gasto em acessos a disco foi medido pelo número de acessos a disco multiplicado pelo tempo médio de 1 acesso (em nossos testes cerca de 5 milisegundos), conforme apresentado em [BKS93] e em uma série de outros trabalhos, de forma a compensar os efeitos do cache do sistema operacional. A Tabela 2 mostra a utilização de espaço nos nós para as duas estruturas em estudo com páginas de 8 kbytes.

	TR-Tree	**2+3D R-Tree**	
		2D R-Tree	3D R-Tree
Min. Entradas / Nó	85	170	145
Máx. Entradas / Nó	255	340	291

Tabela 2 – Parâmetros dos dois MAETs para páginas de 8 kbytes.

Medimos o tempo de construção dos índices bem como o tamanho em bytes dos arquivos gerados pela indexação. Estes resultados encontram-se nos gráficos da figura 3. O valor plotado nos gráficos é o valor médio para cada um dos cinco conjuntos de dados diferentes com as mesmas características.

Podemos analisar através destes dois gráficos que o tamanho e o tempo de criação de ambos os índices praticamente não sofrem influência do número de versões existentes. Embora o tamanho dos índices da TR-Tree seja praticamente o dobro do tamanho dos índices da 2+3D R-Tree, seu tempo de criação é cerca de 6,5 vezes menor na média.

5.2 Descrição das Consultas de Junções

Para cada dupla de conjunto de dados aplicamos uma série de junções espaçotemporais. Todas as junções foram aplicadas na janela (0,0), (1000,1000) de forma que estamos interessados em buscar todos os objetos que possuem interseção entre si em determinados intervalos de tempo. Para os intervalos de tempo variamos os valores em 1, 20 e 50% do total de versões de tempos dos conjuntos de dados, que no caso das árvores com 1000 versões de tempos correspondem a: [990,1000], [800,1000] e [500,1000]. Temos então três junções para cada dupla de conjunto de dados totalizando 120 junções.

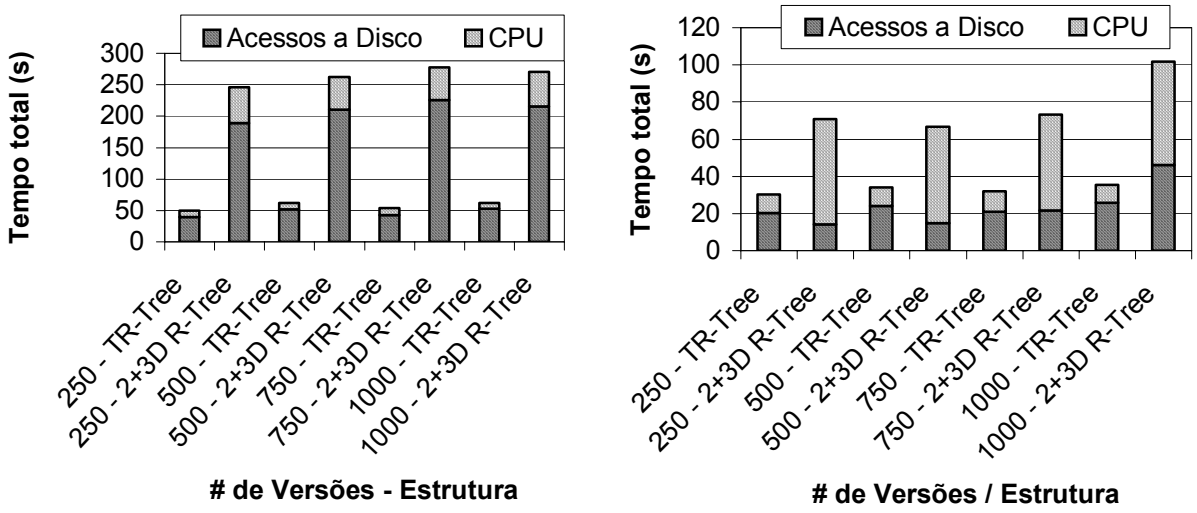


Figura 4: Gráficos do tempo de junção para 1% (esquerda) e 20% (direita) do total de versões.

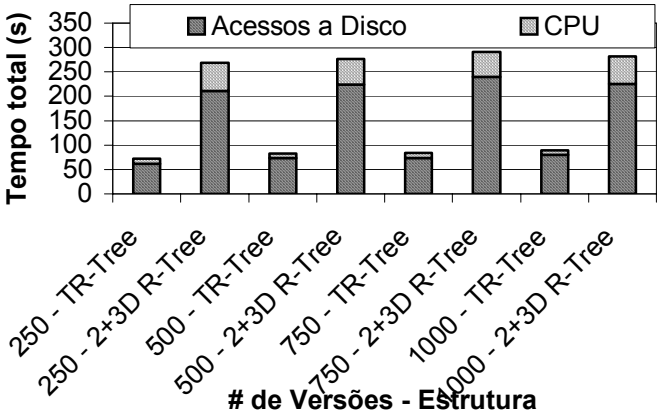


Figura 5: Gráfico do tempo de junção para 50% do total de versões.

O resultados apresentados nas Figuras 4 e 5 foram agrupados por intervalo de tempo, por estrutura e por tipo de conjunto de dados, usando-se os tempos médios e somando-se os tempos de CPU e acesso a disco.

Pelos gráficos das Figuras 4 e 5, podemos verificar que a TR-Tree consistentemente apresenta um desempenho superior ao da 2+3D R-Tree não importando qual o tamanho do intervalo de tempo utilizado. Cabe lembrar que o intervalo de tempo irá determinar o número de junções no vetor de nós raiz, resultando em mais junções de R-Trees. Ainda assim, no intervalo com 50% dos tempos a TR-Tree apresentou um desempenho superior à 2+3D R-Tree. Além disso, podemos concluir que as junções envolvendo intervalos pequenos são CPU-*bounded* no caso da 2+3D R-Tree, pois a mesma guarda versões antigas com novas, além de ter mais entradas por nó conforme a Tabela 2.

6. Conclusões

Este é um trabalho pioneiro no processamento de junções espaçotemporais propondo um novo algoritmo para o problema, e que pode ser utilizado em métodos de acesso espaçotemporais parcialmente persistentes semelhantes, como a PPR-Tree por exemplo.

Pelos testes realizados, podemos concluir que a TR-Tree, utilizando o nosso algoritmo, apresentou consistentemente resultados de desempenho melhores do que a 2+3D R-Tree, perdendo apenas no quesito espaço de armazenamento – o que mesmo assim não chega ser comprometedor, pois essa diferença não resulta em tempo de criação maior, muito pelo contrário.

Assim, enquanto em [ZAS00] concluímos que a TR-Tree é a estrutura mais promissora para indexar seleções espaçotemporais, neste trabalho concluímos que a TR-Tree é também uma promissora estrutura para indexar dados para o processamento de junções espaçotemporais.

Como trabalhos futuros, podemos citar a realização de testes em dados de aplicações com algum tipo de dominância temporal ou espacial, e a possibilidade de paralelização dos algoritmos da TR-Tree.

Bibliografia

[BGO+96] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer: "An Asymptotically Optimal Multiversion B-tree", The VLDB Journal, vol.5(4), pp. 264-275, December 1996.

[BKS+90] N. Beckmann, H. P. Kriegel, R. Schneider, et al. 1990. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: Proceedings of the ACM SIGMOD Intl. Conf on Management of Data, Atlantic City, NJ, 1990.

[BKS93] T. Brinkhoff, H. P. Kriegel, and B. Seeger: "Efficient processing of Spatial Joins Using R-Trees". In Proceedings of the 1993 ACM-SIGMOD Conference, Washington, DC, USA,May 1993.

[G84] A. Guttman: "R-Trees: A Dynamic Index Structure for Spatial Searching". In Proceedings of the ACM SIGMOD Intl. Conf on Management of Data, Boston, MA, 1984.

[K+01] G. Kollios, D. Gunopulos, V.J. Tsotras, A. Delis, M. Hadjieleftheriou: "Indexing Animated Objects Using Spatiotemporal Access Methods", To appear at IEEE Transactions on Knowledge and Data Engineering

[KF94] I. Kamel and C. Faloutsos, "Hilbert R-Tree: An improved R-Tree using fractals", In Proceedings of the 20th Very Large Databases Conference, pages 500-509, September 1994.

[KTF95] A. Kumar, V. J. Tsotras, and C. Faloutsos: “Access Methods for Bitemporal Databases”. In Recent Advances in Temporal Databases, J. Clifford, and A. Tuzhilin (eds), pp. 235--254, Springer Verlag, 1995.

[KTF97] A. Kumar, V. J. Tsotras, and C. Faloutsos: “Designing Access Methods for Bitemporal Databases”. TR3764, Dept. of Comp. Sci. University of Maryland, 1997.

[LS89] D. Lomet, B. Saltzberg, “Access Methods for Multiversion Data”, Proceedings of ACM SIGMOD Conference, 1989.

[MK90] Y. Manolopoulos, and G. Kapetanakis: “Overlapping B + -trees for temporal data”. In Proceedings of the 5th Jerusalem Conference on Information Technology (August 1990), pp. 491- 498, 1990.

[NS98] M. A. Nascimento, J. R. O. Silva, “Towards Historical R-trees”, Proceedings of ACM Symposium on Applied Computing (ACM-SAC), 1998.

[SJ99] Saltenis S. and C. S. Jensen, “R-Tree Based Indexing of General Spatio-Temporal Data”", TimeCenter Technical Report TR-45, December 1999, and Chorochronos Technical Report CH-99-18, December 1999.

[ST94] Salzberg, B., and Tsotras, V.: “A comparison of access methods for time evolving data”. Tech. Rep. NU-CCS-94-21, College of Computer Science, Northeastern University, Boston, USA, 1994.

[TSN99] Theodoridis, Y., Silva, J.R.O., and Nascimento, M.A. On the Generation of Spatiotemporal Datasets. 6th Intl. Symposium on Spatial Databases (SSD'99), Hong Kong, China, July 1999.

[TSPM98] Yannis Theodoridis, Timos Sellis, Apostolos N. Papadopoulos, and Yannis Manolopoulos: “Specifications for Efficient Indexing in Spatiotemporal Databases”, Proceedings of SSDBM’98, Capri, Italy, July 1-3 1998.

[TVS96] Y. Theodoridis, M. Vazirgiannis, and T. Sellis: “Spatio-Temporal Indexing for Large Multimedia Applications” Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems (ICMCS), 1996.

[XHL90] X. Xu, J. Han, W. Lu: “RT-tree: An Improved R-tree Index Structure for Spatiotemporal Databases”, Proceedings of the 4th International Symposium on Spatial Data Handling (SDH), 1990.

[ZAS00] Zimbrão, G., Almeida, V. T, Souza J. M, “Efficient Processing of Spatiotemporal Queries in Temporal Geographical Information Systems”, 6th International Conference on Information Systems, Analysis and Synthesis ISAS 2000. http://www.cos.ufrj.br/~zimbrao/zas00.pdf

[ZAS99] G. Zimbrão, V. T. Almeida, and J. M. Souza: “The Temporal R-Tree”. Technical Report ES492/99, COPPE/Federal University of Rio de Janeiro, Brazil, March 1999. www.cos.ufrj.br/~zimbrao/zaz99.pdf