Universidade Federal de Goiás
Regional Jataí
Curso de Ciências da Computação

Willian Garcias de Assunção

Avaliação de Métodos de Divisão de Nós em Árvores R-Tree para Dados Espaciais do tipo Linha

Jataí-Goiás
2016

Willian Garcias de Assunção

Avaliação de Métodos de Divisão de Nós em Árvores R-Tree para Dados Espaciais do tipo Linha

Monografia apresentada ao Curso de Ciências da Computação da Universidade Federal de Goiás, Regional Jataí, como requisito parcial para obtenção do título de BACHAREL em Ciências da Computação.

Orientador: Prof. Thiago de Oliveira Borges

MsC. em Sistemas Distribuídos

Jataí-Goiás

2016

Garcias de Assunção, Willian

Avaliação de Métodos de Divisão de Nós em Árvores R-Tree para Dados Espaciais do tipo Linha / Willian Garcias de Assunção - 2016

xx.p

1. 2. . I.Título.

CDU 000.00

Willian Garcias de Assunção

Avaliação de Métodos de Divisão de Nós em Árvores R-Tree para Dados Espaciais do tipo Linha

Monografia apresentada ao Curso de Ciências da Computação da UFG, como requisito para a obtenção parcial do grau de BACHAREL em Ciências da Computação.

Aprovado em 15 de Março de 2016

BANCA EXAMINADORA

Prof. Thiago de Oliveira Borges

MsC. em Sistemas Distribuídos

Italo Tiago da Cunha

Msc. em Ciências da Computação

Emília Alves Nogueira

Msc. em Ciências da Computação

Aos meus pais e minha família que sempre me deram apoio durante todo o curso de graduação.

Aos amigos, pelo apoio e companheirismo.

Agradeço primeiramente a Deus, que permitiu a finalização de uma mais uma etapa em minha vida. Ao professor Thiago de Oliveira Borges que não mediu esforços em contribuir com parte do seu trabalho, a todos aqueles que contribuíram de forma direta ou indireta para a realização deste trabalho.

"Paciência e perseverança tem o efeito mágico de fazer as dificuldades desaparecerem e os obstáculos sumirem".

John Quincy Adams

Resumo

Para armazenamento e recuperação de dados espaciais é necessária a utilização de uma estrutura específica para dados multidimensionais ou complexos. Na literatura foram propostas diversas estruturas de indexação, cada uma com suas características e comportamentos específicos. A R-Tree é uma árvore hierárquica, semelhante a árvore B-Tree, que agrupa objetos colococalizados, empregando retângulos envolventes, chamados de MBR, (Minimum Bounding Rectangle) conhecido também como retângulo envolvente. A implementação dessas estruturas possui internamente um algoritmo de divisão chamado *split*, que tem como função montar a estrutura da árvore, deixando os objetos co-localizados juntos em grupos (nós) e manter o balanceamento da árvore. A realização de *split* em nós é um processo crítico para a performance de uma estrutura multidimensional, pois o *split* ajuda determinar a forma final que será uma estrutura de árvore-R. O presente trabalho avaliou um conjunto de algoritmos de *splits* existentes, aplicando-os a dados espaciais do tipo ponto, polígono e em especial as linhas, de forma a identificar situações em que cada algoritmo de *split* construia árvores mais balanceadas, com maior preenchimento de nós, ocupando menos espaço em disco, afim de que seja possível efetuar buscas de dados mais eficientes.

Palavras-chaves: split em R-Tree; estruturas multidimensional; R-Tree.

Abstract

For storage and retrieval of spatial data it is necessary to use a special structure for multidimensional or complex data. In the literature it has been proposed various indexing structures, each with their specific characteristics and behaviors. The R-Tree is a hierarchical tree, similar to B-Tree tree, which groups co-lococated objects, using surrounding rectangles, called MBR (minimum bounding rectangle) also known as rectangle surrounding. The implementation of these structures internally has a division algorithm called split, which has the function to assemble the tree structure, leaving the co-located objects together into groups (node) and maintain balance of tree. The performing split in node is a critical process for the performance of a multidimensional structure, because the split help determine the final shape will be an R-Tree overall goal structure of this study is to evaluate a set of existing splits algorithms, applying the spatial line type data, to identify where each algorithm split to build more balanced trees more fill node up less disk space, so that you can make more efficient data searches.

Keywords: split R-Tree; multidimensional structures; R-Tree.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Vannevar Bush afirmou, a mais de meio século, que um dos problemas mais difíceis para a comunidade científica era buscar formas ou mecanismos para automatizar ações como: armazenar, indexar e recuperar informações [BUSH 1945]. O cenário afirmado por Bush ainda se mantém, porém em maior escala. Em alguns contextos da atualidade é difícil manipular grandes quantidades de dados de forma eficiente, pois o custo computacional destas operações é alto.

Neste cenário, estruturas de dados foram criadas para tratar problemas de armazenamento e recuperação de dados. As árvores binárias como AVL e a Red-Black, são estruturas de armazenamento e organização de dados simples e eficientes, isso porque permitem realizar operações em tempo de complexidade - O(log n) [Cormen et al. 2009]. Estas árvores são estruturas unidimensionais, porque organizam e indexam dados com somente uma dimensão, ou simples, como valores inteiros ou números de ponto flutuante.

Além dos dados unidimensionais, existem os dados que são representados por mais de uma dimensão e conhecidos como dados multidimensionais. Um exemplo de dados multidimensionais são os dados espaciais, que representam informações sobre locais físicos e formas de objetos geométricos. Um tipo comum de dado espacial existente é o geoespacial, ou seja, dados que representam a geografia do mundo, como: localização espacial, relevo, limites de áreas, característica de uma determinada região, vegetação, hidrografia, malha viária, construções em geral, entre outros. Estes dados são formados por conjuntos de pontos definidos por uma latitude e longitude e, portanto, possuem duas ou mais dimensões e são representados como objetos em forma de pontos, polígonos e linhas.

Um tipo particular de dado espacial é o dado do tipo linha, usado para representar objetos como avenidas, rios e seus afluentes, rodovias, curvas de nível, contorno de relevo, dentre outros. Este tipo de dado é particular se comparado com os demais objetos espaciais polígonos e pontos, principalmente devido à sua extensão geográfica e a sinuosidade natural de alguns contornos geográficos como hidrografia e relevo.

Para armazenamento e recuperação de dados espaciais é necessária a utilização

de uma estrutura específica para dados multidimensionais ou complexos. Na literatura foram propostas diversas estruturas de indexação, cada uma com suas características e comportamentos específicos. Em 1984, Guttman propôs a árvore R-Tree para organização de objetos multidimensionais. A R-Tree é uma árvore hierárquica, semelhante a árvore B-Tree, que agrupa objetos colococalizados, empregando retângulos envolventes, chamados de MBR[1] [Guttman 1984].

Após Guttman propôr a sua estrutura para dados espaciais, novas variantes da R-Tree foram criadas por outros autores, como a R*-Tree [Beckmann et al. 1990], R0-Tree [Zhang e Xia 2004] e a R+-Tree [Sellis, Roussopoulos e Faloutsos 1987]. Essas estruturas implementaram várias mudanças na R-Tree para melhorar seu desempenho. A R^+-Tree para um domínio de aplicações específicas, a R*-Tree e a R0-Tree para aplicações em sistemas de informações geográficas (SIG) e aplicações de bancos de dados espaciais. Com isso, essas estruturas foram conhecidas pela sua eficiência para trabalhar com objetos multidimensionais e, portanto, têm sido empregadas em inúmeras aplicações para indexar objetos espaciais [Sleit e Al-Nsour 2014].

A implementação dessas estruturas possui internamente um algoritmo de divisão (*split*), que tem como função montar a estrutura da árvore, formando grupos (nós) com os objetos co-localizados e manter o balanceamento da árvore. A literatura apresenta vários algoritmos de *split*, como, por exemplo: linear, quadrático e exponencial [Guttman 1984]; algoritmo de corner [Sleit e Al-Nsour 2014], algoritmo de *split* da R*-Tree [Beckmann et al. 1990], entre outros. Segundo Sleit [Sleit e Al-Nsour 2014] é necessário que os algoritmos de *splits* sejam bons e executem em um tempo de complexidade razoável, produzindo uma árvore balanceada, que reduz o tempo de processamento de buscas, enquanto mantém o uso de armazenamento proporcional ao tamanho do conjunto de dados. No entanto, este problema é complexo e a solução exponencial proposta por Guttman [Guttman 1984] não é aplicável na prática. Os demais algoritmos propostos são baseados em heurísticas e possuem complexidade menor.

A realização de *split* em nós é um processo crítico para a performance de uma R-Tree, pois o *split* determina a forma final da estrutura [Sleit e Al-Nsour 2014]. A forma final da árvore também impacta nos algoritmos de busca. Há duas métricas principais que podem afetar uma consulta em uma árvore-R: I) a área total de uma MBR e, II)

[1] *Minimum Bounding Rectangle* (MBR) - conhecido também como retângulo envolvente.

área de sobreposição entre as MBRs. O algoritmo de *split* decide como um conjunto de objetos de um nó da árvore será particionado observando estas métricas, afim de reduzir a área de sobreposição [Liu, Fang e Han 2009].

Um algoritmo de *split* é dito eficiente quando se realiza boas divisões em nós com poucas sobreposições entre as MBRs. Segundo [Outsios Evangelos 2011] os algoritmos de *split* que buscam maximizar o preenchimento dos nós da árvore, automaticamente economizam espaço de armazenamento. Isso ocorre porque menos nós serão criados e menos páginas de disco serão necessárias.

Para [Caldwell 2005], quando um algoritmo de *split* não consegue construir uma boa MBR, é porque ocorreu uma tentativa de agrupar objetos com grande extensões. A má realização de um *split* é uma das maiores preocupações na organização de dados espaciais. Isso porque aumenta drasticamente o custo de realizar operações de consulta nas árvores-R.

Nestes experimentos, é comum que a realização de testes aconteça com objetos espaciais dos tipos pontos e polígonos, e uma generalização de resultados seja feita para o tipo de objeto linha. Outras vezes, os testes dos objetos do tipo linha são deixados para trabalhos futuros devido a maior dificuldade em realizar e interpretar os resultados dos experimentos. Este fato pode ser observado nos trabalhos de [Beckmann et al. 1990], [Zhang e Xia 2004] e [Sleit e Al-Nsour 2014]. No entanto, entendemos que esta generalização não é adequada, devido as particularidades citadas dos objetos espaciais do tipo linha.

Neste trabalho foi avaliado um conjunto de algoritmos de *splits* existentes, aplicando-os a dados espaciais do tipo ponto, polígono e em especial as linhas, de forma a identificar situações em que cada algoritmo de *split* construa árvores mais balanceadas, com maior preenchimento de nós, ocupando menos espaço em disco, afim de que seja possível efetuar buscas de dados mais eficientes. Para atingir este objetivo foram implementados algoritmos de *split* propostos por Guttman, os quais foram validados, confirmando sua corretude, também foram realizados experimentos com um conjunto de dados espaciais do tipo ponto, polígono e especialmente do tipo linha para determinar a partir dos experimentos, o melhor algoritmo de *split* para cada cenário.

O restante do trabalho está organizado da seguinte forma: O Capítulo 2, apresenta um levantamento bibliográfico sobre dados espaciais, árvore R-Tree, variantes

da R-Tree e os algoritmos de *split*. No Capítulo 3 são apresentados os trabalhos correlatos ao que aqui se propõe. Capítulo 4, mostra os procedimentos metodológicos de como ocorreram os experimentos. Logo a seguir, são apresentados os detalhes dos resultados dos experimentos no Capítulo 5. Por fim, no Capítulo 6 apresenta a conclusão do trabalho e os trabalhos futuros.

2 Referencial Teórico

2.1 Representação de dados espaciais

Dados espaciais ou dados geográficos se diferenciam dos demais tipos de dados pela sua componente espacial. Estes dados são representações físicas terrestres ou representações de localizações no espaço, ou seja, possuem uma latitude e longitude. Existem duas formas distintas de representar os dados espaciais: vetorial e matricial [Davis e Rocha 2007].

Na forma vetorial, cada objeto é representado por ao menos um par de coordenadas. Os objetos podem ser representados por pontos, polígonos e linhas conforme mostra a Figura 2.1. Objetos do tipo ponto são usados para representar localizações de ocorrências únicas, por exemplo, crimes ou doenças. Objetos do tipo linha são utilizados na representação de rios, rodovias, redes de esgoto, trajetos e outros semelhantes. Polígonos são utilizados em representações de áreas ou regiões como, por exemplo, cidades, estados, países, e outros similares.

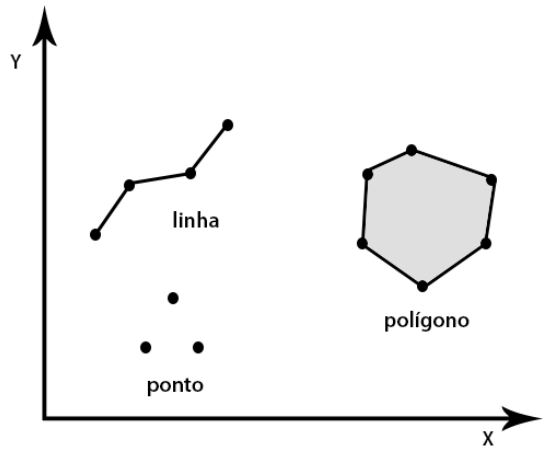

Figura 2.1: Representação vetorial. Adaptado de [Camara 2001].

Na forma matricial, as representações são realizadas por meio de uma matriz utilizando-se dos valores associados entre as linhas e colunas para representar os objetos em forma de imagem. A Figura 2.2 ilustra a representação matricial. Os dados matriciais podem ser capturados por satélites ou sensores, e depois processado para dados vetoriais [Camara 2001].

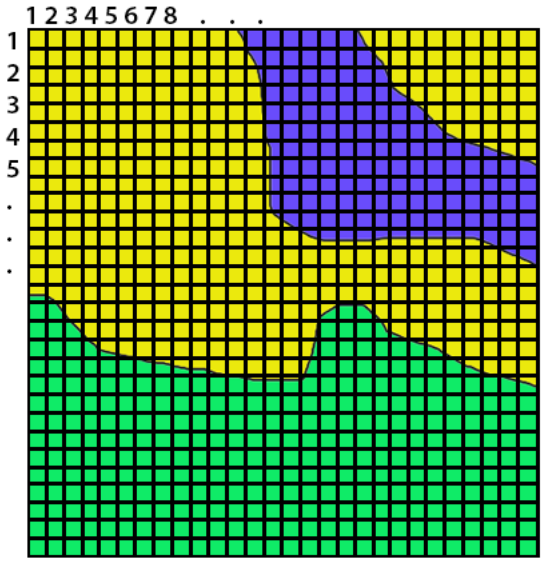

Figura 2.2: Representação matricial.

A escolha de qual modelo de representação usar é ligada a vários fatores, pois as duas representações possuem vantagens e desvantagens. Por exemplo, na representação de ambientes com fenômenos de alteração contínua é melhor a utilização da classe matricial. Já a representação por meio de vetores (coordenadas) se aplica em situações nas quais se faz necessário uma alta precisão nas representações [Camara 2001].

Para esse trabalho, será adotado o modelo de representação vetorial, pois é a tentativa mais exata de reproduzir um objeto. O dado vetorial é usado devido sua maior eficiência de processamento e a exatidão nas representações. Os objetos são reduzidos a três formas básicas: pontos, polígonos e linhas. A partir dessas formas, no modelo vetorial é possível que a localização e aparência gráfica de cada objeto sejam representadas por um ou mais pares de coordenadas. Portanto, é ideal a utilização de coordenadas para esse trabalho, visto que as árvores-R organizam apenas dados vetoriais, para fins de armazenamento e recuperação.

2.2 Estruturas de indexação unidimensionais

Existem na literatura estruturas como AVL, B-Tree e a Splay-Tree que são usadas para organizar dados unidimensionais. Esse tipo de estrutura é constituída de um nó raiz, nós intermediários (conhecidos como nós diretórios) e nós folhas, sendo que cada nó possui campos *esquerdo*, *direito* e p que apontam para nós filhos da esquerda, filhos da direita e

ao pai respectivamente. Em casos onde há a ausência do pai ou do filho, aponta-se para um elemento nulo. O único campo em que se tem um pai nulo é o nó raiz, visto que é a primeira entrada da árvore. Estruturas do tipo árvore admitem muitas operações de conjuntos dinâmicos, como: busca, inserção e remoção.

Dois exemplos de árvores binárias são AVL e a Red-Black. Essas árvores são balanceadas, ou seja, todos os caminhos desde a raiz até os nós folhas possuem o mesmo tamanho. Adicionalmente, as operações básicas realizadas pela árvore binária possuem o tempo proporcional à sua altura, em termos matemáticos dizemos que a complexidade para realização de uma operação em uma árvore binária é $O(log\ n)$, sendo n a quantidade de nós da árvore [Cormen et al. 2009].

Em situações nas quais é desejável armazenar uma grande quantidade de dados, a memória principal do computador pode não ser suficiente e, para resolver esse problema, é comum armazenar os dados em memória secundária. Uma das maneiras é utilizar a representação de índices por meio de uma árvore B-Tree.

Vários sistemas de bancos de dados utilizam estruturas de árvores similares a B-Tree, ou suas variações, para armazenar informações [Jaluta 2014]. As árvores B buscam minimizar o acesso ao disco durante o processo de busca agrupando várias chaves em um único nó chamado de página [Cormen et al. 2009].

Ambas estruturas apresentadas são estruturas unidimensionais, nas quais a chave de pesquisa deve ser formada por apenas um atributo ou pela junção de vários atributos. Em sistemas de geoprocessamento, banco de dados espaciais e sistemas de informações geográficas é necessário armazenar e realizar operações com dados multidimensionais e, portanto, estas estruturas não podem ser usadas, visto que não preservam a distância entre os elementos no espaço [Cormen et al. 2009] [Bed e Monteiro 2008].

2.2.1 Estruturas de Indexação Multidimensionais

As estruturas de indexação multidimensionais buscam dividir os objetos espaciais de acordo com a sua proximidade espacial e reduzir a quantidade de busca necessária por um objeto. O objetivo de uma estrutura multidimensional é proporcionar uma rápida recuperação dos objetos espaciais, desde que satisfaçam sua organização e relacionamento topológico. Cada estrutura organiza seu espaço de forma diferente, e fazendo com que

tenha comportamentos diferentes em relação a geometria de cada objeto. Portanto, estruturas multidimensionais foram propostas e projetadas para prover um caminho otimizado em armazenar e recuperar dados espaciais [Bed e Monteiro 2008].

A literatura apresenta um conjunto de métodos e estruturas de acesso multidimensionais capazes de armazenar e recuperar objetos espaciais, as quais constituem o objeto de estudo deste trabalho e estão organizadas da seguinte forma: A Seção 2.3 introduz estrutura R-Tree. Seções 2.4 e 2.5 operações da R-Tree, bem como os métodos de divisões de nós. A Seção 2.6 apresenta as variantes da R-Tree: R*-Tree e R0-Tree.

2.3 R-Tree

A limitação de estruturas de indexação existentes até 1984, para objetos multidimensionais, motivou o desenvolvimento de pesquisas para a criação de uma nova estrutura que fosse capaz de armazenar objetos multidimensionais. Guttman, em 1984, propôs a R-Tree como sendo uma nova estrutura de indexação multidimensional hierárquica na forma de uma árvore semelhante a B-Tree [Guttman 1984].

A R-tree é uma árvore balanceada assim como a B-Tree, mas para o espaço multidimensional, na qual os objetos são aproximados por meio de retângulos envolventes. As folhas da árvore apontam para o objeto espacial e os nós diretórios para outras páginas. A dimensão do espaço é dividido hierarquicamente, sobrepondo retângulos de limites mínimos (MBR - *minimum bounding rectangle*), na qual indexam geometrias de várias formas levando em consideração a sua aproximação no espaço multidimensional. Duas MBR podem ser vista na Figura 2.3.

Uma MBR é o menor retângulo que envolve totalmente um objeto espacial, que possui os lados paralelos aos eixos X e Y. A área adicional da Figura 2.3 necessária para cobrir o polígono como um todo é chamada de área morta e não faz parte do objeto espacial. Quando duas MBR estão próximas ao nível de uma interceptar a outra, denomina-se que há uma área de sobreposição durante uma busca na R-Tree. Ao contrário da B-Tree, onde uma busca percorre apenas um ramo da árvore, na R-Tree, a sobreposição de MBR's obriga que o algoritmo desça duas ou mais sub-árvores, o que provoca um aumento no tempo de processamento.

A estrutura da R-Tree é apresentada nas Figuras 2.4 e 2.5, bem como as

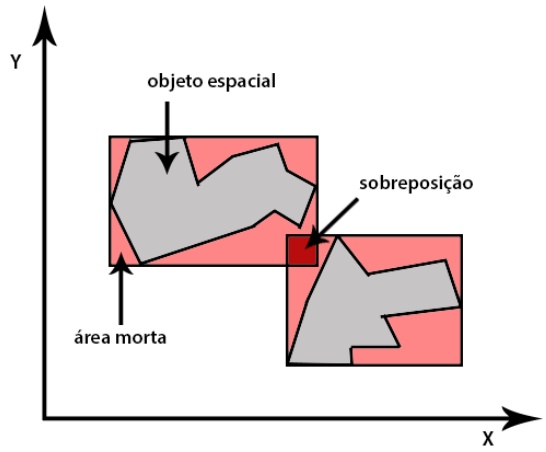

Figura 2.3: Minimum Bounding Rectangle (MBR).

relações de sobreposições que podem existir entre as MBRs. Na Figura 2.4, os objetos espaciais são agrupados de acordo com sua proximidade espacial, e posteriormente se tornando em MBRs, que são representadas pelas linhas serrilhadas. Com as MBRs construídas, pode se verificar a existência de uma relação entre as próprias MBRs. Essa relação ocorre pelo fato de haver sobreposições entre as MBRs, fazendo com que, em uma busca por um objeto espacial seja necessário acessar todas as áreas sobrepostas.

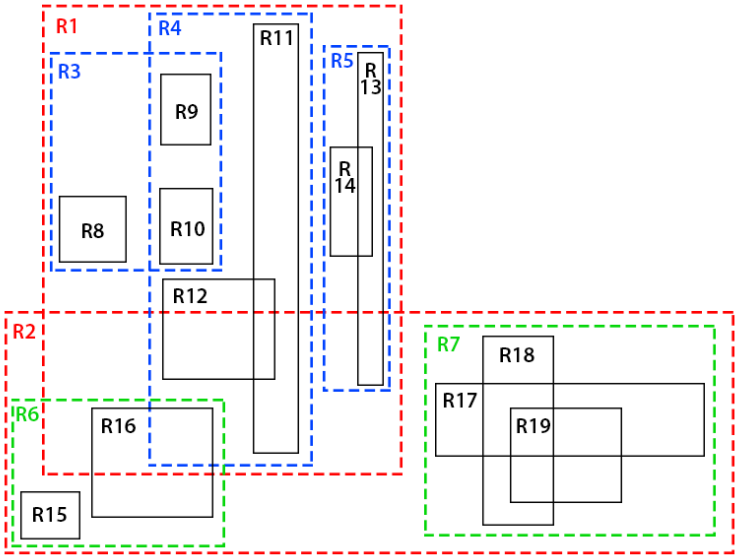

Figura 2.4: Representação das MBRs na R-Tree. Adaptado de [Guttman 1984].

A Figura 2.5 mostra a estrutura de uma árvore R-Tree similar a estrutura da árvore B. No entanto, cada nó da árvore R-Tree é gerado a partir de uma MBR. Os nós correspondem a páginas de disco com as seguintes características: nós diretórios apontam para os nós filhos (MBR); nós folhas apontam para os objetos.

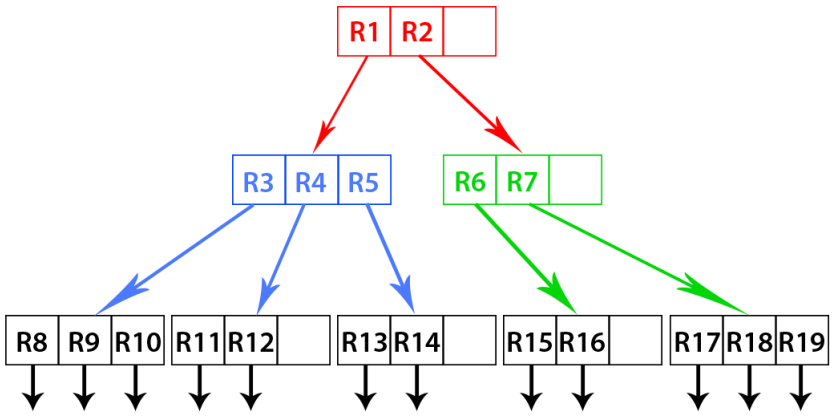

Figura 2.5: Estrutura da árvore R-Tree. Adaptado de [Guttman 1984].

De forma mais precisa são apresentadas abaixo as principais características que definem uma estrutura como a R-Tree.

- Cada folha contém entradas do tipo (*I, Tid*), em que *Tid* se refere (pode ser um ponteiro) a uma tupla na base de dados, contendo os atributos não espaciais do objeto. E *I* corresponde ao MBR deste objeto;
- Cada nó diretório contém entradas do tipo (*I, Cid*), em que *Cid* é um ponteiro para algum filho. *I* corresponde ao MBR que cobre todas as regiões descritas pelos MBR dos nós inferiores;
- É considerado um M sendo um número máximo de entradas em um nó e m um parâmetro que especifica o número mínimo de entradas em um nó;
- Todo nó diretório tem entre m e M filhos, a menos que seja a raiz;
- A raiz ter dois filhos, tanto como folha ou como diretório;
- Todas as folhas aparecem no mesmo nível.

A R-Tree nos últimos anos ganhou muita popularidade e chegou a ser incorporada em vários produtos acadêmicos e comerciais, tais como sistemas para informações geográficas e sistemas gerenciadores de bancos de dados. Com essa popularidade crescente, novos interesses por pesquisas em índices multidimensionais foram despertados, motivando ainda mais o avanço da estrutura de Guttman, e propiciando o surgimento das variantes da R-Tree.

A R-Tree tem sido usada na prática por sua capacidade de armazenar objetos espaciais de várias formas, responder a várias consultas e simplicidade. As áreas de aplicações mais comuns são os sistemas de informações geográficas, CAD (desenho por computador), visão computacional e robótica [Yi 2012].

2.4 Operações da R-Tree

Nesta Seção apresentamos o algoritmo de inserção, remoção de objetos e de busca. Estes algoritmos são relevantes neste trabalho pois são a base para construir e manter uma árvore-R balanceada.

2.4.1 Inserção

O algoritmo de inserção tem como objetivo realizar alocações de novos objetos espaciais nos nós folhas da árvore R-Tree, buscando minimizar a área total de cada nó e assim reduzir a possibilidade de interseção com uma MBR. Na R-Tree as novas entradas na árvore são inseridas nas folhas, e os nós que ultrapassam a quantidade máxima definida por M são particionadas e propagadas para os pais. Os seguintes passos são referentes ao algoritmo de inserção da árvore R-Tree:

1. Para inserir uma nova entrada E em uma árvore R-Tree, escolhe-se a folha L onde E será inserido;

2. Se for verificado que em L há espaço para a nova inserção então insira E, caso contrário, divide o nó com o algoritmo de *split* para obter um novo L e LL contendo E e as antigas entradas de L;

3. Após inserido o elemento, a árvore é ajustada com o L ou L e LL (se houver uma divisão de nós).

4. Caso chegue em um ponto em que a divisão de nós provocou uma divisão na raiz, deve-se criar uma outra raiz tendo como filho os dois novos nós gerados.

2.4.2 Consulta

Consultas espaciais respondem por buscas de localizações específicas, áreas, intersecção de áreas, entre outros parâmetros. O seu principal objetivo é retornar uma informação ou dado de forma rápida, até mesmo com vários parâmetros de entrada. Em sistemas que trabalham com informações multidimensionais, a característica principal é responder a consultas espaciais. Entre essas consultas, as principais são: consulta por apontamento, que é a busca por um determinado ponto no espaço e busca por região, mais conhecida por busca de janela (*Window Query*).

A R-Tree provê uma consulta semelhante a efetuada na B-Tree. A busca se inicializa no nó raiz e continua em direção aos nós folhas. A diferença se dá pelo fato de ser necessário pesquisar por um objeto em mais de uma sub-árvore, o que faz aumentar o custo de uma consulta. Isso ocorre devido às sobreposições das entradas dos nós. Os algoritmos de inserção e *split* são uns dos principais responsáveis em diminuir o custo de uma consulta provocados pelas sobreposições. Os seguintes passos são referentes ao algoritmo de consulta da árvore R-Tree:

Localiza todas entradas cujos MBRs intersecta com o intervalo da consulta espacial S, a partir da R-Tree com a raiz indicada por T:

1. Para T sendo um nó interno, percorra todas as entradas E verificando se suas entradas fazem intersecção com S. Em todas entradas que há sobreposição, ou seja, fazem intersecção, repita esse processo novamente atualizando T como sendo a subárvore.

2. 2. Quando T é uma folha, verifica se todas suas entradas fazem intersecção com S. Para quando as entradas fazem a intersecção, então componha o conjunto da resposta.

2.4.3 Remoção

Uma remoção pode ser necessária quando se deseja retirar um objeto da árvore R-Tree, por exemplo, uma região com grande incidência de doenças passa a ser anulada, com isso, faz se necessário remover os objetos que representavam essa incidência na região. Para a realização de uma remoção na árvore R-Tree, são feitas quatro operações básicas: (a) a

busca do nó folha; (b) a remoção do objeto; (c) propagação das mudanças; e (d) ajuste do nó raiz.

(a) a busca pelo objeto é feita similar ao algoritmo de consulta;

(b) quando encontrado o nó folha que possui a entrada para ser removida, efetua a remoção do objeto;

(c) inicia a propagação das mudanças até o nó raiz, se necessário são feitas as atualizações nas MBRs. Se após a remoção de uma entrada, o número de entradas é inferior a capacidade mínima definida por m, então o nó deve ser eliminado e suas entradas restantes reinseridas na árvore.

(d) por fim a raiz é ajustada se possuir somente uma entrada, o nó filho vira a nova raiz da estrutura.

2.5 Métodos de divisão de nós

O particionamento ou *split* é um algoritmo muito importante para a otimização da árvore R-Tree. Ele deve evitar que dois nós gerados sejam percorridos durante a realização de uma consulta. Um nó é ou não percorrido se sua MBR faz intersecção com um outro intervalo de consulta. Portanto, a área gerada de uma MBR dos nós deve ser a menor possível. A Figura 2.6 mostra dois tipos de MBRs geradas durante a divisão de um mesmo nó. O primeiro *split* realizado possui uma área morta grande e sem sobreposição, ou seja, uma área que não contém nenhum dado de importância, acarretando em um aumento de custo na realização de uma consulta. Já no segundo *split* aparece uma sobreposição, mas há um ganho de otimização da área morta [Filho, Traina e Junior 1999].

Há uma variedade de trabalhos na literatura com propostas de algoritmos para *splits* [Outsios Evangelos 2011] [Korotkov 2012] [Liu, Fang e Han 2009] [Sleit e Al-Nsour 2014] [Guttman 1984] [Zhang e Xia 2004]. Em muitos têm como objetivo diminuir sobreposições e áreas mortas entre os nós da árvore. A Figura 2.7 mostra um split realizado para objetos do tipo ponto, polígono e linha. A maioria dos autores trazem como resultado final, *splits* otimizados para realizar poucas sobreposições, conforme o ilustrado na Figura 2.7 (pontos e polígonos). Quando se aplicam à objetos do tipo ponto, a área de sobreposição é nula, já para polígonos a área de sobreposição aumenta. É comum que a maioria dos resultados sejam generalizados para objetos do tipo linha. Mas como pode

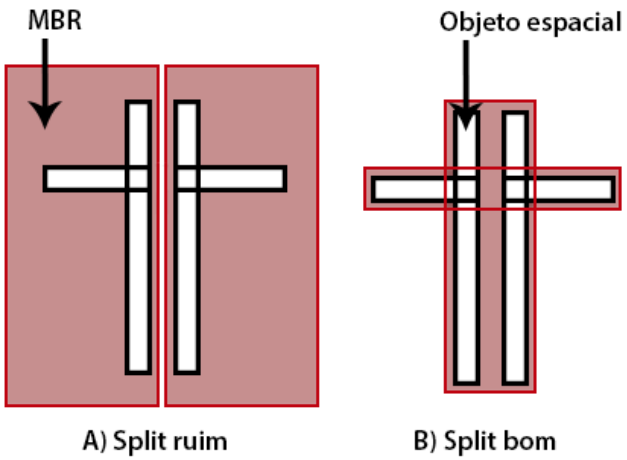

Figura 2.6: *Splits* distintos. Adaptado de [Guttman 1984].

ser visto na Figura 2.7, a sobreposição dos objetos com linhas são maiores, pois linhas são mais sinuosas e possuem tamanhos variados comparados a objetos do tipo ponto e polígono.

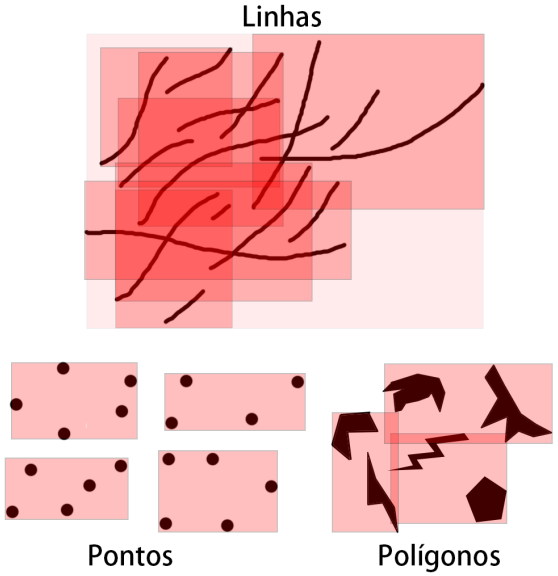

Figura 2.7: *Splits* de objetos distintos.

As Subseções abaixo apresentam, quatro algoritmos de *splits*:

1. Os algoritmos de split quadrático e linear pertencentes a R-Tree, que é a primeira estrutura proposta para indexação de objetos espaciais e a mais referenciada da literatura. Seus algoritmos são apresentados através de passos, como descrito no trabalho de Guttman;

2. O algoritmo de split da R*-Tree foi escolhido devido sua árvore ser uma das mais utilizadas em sistemas de bancos de dados e sistemas de informações geográficas;

3. O algoritmo de split da R0 que além de ser também utilizado em vários sistemas de bancos de dados é altamente referenciado na literatura.

2.5.1 Algoritmo Quadrático

O algoritmo quadrático busca encontrar uma combinação de entradas com áreas pequenas, mas não garante que essa área encontrada seja a melhor combinação. Seu custo é quadrático com relação a M e também linear no número de dimensões. Segundo [Beckmann et al. 1990], os problemas do algoritmo de particionamento de nó quadrático está na escolha de sementes pequenas, pelo fato das MBRs conterem tamanhos variados ou na ocorrência de grandes sobreposições.

O primeiro passo do algoritmo quadrático consiste em obter duas entradas, dentre as M+1, como sementes (primeira entrada) para compor os dois grupos que serão gerados depois do particionamento. Estas sementes correspondem às entradas que iriam gerar a maior área, quando comparadas duas a duas, caso fossem colocadas no mesmo nó. A seguir são descritos os passos do algoritmo de *split* quadrático:

Algoritmo de *split* quadrático

1. Divide o grupo de entradas $M + 1$ em dois grupos.

2. Execute o *PickSeeds* para escolher as duas primeiras entradas incluindo cada uma em um grupo;

3. Se todas as entradas já foram inseridas, encerre. Se um dos grupos possui poucas entradas, ao ponto de ser inferior a m de entradas, insira todas restantes e encerre;

4. Execute *PickNext* para obter a próxima entrada a ser inserida. Insira esta entrada no grupo que cujo MBR total terá que ser menos expandido após a inserção. Na ocorrência de empate, a inserção é feita naquele de menor área total. Depois, naquele com menor número de entradas e por fim, no que apresenta os dois casos;

5. Retorne ao passo 2.

Algoritmo *PickSeeds* do quadrático

1. Selecione duas entradas para serem as primeiras de cada grupo;
2. Para cada par de entradas, E_1 e E_2 , componha um MBR J que engloba E_1.I e E_2.I. Calcule: d = área(J) - área(E_1.I) - área(E_2.I);
3. Escolha o nó que apresenta o maior valor de d

Algoritmo *PickNext* do quadrático

1. Escolhe uma entrada, dentre as que restam, para ser testada em um grupo;
2. Para cada entrada E que ainda não foi inserida em um dos grupos, calcule d_1 = aumento necessário à área do MBR total do primeiro grupo para incluir E.I. Considerando, agora, o segundo grupo, calcule d_2 da mesma forma;
3. Escolha qualquer entrada que apresente a máxima diferença entre d_1 e d_2.

2.5.2 Algoritmo Linear

Segundo [Yannis et al. 2006], o algoritmo linear é o mais eficaz em tempo para realizar o *split* de nós cheios comparados ao quadrático e exponencial de Guttman mas, no entanto, não consegue determinar uma divisão ótima. A complexidade do algoritmo linear é $O(M)$ quanto ao número de dimensões e sua estrutura é similar ao algoritmo quadrático, mas difere-se em sua simplicidade de escolha das sementes dos nós.

Apesar da simplicidade das decisões do algoritmo linear, ele também não garante encontrar a menor MBR para ambos os nós. A escolha das sementes é realizada verificando a distância normalizada entre o mais alto lado inferior e o mais baixo lado superior dos MBRs de cada uma das dimensões, como mostra a Figura 2.8. O algoritmo primeiro busca pelo objeto com seu lado superior mais próximo do eixo Y e, posteriormente, encontra o objeto que possui a parte inferior mais próxima do eixo Y para definir a semente. Diferente do algoritmo quadrático, no algoritmo linear as entradas restantes são adicionadas ao acaso, escolhendo qualquer uma restante.

O algoritmo linear é quase idêntico ao algoritmo quadrático, mas com o algoritmo *PickSeeds* diferente. Além disto, o algoritmo para obter as próximas entradas,

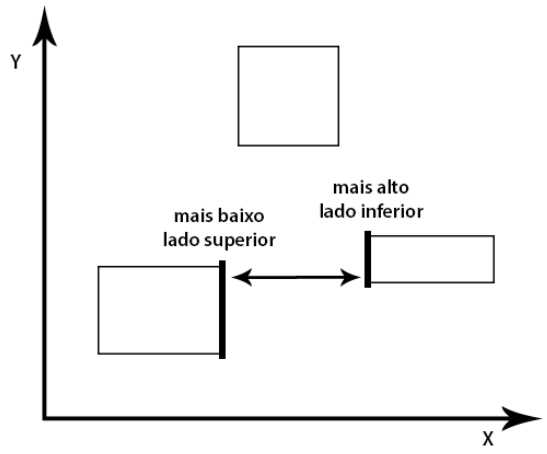

Figura 2.8: Normalização do algoritmo Linear.

escolhe qualquer entrada dentre as restantes. O algoritmo *PickSeeds* do linear é descrito a seguir:

Algoritmo *PickSeeds* do Linear

1. Seleciona duas entradas para serem as primeiras de cada grupo, encontre a entrada cujo MBR tem o lado inferior mais alto e a que tem o lado superior mais baixo, conforme a Figura 2.8 mostra. Armazene a distância entre eles;

2. Normalize as distâncias dividindo pelo tamanho de todo o conjunto ao longo da dimensão correspondente;

3. Escolha o par com a maior distância normalizada, dentre todas as dimensões.

2.5.3 Algoritmo da R*

A R* usa o seguinte método para encontrar bons *splits*. Ao longo de cada eixo, as entradas são primeiro ordenadas pelo valor mais baixo, então classificada pelo valor superior dos retângulos. Para cada tipo são determinadas $M-2+2m$ distribuições das entradas $M+1$ em dois grupos, em que a distribuição do k-ésimo ($k = 1$, $(M - 2m + 2)$) é descrito como se segue.

O primeiro grupo contém a primeira $(M - 1) + K$ entradas, o segundo grupo contém as entradas remanescentes. Para cada distribuição os melhores valores são determinados. Dependendo desses valores a distribuição final das entradas é determinada. Os

três valores melhores e de diferentes abordagens combinações são testados experimentalmente.

1. **Valor da área**: área [bb (primeiro grupo)] + área [bb (segundo grupo)];
2. **Valor da margem**: a margem [bb (primeiro grupo)] + margem [bb (segundo grupo)];
3. **Valor sobreposto**: área [bb (primeiro grupo) ∩ bb (segundo grupo)].

Aqui bb denota a caixa delimitadora de um conjunto de retângulos. Os possíveis métodos de processamento são determinar:

- o mínimo sobre um eixo ou um tipo;
- o mínimo da soma dos melhores valores sobre um eixo ou uma espécie;
- o mínimo global.

Os valores obtidos podem ser utilizados para determinar o eixo de divisão, ou a distribuição final (em um eixo de divisão escolhido) do melhor desempenho global.

2.5.4 Algoritmo da R0

Zhang e Xia, em seu trabalho apresenta a R0-Tree, uma melhora para a R*-Tree, modificando-a e apresentando a ideia do uso de oversize shelves (prateleiras de grande dimensão) para armazenar objetos outlier (isolado). Esses oversize buscam reduzir a extensão espacial dos MBRs dos nós de índice, mantendo nos espaços vazios de nível superior nodos esses objetos que possuem grande extensão espacial ou que são espacialmente isolado. Os efeitos são semelhantes aos de reinserção forçada, isto é, que a utilização de nó é aumentada e a extensão espacial de MBR é reduzida. A análise de desempenho que Zhang e Xia presente sugere que a R0 supera o R*-tree em até 48% em vários tipos diferentes de consultas.

2.6 Extensões da R-Tree

O estudo realizado por Gaede and Guenther [Gaede e Gunther 1998] mostra uma vasta lista de citações relacionadas a estruturas de indexação espacial por árvores-R. Ao longo

dos anos, várias extensões da R-Tree foram criadas com o objetivo de melhorar seu desempenho e estruturas com novas adaptações foram projetadas para um domínio de aplicações específicas. É o caso das aplicações de sistemas de informações geográficas e os bancos de dados.

Com base nessa variedade de extensões, essa Subseção não busca apresentar todas as variantes da R-Tree e suas vantagens, visto que a quantidade de variantes é muito grande. Portanto, são apresentadas as características específicas da R*-Tree e da R0-Tree, que são as estruturas mais reconhecidas e citadas por vários autores na literatura, além de serem aplicadas em uma grande quantidade de aplicações geográficas e bancos de dados espaciais [Jaluta 2014] [Oliveira et al. 2011].

A R*-Tree foi desenvolvida em 1990 com o objetivo de melhorar o desempenho da R-Tree [Beckmann et al. 1990] e, segundo as análises realizadas por seus projetistas, possui um melhor desempenho para manipular objetos multidimensionais do que a R-Tree de Guttman.

Enquanto a R-Tree baseia-se na minimização das áreas de cada MBR, a R*-Tree, além de examinar esse critério, procura examinar vários outros critérios para que se tenha um melhor desempenho na realização das consultas. Uma mudança substancial da R* é o algoritmo de *split* proposto, que engloba as melhorias descritas abaixo:

- **Minimização da área coberta por um MBR.** Esse critério busca minimizar a área morta existente na área coberta pelas MBR. Esse também é um critério examinado pela R-Tree;
- **Minimização da sobreposição entre as MBRs.** Diminuir as sobreposições para não percorrer mais de um caminho na árvore quando se realiza uma consulta;
- **Minimização das margens das MBRs (perímetro).** Visa dar formas mais quadradas para as MBRs e assim melhorar o desempenho da consulta. Além disso, MBRs com formas mais quadradas tendem a ter um novo englobamento por outra MBR mais facilmente, o que indiretamente faz a área total do nó ser otimizada;
- **Maximização da utilização do armazenamento.** Busca utilizar o máximo do armazenamento. Quando a utilização é baixa, mais nós tendem a ser chamados durante o processamento das consultas;

A R0-Tree é uma outra estrutura para indexar objetos espaciais assim como a R-Tree, no entanto, esta veio como variante da R*-Tree. Os seus criadores Donghui Zhang e Tian Xia observaram e estudaram a R*-Tree e chegaram a conclusão que R*-Tree poderia ser melhorada em vários aspectos, desde sua estrutura até suas operações internas [Xia e Zhang 2005]. Com isso, os autores propuseram uma nova estrutura chamada de R0-Tree com as seguintes contribuições em seu trabalho:

1. Definindo matematicamente o ganho e a perda de alguns objetos em uma MBR. Esta métrica integra as seguintes preocupações: a área de um MBR deve ser pequena e a forma de um MBR deve ser próxima à forma de um quadrado. Algumas métricas relacionadas também são definidas: a qualidade de um MBR e a perda de expansão um MBR [Zhang e Xia 2004].

2. A ideia de armazenar objetos discrepantes nos índices dos nós para melhorar o desempenho da consulta e que se fazem necessários para os algoritmos de inserção / remoção. Descrição de métodos para lidar com estouro de páginas positivo e negativo, o que ajusta dinamicamente a árvore de tal modo que os objetos discrepantes possam ser promovidos ou rebaixados sistematicamente, enquanto a árvore ainda mantém um fan-out mínimo garantido. Uma vez que a estrutura é construída, as mudanças se fazem necessárias para os algoritmos de consulta.

3. Uma comparação extensa e experimental em datasets[1] reais entre a R*-Tree e a R0-Tree por cinco consultas espaciais amplamente utilizadas;

Após apresentar as propostas e os resultado obtidos, os autores concluem que a R0 possui um custo de construção equivalente a R* mas, no entanto, a consulta por janela é até 40% melhor e, para consultas de vizinhos mais próximos a R0, é 51% melhor. Ainda [Xia e Zhang 2005], afirmam que a R0 pode ser adotada para bancos de dados e aplicações comerciais facilmente.

[1] O termo *dataset* foi empregado no texto para se referir a um conjunto de dados espaciais, como por exemplo um arquivo shapefile. Do Inglês data set.

2.7 Considerações finais

Este trabalho irá avaliar quatro algoritmos de *split*: quadrático, linear, R0 e R*. Como estrutura base é utilizada uma implementação da R-Tree sem o algoritmo de *outlier* da R0. O detalhamento da metodologia de comparação e de testes estão descritos no próximo Capítulo.

3 Trabalhos Relacionados

3.1 Introdução

Para condução deste trabalho foram elencados trabalhos relacionados à avaliação de *split* em índices multidimensionais e métodos de *splits* para índices multidimensionais. A pesquisa foi realizada utilizando-se bases de dados para trabalhos científicos. As bases utilizadas para este trabalho foram: Periódicos da Capes[1], SciELO[2], Google Acadêmico[3], IEEE Xplore[4], além de Revistas e Livros.

Foram utilizados os seguintes termos como palavras chaves: algoritmos de *split* em R-Tree, métodos de *split* em R-Tree, divisão de nós em R-Tree. Para filtrar os trabalhos encontrados foi realizado um mapeamento dos trabalhos resultantes afim de encontrar os que relatavam sobre propôr novos métodos de *split* ou realizar alguma avaliação de estruturas multidimensionais.

Com isso, foram definidos os seguintes trabalhos relacionados:The R*-tree: an efficient and robust access method for points and rectangles [Beckmann et al. 1990], que é apresentado na Seção 3.1, A novel improvement to the R*-tree spatial index using gain/loss metrics [Zhang e Xia 2004] na Seção 3.2 e Corner-based splitting: An improved node splitting algorithm for R-tree [Sleit e Al-Nsour 2014] apresentado na Seção 3.3.

3.2 The R*-tree: an efficient and robust access method for points and rectangles

Este é o trabalho ao qual Beckmann et al. propõem a árvore R*, juntamente como um novo método de *split* [Beckmann et al. 1990]. Beckmann et al. propõe a R*-Tree e seus principais algoritmos. Os autores apresentam pontos falhos que prejudicam o desempenho

[1]CAPES - www.periodicos.capes.gov.br/
[2]SciELO - www.scielo.org/
[3]Google Acadêmico - scholar.google.com.br
[4]IEEE Xplore - http://ieeexplore.ieee.org/

da R-Tree original proposta por Guttman [Guttman 1984].

A R*-Tree incorpora uma otimização combinando área, margem e as sobreposições de cada MBR colocando em um nó diretório. Em uma comparação de desempenho exaustiva utilizando a plataforma de teste padronizada proposta pelos autores, descobriram-se que o R*-Tree é superior as variantes da R-Tree existentes. O mesmo para os algoritmos de *split* linear e quadrático de Guttman e a variante da R-Tree de Greene. Esta superioridade da R*-Tree vale para vários tipos de consultas e operações, até mesmo para a quantidade de sobreposição em objetos do tipo retângulo e ponto. A R*-Tree suporta pontos e dados espaciais ao mesmo tempo e seu custo de implementação é ligeiramente maior do que a de outros árvores R.

Na R*-Tree foram inseridos novos conceitos baseados na redução da área, margem e sobreposição de diretórios que torna-a a robusta para distribuições de dados complexas. Além disso devido ao conceito de reinserção forçada, *splits* podem ser prevenidos enquanto a estrutura é reorganizada de forma dinâmica [Beckmann et al. 1990].

O ponto principal deste trabalho é a parte em que os autores realizam os seus experimentos comparando com a R-Tree original de Guttmann [Guttman 1984]. Beckmann et al. realizam seus experimentos com objetos do tipo ponto e polígono.

Em nosso trabalho, expandimos a avaliação feita pelos autores, comparando o algoritmo de *split* da R*-Tree com novos algoritmos propostos recentemente, e ainda avaliando-o com objetos espaciais do tipo linha, algo não realizado na época da publicação.

3.3 A novel improvement to the R*-tree spatial index using gain/loss metrics

Zhang e Xia introduz modificações na R* e dentre elas as métricas de ganho e perda. Com a evolução da R* a nova estrutura resultante passou a ser chamada de R0-Tree [Zhang e Xia 2004].

Os autores definiram matematicamente a qualidade de um nó e o ganho para remover alguns objetos de uma MBR. Esta métrica integra a seguinte preocupação: a área do MBR a ser removida deve ser pequena e próxima ao canto da MBR em que está contida. Com base nas definições propostas, propuseram algoritmos para localizar

conjuntos de objetos a serem removidos obtendo o maior ganho possível. Esses algoritmos mostraram ter melhores desempenhos do que os da R*-Tree original. Melhoraram a R*-Tree de várias outras maneiras. A versão melhorada da R*-Tree foi experimentalmente comparada com a original R*-Tree por meio de reais conjuntos de dados. A R0-Tree obteve nos experimentos uma melhoria de até 20% no desempenho das consultas [Zhang e Xia 2004].

Nos experimentos de seu trabalho, Zhang e Xia usam experimentos *datasets* contendo pontos e linhas, verificando o tempo de consulta, tamanho das páginas de disco, entradas e saídas do disco e o armazenamento em relação ao tamanho da consulta. Todos esses experimentos foram comparados com a R*-Tree original. Em nosso trabalho além de verificar o armazenamento, tempo de construção e tempo de uma consulta para a R0-Tree e R*-Tree, também buscamos mostrar uma comparação com a R-Tree de Guttaman afins de apresentar os algoritmos de *split* que produzem maior quantidade de nós, sobreposições e a média de preenchimento dos nós em objetos do tipo linha, ponto e polígono.

3.4 Corner-based splitting: An improved node splitting algorithm for R-tree

Sleit et al. não propõe uma nova estrutura de indexação multidimensional mas sim um novo método de *split* para a R-Tree [Sleit e Al-Nsour 2014]. Em seu trabalho o autor compara seu novo método de *split* com o algoritmo quadrático de Guttman [Guttman 1984] e o novo *split* linear proposto por Chuan Heng Ang [Ang e Tan 1997]. Os experimentos são definidos para encontrar a qualidade dos *splits* através das sobreposições e da quantidade de nós que foram produzidos.

O método proposto trabalha para dados dimensionais mais elevados. Os experimentos para testar o algoritmo de Sleit et al. foram conduzidos utilizando *datasets* para dados sintéticos e reais, ao mudar o tamanho da página em disco e o valor de preenchimento mínimo os experimentos demonstraram um desempenho superior favorável ao algoritmo de Sleit et al., ou seja, o algoritmo de *split* quadrático da R-Tree e o algoritmo de *split* New Linear foi superado nos experimentos no momento da criação do índice, sobreposição e área de cobertura total [Sleit e Al-Nsour 2014].

O trabalho aqui proposto em relação ao do Sleit et al. não somente avalia so-

breposições e o número de nós que produzem, como também identifica o melhor algoritmo para realizar consultas e o que melhor sobressai para objetos do tipo linha.

O trabalho se Sleit et al. apesar de ser publicado recentemente, o autor não compara seu algoritmo de *split* a R* e a R0, sabendo que esses dois algoritmos são bastantes referenciado na literatura. Perante isso, o nosso trabalho possui mais um aspecto favorável, pois nos experimentos estão incluídos os algoritmos da R* e da R0 que são bastantes mencionados na literatura.

4 Metodologia de Avaliação

4.1 Introdução

Este capítulo apresenta os detalhes de implementação dos algoritmos de *split* que fundamentaram o presente trabalho, a metodologia de avaliação, a validação da implementação, a comparação dos algoritmos e da estrutura de dados gerada por cada um deles. As seções a seguir apresentam os passos para a implementação dos algoritmos de split, bem como a aplicação utilizada para a realização dos experimentos.

Na Seção 4.2 é apresentada a descrição dos algoritmos usados nos experimentos, indicando o que foi necessário implementar e o que já se encontrava disponível. A Seção 4.3 mostra os passos que foram realizados para validar a implementação e a execução dos algoritmos.

Após os algoritmos serem validos, os mesmo tiveram que ser integrados na R-Tree, que é apresentado na Seção 4.4. A Seção 4.5 caracteriza os *datasets* que foram definidos para a realização dos testes, incluindo os tipos de objetos que foram utilizados.

Na Seção 4.6 é apresentada o ambiente e a máquina que foram utilizados para a realização de todos os testes e na Seção 4.7 todas as métricas que foram utilizadas para avaliar o desempenho de cada algoritmo de *split*. Por fim, a Seção 4.8 é apresentada a aplicação console desenvolvida para executar comandos e realizar os experimentos.

4.2 Algoritmos Usados nos Experimentos

Os experimentos foram realizados com os seguintes algoritmos de *split*: linear, quadrático, algoritmo de *split* da R0 e da R*. Todos esses algoritmos foram usados na construção de árvores R-Tree com objetos do tipo ponto, polígono e linha. No entanto, os experimentos para as linhas foram realizados com maior abrangência, pelo fato de serem o escopo desse trabalho.

Os algoritmos linear e quadrático foram implementados seguindo a descrição

no artigo de Guttman [Guttman 1984]. Os outros dois algoritmos, *split* da R0 e R* encontravam-se implementados e já validados pelo grupo de pesquisa em sistemas distribuídos da Regional Jataí da UFG, implementação essa usada na publicação de outros trabalhos científicos como [Oliveira, Costa e Rodrigues 2015].

A implementação dos algoritmos de *split* Linear e Quadrático propostos por Guttman [Guttman 1984], foi efetuada na linguagem C/C++, utilizando o ambiente de desenvolvimento NetBeans [1].

Os algoritmos Linear e Quadrático foram implementados seguindo as descrições do artigo de Guttman [Guttman 1984]. O código foi modularizado em três funções: *Split*, *PickSeeds* e *PickNext*, conforme o autor propôs em seu trabalho.

4.3 Validação da Implementação dos Algoritmos

Após a implementação, os algoritmos foram validados para garantir sua corretude no processo de *split* utilizando os exemplos apresentados pelo próprio autor e trabalhos relacionados como [Guttman 1984], [Sleit e Al-Nsour 2014], e seis casos de *split* foram utilizados para validar os três algoritmos implementados.

Com o uso da ferramenta PhotoShop, as imagens dos artigos foram rasterizadas para extrair cada coordenada dos retângulos. Assim, os objetos ficaram representados com suas coordenadas e aptos para os algoritmos processarem o *split*. Nos algoritmos foram adicionadas funções auxiliares que foram capazes de imprimir os *splits* gerados em formato geojson. Com esse formato foi possível abrir os arquivos com os resultados na ferramenta Quantum GIS (QGIS)[2] e comparar com os resultados apresentados nos artigos. As Figuras 4.1 e 4.2 mostra o exemplo apresentado no artigo de Sleit et al. e o resultado do *split* após a validação [Sleit e Al-Nsour 2014].

A partir do conjunto de objetos em a) da Figura 4.1 foi realizado por Sleit et al. dois *splits*, sendo b) e c). Sleit et al. em seu trabalho mostrou como era realizado o *split* de um nó cheio a) pelo algoritmo do quadrático, afim de comparar com o novo algoritmo que é apresentado em seu trabalho. O mesmo conjunto empregado no trabalho de Sleit et al. para apresentar o *split* do quadrático é utilizado neste trabalho para mostrar como

[1] NetBeans - Ambiente de desenvolvimento integrado (https://netbeans.org/)
[2] QGIS - http://www.qgis.org/en/site/

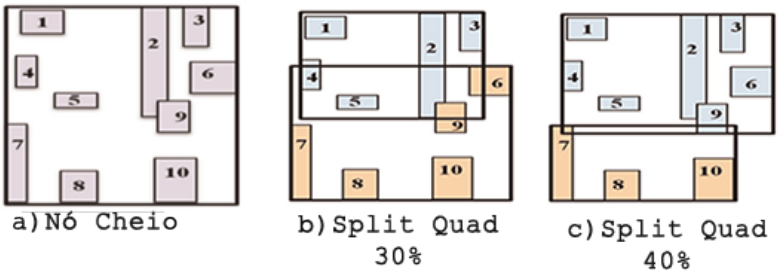

Figura 4.1: Passos da realização de *split* em um nó cheio utilizando o algoritmo quadrático para valores de M com 30% e 40% apresentado no artigo de Sleit et al.

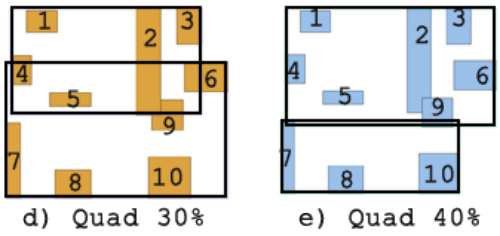

Figura 4.2: Resultado do novo *split* realizado no exemplo de Sleit et al. para fins de validação da implementação do algoritmo quadrático.

foi realizado a validação e corretude das implementações. O resultado para comparação pode ser visto na Figura 4.2, na qual os *splits* em d) e f) são os resultados dos novos *splits* gerados após a implementação dos algoritmos.

4.4 Integração e Validação dos Algoritmos na R-Tree

Com os algoritmos de *split* implementados e validados, os mesmos foram integrados no código da implementação da R-Tree já existente, conforme descrito na Seção 4.2. Para realizar a integração foi necessário refatorar todo o código corrigindo os nomes das funções, parâmentros e adaptando o *split* para dois cenários distintos em que a R-Tree exigia, *splits* para nós diretórios e *splits* para nós folhas.

Para validar a integração do código, os *datasets* utilizados nos trabalhos relacionados foram obtidos por meio do site Census TIGER/Line[3] e outras fontes na internet, conforme indicação nos artigos, e foram realizados os mesmos testes para fins de comparação quanto a corretude da implementação dos três algoritmos de *split*.

[3]Census TIGER/Line - https://www.census.gov/cgi-bin/geo/shapefiles/index.php

4.5 Conjuntos de dados (*Datasets*)

Para realizar os experimentos foi escolhido um conjunto de datasets reais, obtidos no site do IBGE[4] e do Laboratório de Processamento de Imagens e Geoprocessamento (LAPIG)[5] da UFG. Os datasets empregados e suas características estão descritos na Tabela 4.1. Foram escolhidos *datasets* com objetos do tipo ponto, polígono e linhas. A cardinalidade é apresentada, com finalidade de mostrar a quantidade de objetos contidos em cada um dos *datasets*. A coluna Tamanho SHP (Shape - Forma), refere-se ao tamanho em MB (MegaBytes) de cada um dos arquivos que foram utilizados nos experimentos.

Tabela 4.1: Datasets utilizados nos testes.

Nome	Sigla	Tipo	Cardinalidade	Tamanho SHP (MB)
Trechos de Energia	TE	Linhas	784	<1,0
Hidrografia	H	Linhas	245.912	112,1
Malha Viária	MV	Linhas	51.593	15,2
Terras Indígenas	TI	Polígonos	443	6,2
Unidades de Uso Sustentável	US	Polígonos	632	10,8
Portos	P	Pontos	120	<1,0
Extração Mineral	EM	Pontos	2.957	<1.0

Todos os *datasets* descritos na Tabela 4.1 representam os dados reais de todo território brasileiro. Em específico, o *dataset* de Hidrografia representa apenas o território do estado de Goiás.

4.6 Infraestrutura Física

Os experimentos executados foram realizados no LARC (Laboratório de Arquitetura e Redes de Computadores) do Curso de Bacharelado em Ciências da Computação da Universidade Federal de Goiás na Regional Jataí, unidade Jatobá.

A máquina utilizada no experimentos é do modelo D360 *Desktop* da Positivo Informática, com processador Intel Core i5 de terceira geração 3330 (3.0 GHz, 6 MB de Cache) com tecnologia Turbo Boost até 3.20 GHz e 4 núcleos de processamento. Placa mãe modelo POS-MIH61C; memória RAM DDR3 de 4 GB (1333 MHz, Non-ECC) com suporte ao modo Dual Channel. Disco rígido com armazenamento de 500 GB e 7.200 RPM, SATA II, 3,5".

[4]IBGE - www.ibge.org

[5]LAPIG - www.lapig.iesa.ufg.br/lapig/

Essa configuração permite que todos os *datasets* sejam carregados na própria memória sem a necessidade de criar páginas de disco. Em nossos experimentos não avaliamos E/S de disco por dois motivos.

1. A avaliação de E/S de disco exige a implementação de estruturas de *buffers* para preservar páginas mais usadas na memória, os quais fogem do escopo do nosso trabalho; e

2. Com o aumento da disponibilidade de memória nos computadores atuais, cada vez mais estas estruturas são mantidas em memória para acelerar a execução dos algoritmos de busca.

4.7 Métricas para Avaliação dos Algoritmos

Para as linhas foram realizados testes utilizando três *datasets* reais verificando o tempo de construção de índice da R-Tree, quantidade de nós (diretórios e folhas), sobreposição dos nós (diretórios e folhas), percentual de preenchimento dos nós (diretórios e folhas) e o tempo de execução das consultas no índice da R-Tree.

Para os pontos e polígonos foram realizados testes utilizando dois *datasets* reais verificando o tempo de construção do índice da R-Tree, quantidade de nós (diretórios e folhas), sobreposição dos nós (diretórios e folhas) e a percentual de preenchimento dos nós (diretórios e folhas).

Duas classes de métricas foram utilizadas: métricas para avaliar a estrutura final da R-Tree e métricas para avaliar a consulta na R-Tree.

4.7.1 Métricas para avaliação da construção da R-Tree

- **Indexados:** esse valor define o número de objetos que foram indexados e, portanto, foi usado para verificar se todos os objetos do *dataset* foram indexados corretamente;

- **Diretórios:** quantidade de nós diretórios que foram criados após a árvore ser construída. Árvores com um grande número de diretórios usam mais espaço de armazenamento;

- **Folhas:** quantidade de nós folhas que foram criadas, também impactam no armazentamento em memória/disco;

- **Preenchimento médio:** é o percentual médio de preenchimento dos nós. Um bom preenchimento dos nós geram árvores com altura menor e mais balanceadas, que consequentemente exigirão menos espaço de armazenamento. Essa métrica foi obtida usando a seguinte fórmula:

$$\frac{(\sum_{i=0}^{N} no_i.items)/N}{M}$$

  Ou seja, o somatório da quantidade de itens nos nós da árvore ($no_i.items$), dividido pela quantidade de nós da árvore (N), dividido pelo valor de M, que é a quantidade máxima de itens em um nó da árvore.

- **Sobreposição de diretórios:** sobreposição de nós diretórios. Um número grande de sobreposições faz com que as consultas tenham um custo alto, isso porque é necessário encontrar o objeto em mais de um ramo da árvore;

- **Sobreposição de nós folhas:** sobreposição de nós folhas;

- **Tempo de construção:** Tempo gasto para construção da árvore em milisegundos;

4.7.2 Métrica para avaliação de consultas na R-Tree

Para realizar as consultas foram criados arquivos com buscas de janelas que representavam 5%, 10%, 15% e 20% da área total do *dataset*. Por meio de um *script* desenvolvido na linguagem C, foram extraídos dos *datasets* as coordenadas que representavam a sua dimensão e, posteriormente, calculado regiões para busca em 5%, 10%, 15% e 20% de todo território do *dataset*. Com os arquivos contendo as regiões de buscas em torno de toda área do *dataset* foi possível realizar um conjunto de consultas para fins de analisar os seguintes itens:

- **Tempo de execução:** tempo gasto para a execução da consulta em microsegundos;

- **Geometrias verificadas:** quantidade de objetos que foram percorridos durante a consulta;

- **Nós verdadeiros e falsos:** refere-se aos nós diretórios e folhas que foram visitados corretamente, pois o elemento da busca estava contido no nó e os nós que foram visitados indevidamente, sem a presença do elemento da busca. ;
- **Resultado:** quantidade de objetos encontrado na consulta.

4.8 Aplicação de Experimento

Para a realização dos experimentos foi desenvolvida uma aplicação console na linguagem de programação C. A aplicação console recebe parâmetros de linha de comando e executa os algoritmos implementados, gerando árvores para os *datasets*. A sintaxe para execução da aplicação é:

./main file.shp M (-rgut -rlinear -r0 -rstar) -s -g

Sendo que cada argumento é descrito a seguir:

- *file.shp* - caminho do arquivo do dataset;
- *M* - quantidade máxima de objetos por nó, esse valor foi definido com base a divisão do tamanho da página de disco 4096 *bytes* pelo tamanho do nó da árvore 64 bytes, resultando assim o valor de M para 64;
- *-rgut* - define como algoritmo de split o quadrático;
- *-rlinear* - define como algoritmo de split o linear;
- *-r0* - define como algoritmo de split o da R0;
- *-rstar* - define como algoritmo de split o da R*;
- *-s* - arquivo de janela, contém entradas pré-determinadas para realizar buscas em todo *dataset*, conforme descrito na Seção 4.7.1;
- *-g* - opção para gerar arquivos no formato *.shp* contendo as representações dos novos *split* em vários níveis da árvore, para o processo de validação.

5 Resultados

5.1 Introdução

Neste capítulo é abordado e apresentado os resultados alcançados a partir dos experimentos. As Seções a seguir apresentam os o tempo de construção do índice R-Tree, quantidade de nós diretórios e folhas, sobreposições de nós diretórios e folhas, preenchimento médio dos nós e o tempo de execução de consultas na árvore para cada um dos objetos do tipo linha, ponto e polígono.

O Capítulo está organizado da seguinte forma: é apresentado primeiramente os resultados dos experimentos realizados para objetos do tipo linha na Seção 5.2, para objetos do tipo polígono na Seção 5.3 e na Seção 5.4 é apresentado para os objetos do tipo ponto.

5.2 Resultados para Objetos do Tipo Linha

5.2.1 Tempo de Construção do Índice R-Tree

A Figura 5.1, apresenta três gráficos com o tempo de construção das árvores R-Tree para os três *datasets* definidos para as linhas, e cada um dos algoritmos de *split*.

Em todos os três *datasets*, percebe-se que o algoritmo de *split* da R* possui o pior tempo para a construção do índice e o algoritmo linear o melhor tempo nos *datasets* de Hidrografia e Malha Viária.

No *dataset* Trecho de Energia, que é um *dataset* com poucos objetos, os algoritmos de *split* resultaram em um tempo similar para realizar a indexação. Já nos outros dois *datasets*, o de Hidrografia e o de Malha Viária, que possuem mais objetos é possível que o algoritmo de *split* da R* tende a ter o um maior tempo para a construção do índice quando se tem *datasets* com uma grande quantidade de objetos.

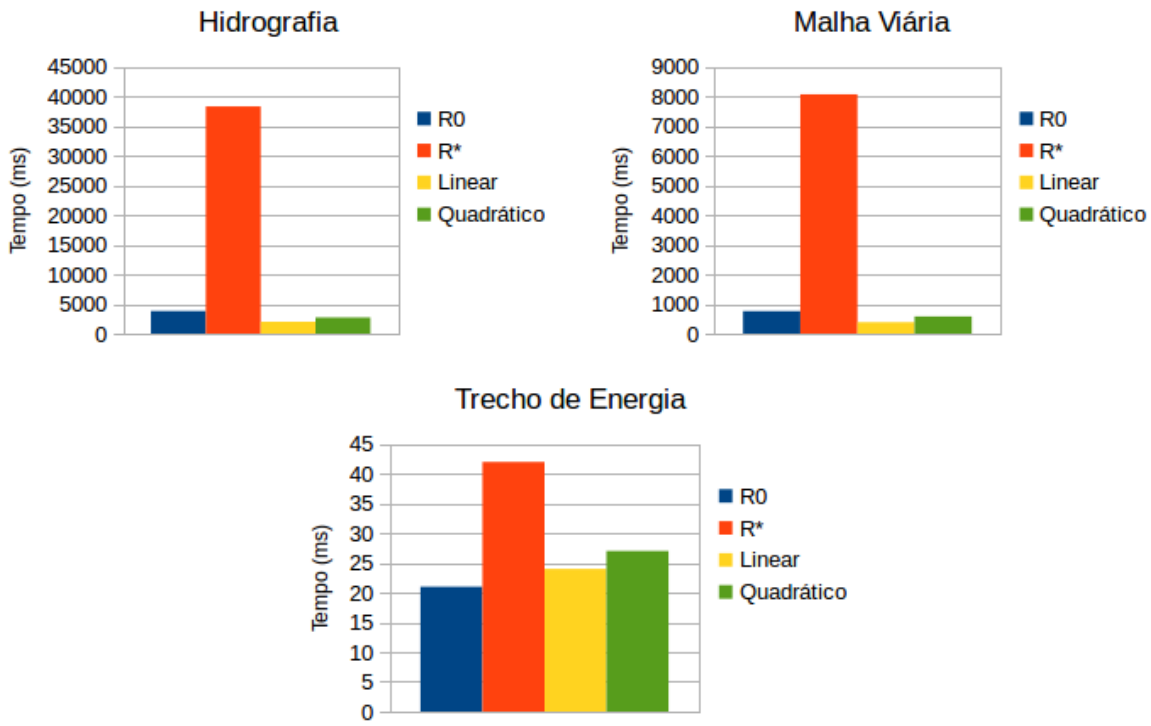

Figura 5.1: Gráficos com a representação do tempo de construção dos indíces da R-Tree para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

5.2.2 Nós Diretórios e Nós Folhas

A Figura 5.2 possui três gráficos para cada um dos *datasets* definidos para as linhas. Em cada um desses gráficos é apresentado a quantidade de nós diretórios que foram gerados durante a criação do índice.

No *dataset* Trecho de Energia os algoritmos produziram a mesma quantidade de nós diretórios, isso ocorreu pelo fato de ser um *dataset* com pouco objetos para indexar. A partir dos *datasets* de Hidrografia e Malha Viária é possível perceber a diferença entre os algoritmos e encontrar o que mais se sobressaiu.

Conforme aumenta a quantidade de objetos do *dataset* o algoritmo da R0 tende a produzir mais nós diretórios comparado aos outros algoritmos de *split*. O algoritmo da R* tende a produzir menos nós diretórios comparado com os demais.

A Figura 5.3 apresenta o cenário para os nós folhas. Em todos os três *datasets* o algoritmo quadrático foi o que mais produziu nós folhas, seguido pleo algoritmo linear. Para o *dataset* Trecho de Energia, o qual possui menos objetos para indexar, o algoritmo da R0 criou menos nós folhas porém, este comportamento não se mantém para os *datasets*

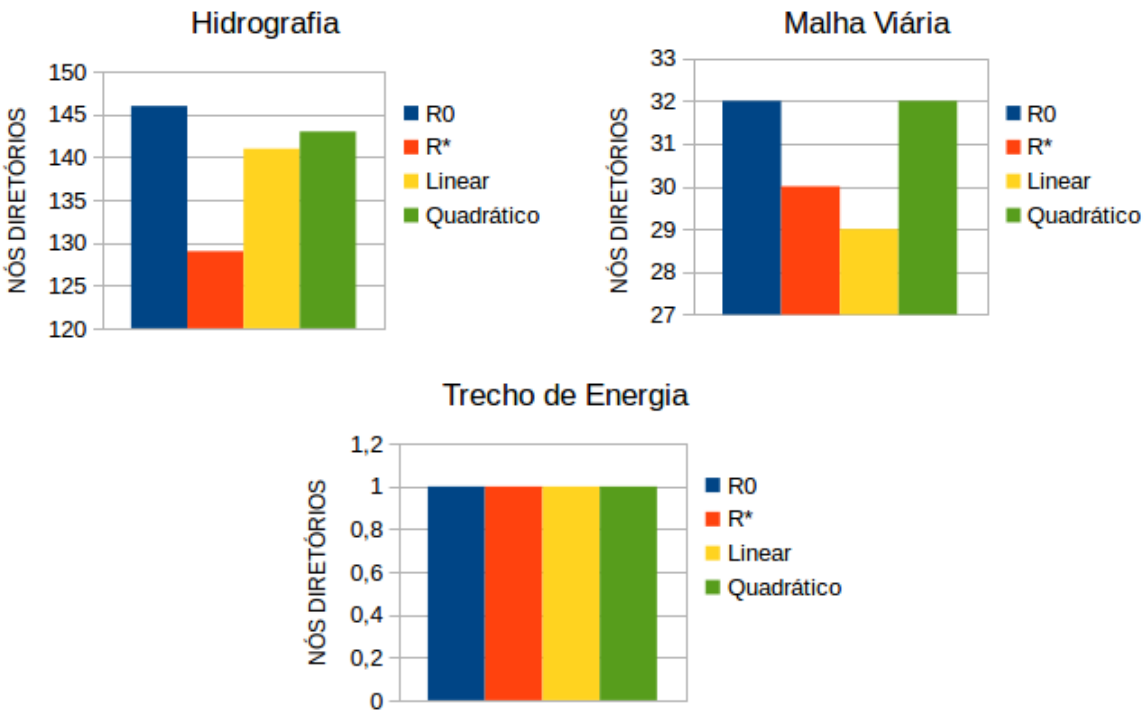

Figura 5.2: Gráficos com a representação da quantidade de nós diretórios para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

com mais objetos. O algoritmo da R* produz menos nós folhas para os *datasets* com mais objetos.

Quando a construção da árvore produz um grande número de nós folhas e de nós diretórios mais espaço de armazenamento é necessário. Para identificar quais algoritmos exigiram mais espaço de armazenamento foi produzido uma tabela para cada *dataset* contendo a soma dos nós diretórios e folhas produzidos para cada algoritmo de *split*.

A Tabela 5.1 apresenta o algoritmo R0 sendo o que menos exige espaço de armazenamento e o quadrático o que mais exige espaço de armazenamento, isso para o cenário utilizando *dataset* com poucos objetos.

Tabela 5.1: Quantidade total de nós para o *dataset* Trecho de Energia.

Algoritmo	Nós diretórios	Nós Folhas	Total
R0	1	16	17
R*	1	17	18
Linear	1	18	19
Quadrático	1	19	20

As Tabelas 5.2 e 5.3 que representa os *datasets* Hidrografia e Malha Viária,

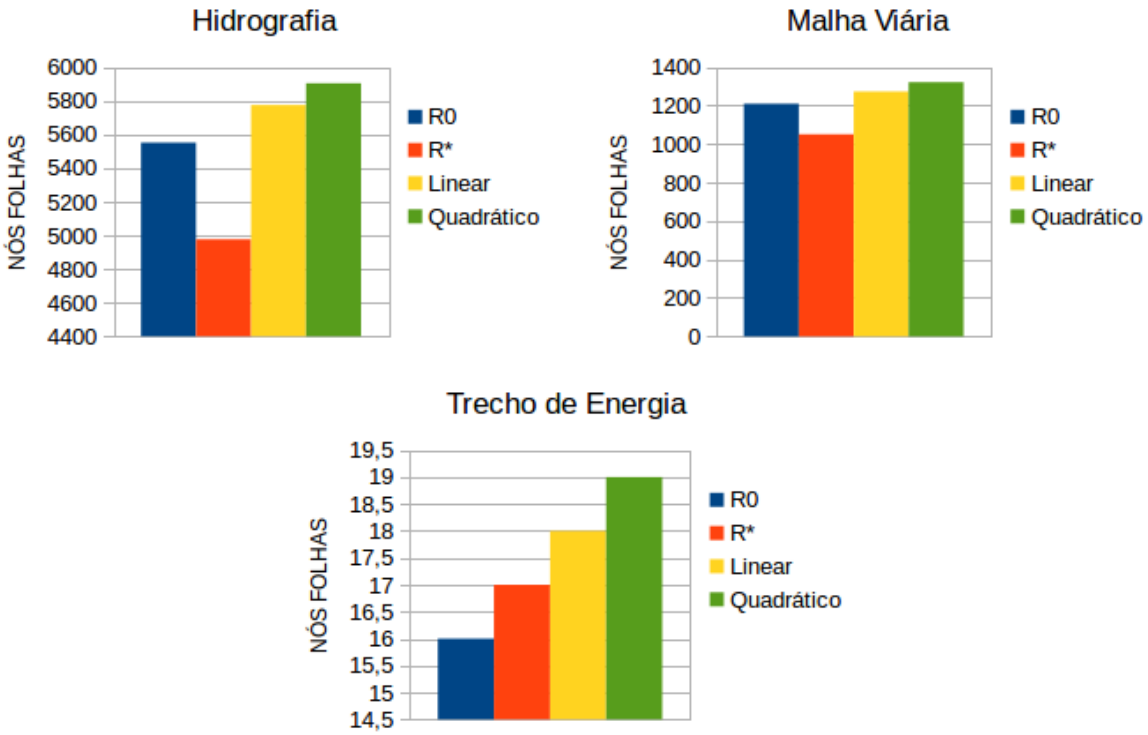

Figura 5.3: Gráficos com a quantidade de nós folhas para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

mostra o algoritmo R* sendo o algoritmo que menos exige espaço de armazenamento e o quadrático como o algoritmo que exige mais espaço de armazenamento.

Tabela 5.2: Quantidade total de nós para o *dataset* Hidrografia.

Algoritmo	Nós diretórios	Nós Folhas	Total
R*	129	4977	5106
R0	146	5551	5697
Linear	141	5775	5916
Quadrático	143	5904	6047

5.2.3 Sobreposição de Nós Diretórios e Nós Folhas

Um grande número de sobreposições aumenta cada vez mais o tempo de busca por um objeto na árvore, isso porque as sobreposições entre dois ou mais nós faz com que o algoritmo de busca tenha que percorrer todos os ramos/subárvores encontrar o elemento desejado.

Pelo fato das sobreposições influenciarem no desempenho de uma busca na árvore, avaliamos a sobreposição para cada um dos *datasets* e algoritmos. As Figuras

Tabela 5.3: Quantidade total de nós para o *dataset* Malha Viária.

Algoritmo	Nós diretórios	Nós Folhas	Total
R*	30	1051	1081
R0	32	1210	1242
Linear	29	1273	1302
Quadrático	32	1322	1354

5.4 e 5.5 mostram o comportamento de cada algoritmo de *split* em relação ao número de sobreposições que os mesmos produzem.

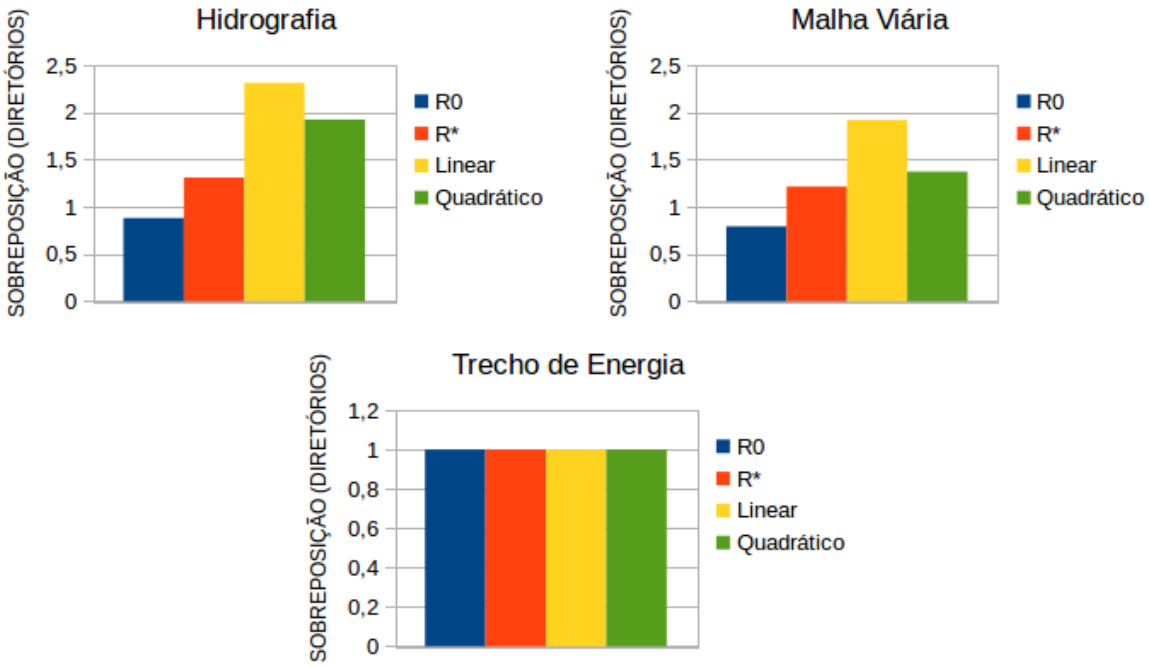

Figura 5.4: Gráficos com a quantidade de sobreposições em nós diretórios para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

Na Figura 5.4 o *dataset* Trecho de Energia teve apenas uma sobreposição de nó diretório em todos os algoritmos de *split*. Isso acontece pelo fato de conter poucos objetos no *dataset*, assim os algoritmos conseguiram adicionar todas as folhas em apenas um nó diretório. O algoritmo que sobressai gerando o maior número de sobreposições em nós diretórios é o linear, tanto no *dataset* de Hidrografia e Malha Viária. O algoritmo da R0 foi o que menos gerou sobreposições e assim se mostrou mais eficiente em relação aos demais algoritmos.

Enquanto a R* ficou apenas atrás da R0 em sua eficiência para sobreposições de nós diretórios, para sobreposições em nós folhas a Figura 5.5 mostra um cenário diferente. Nos dois *datasets* com mais objetos a R* teve o pior caso, sendo assim gerou um grande

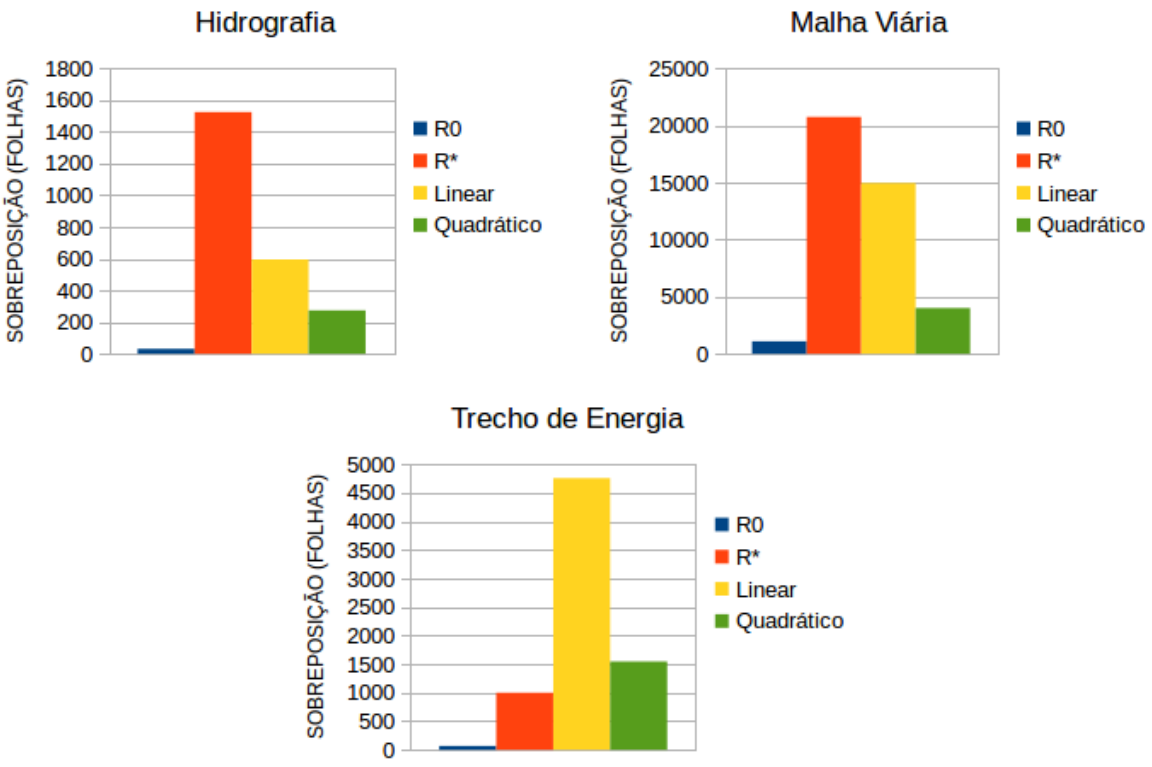

Figura 5.5: Gráficos com a quantidade de sobreposições em nós folhas para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

número de sobreposições comparados aos demais algoritmos. Da mesma forma para os diretórios o algoritmo da R0 também sobressaiu aos demais algoritmos produzindo uma menor área de sobreposição para os nós folhas. Mas quando os algoritmos são aplicados a *datasets* com menores quantidades de objetos, outro cenário pode ser visto, no *dataset* de Trecho de Energia a diferença dá se entre o algoritmo da R* e o linear, na qual o linear é o algoritmo que traz o maior número de sobreposições para os nós folhas.

5.2.4 Preenchimento Médio dos Nós Diretórios e Folhas

Quando os nós diretórios e os nós folhas obtêm uma margem de preenchimento de objetos próxima a 100% faz com que seja necessário criar menos nós diretórios e nós folhas. Isso impacta no espaço de armazenamento, pois quanto mais nós são criados, mais espaço de armazenamento é necessário.

A Figura 5.6 traz os resultados de cada algoritmo. Em todos os três *datasets* o algoritmo quadrático apresentou o pior caso para preenchimento, seguido do algoritmo linear. Nos *datasets* de Hidrografia e Malha Viária que possuem mais objetos, o algoritmo

da R* foi superior aos demais algoritmos e apresentou o melhor resultado, com preenchimento próximo a de 80%. O cenário so é diferente no *dataset* de Trecho de Energia, em que o algoritmo da R0 apresenta preenchimento melhor que o algoritmo da R*.

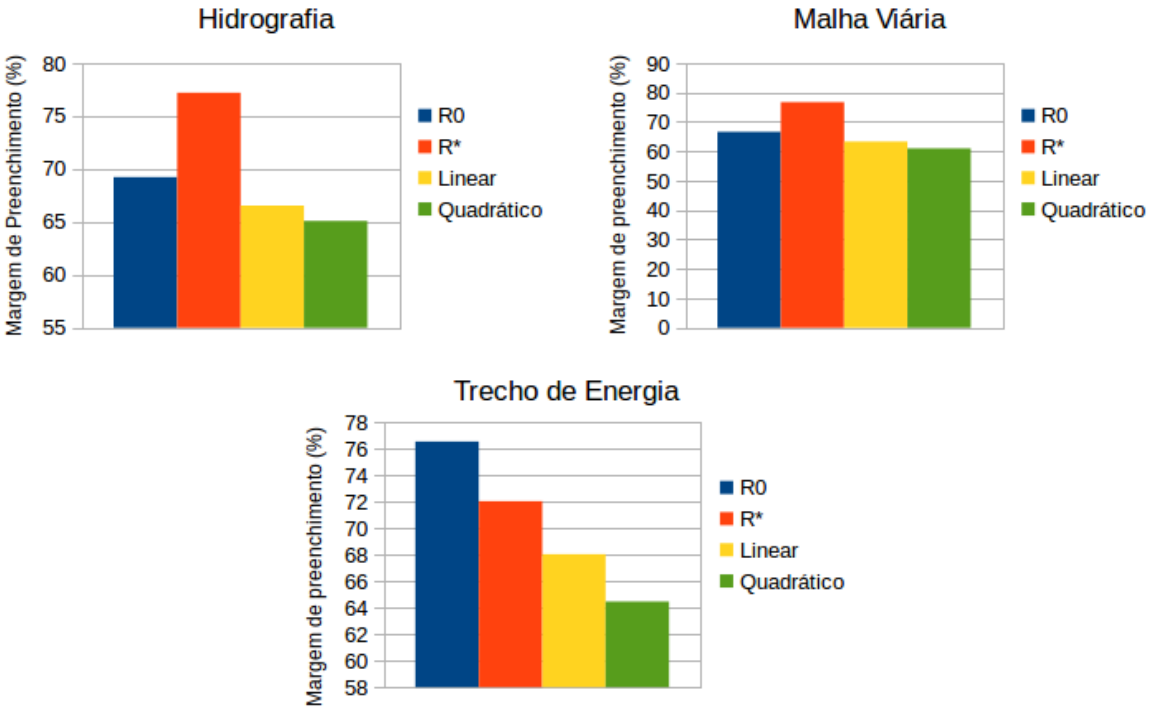

Figura 5.6: Gráficos com a representação da margem de preenchimento dos nós para os *datasets* de Hidrografia, Malha Viária e Trecho de Energia (Linhas).

5.2.5 Tempo de Execução de Consultas no Índice R-Tree

As consultas realizadas nos *datasets* foram feitas por buscas de regiões em cada *dataset* que representavam 5%, 10%, 15% e 20% da área total do *dataset*. As Figuras 5.7, 5.8 e 5.10 apresentam os tempos de execução acumulados para as buscas realizadas em cada um dos *datasets*.

Na Figura 5.7 o gráfico apresenta algoritmo linear como sendo o mais rápido para realizar buscas em todos tamanhos de região. A media que a região do *dataset* é definida para 15% ocorre uma oscilação do tempo da consulta em que o algoritmo linear torna se inferior ao quadrático.

A Figura 5.8 apresenta o gráfico de consultas por região no *dataset* de Malha Viária e nesse gráfico é possível verificar que quanto mais a região da consulta tende a ser menor o algoritmo linear tende a ter um alto tempo para realizar as buscas. Assim como

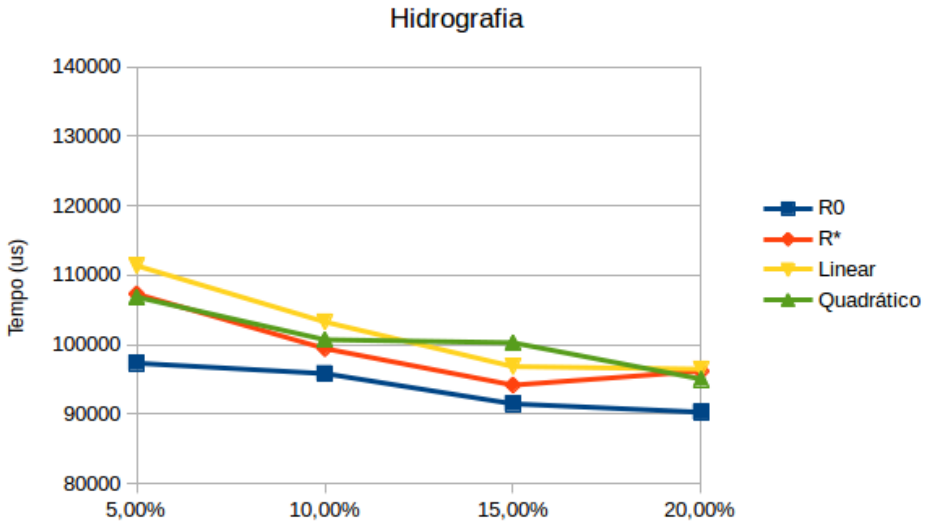

Figura 5.7: Gráfico com a representação de consultas por região no *dataset* de Hidrografia.

no *dataset* de Hidrografia o algoritmo da R0 se manteve sendo o algoritmo que menos tempo utiliza para realizar uma busca.

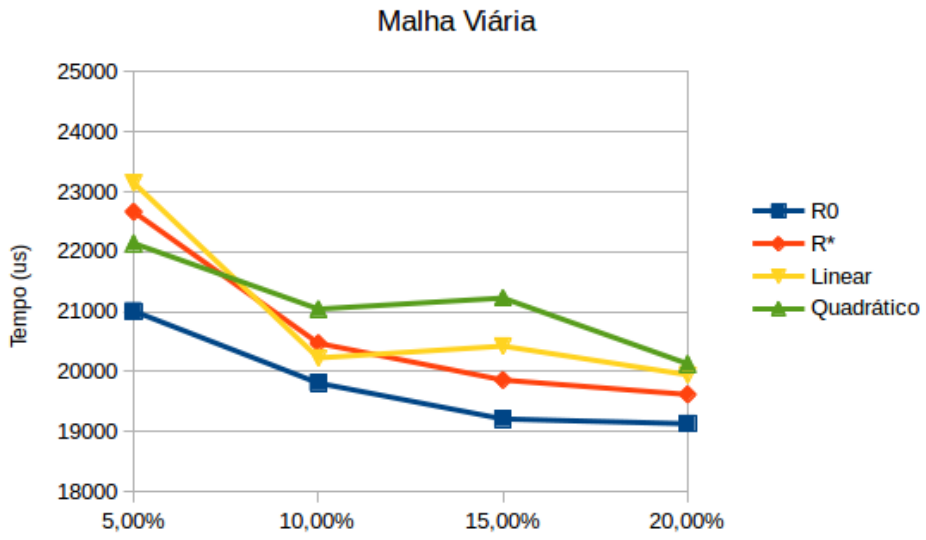

Figura 5.8: Gráfico com a representação de consultas por região no *dataset* de Malha Viária.

O *dataset* Trecho de Energia que é apresentado na Figura 5.10 possui um gráfico que o linear novamente sendo o pior algoritmo para realizar buscas quando a região de busca é aproximadamente 5%, a medida que essa região aumenta os tempos dos algoritmos oscilam entre si, isso porque o número de busca que é necessário realizar diminuem. Por fim o algoritmo da R0 aparece outra vez sendo o algoritmo que leva o menor tempo para realizar as buscas.

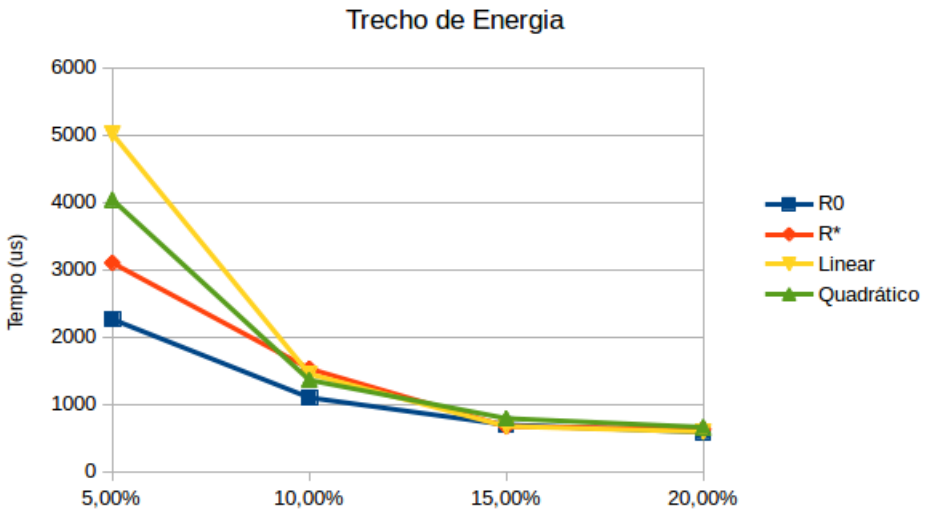

Figura 5.9: Gráfico com buscas.

Figura 5.10: Gráfico com a representação de consultas por região no *dataset* de Trecho de Energia.

Como o tempo de busca varia de um teste para o outro, uma média foi realizada com cada algoritmo levando em consideração os quatro tipo de busca e isso pode ser visto na Tabela 5.4.

Tabela 5.4: Média de consultas por regiões nos *datasets* de Hidrografia, Malha Viária e Trecho de Energia.

Algoritmo	Hidrografia	Malha Viária	Trecho de Energia
R0	93702,75	19791,75	1162
R*	99254,75	20655,25	1483,25
Linear	101977,75	20938,25	1933,25
Quadrático	100703	21136,5	1712,75

O algoritmo da R0 obteve a melhor média de busca em todos os três *datasets*, já o algoritmo linear obteve a pior média de tempo nos *datasets* de Hidrografia e Trecho de Energia. O fato de o linear não ter sido pior no *dataset* de Malha Viária e sim o quadrático pode ter ocorrido pela inconsistência da variação do tempo que resulta em cada teste realizado ou na quantidade de objetos contido no *dataset*. Essa inferência so pode ser comprovada com a realização de testes com maiores *datasets*.

5.2.6 Resultado médio das Consultas no Índice R-Tree

Por meio de uma média realizada com consultas de 5%, 10%, 15% e 20% da região dos *datasets* definidos para as linhas, foi possível construir as Tabelas 5.5, 5.6 e 5.7 para apresentar a comparação entre os algoritmos de *splits*, visando mostrar a eficiência de cada algoritmo em realizar consultas na árvore R-Tree.

As Tabelas 5.5, 5.6 e 5.7 mostra a quantidade de geometrias que foram necessárias ser verificadas para encontrar os elementos de busca, bem como também apresenta a quantidade de nós verdadeiros e falsos que foram necessários ser verificados.

Tabela 5.5: Resultado de consulta realizada no *dataset* de Hidrografia.

Algoritmo	Resultado	Dir. Verdadeiro	Dir. Falso	Folha Verdadeira	Folha Falsa	Geo Verif.
R0	254960	1471	4578	150	108	394665
R*	254960	1758	6385	183	670	667716
Linear	254960	2404	6052	507	406	535587
Quadrático	254960	2208	5567	343	305	474322

A Tabela 5.5 mostra os resultados dos experimentos de todos os algoritmos de *split* realizados no *dataset* de Hidrografia. Nesse *dataset* tem-se que o algoritmo da R0 foi o algoritmo que menos verificou geometrias para realizar a consulta, seguido do quadrático, linear e R*. A R0, além de ser o algoritmo que menos verificou geometrias, também foi o que menos percorreu nós diretórios falsos, nós diretórios verdadeiros, nós folhas falsas e nós folhas verdadeiras.

O algoritmo quadrático foi o segundo que melhor se sobressaiu nos experimentos para verificar geometrias, mas apesar disso, a R* também se destacou em relação ao quadrático e o linear por verificar um número menor de nós diretórios verdadeiros e nós folhas verdadeiras.

Tabela 5.6: Resultado de consulta realizada no *dataset* de Malha Viária.

Algoritmo	Resultado	Dir. Verdadeiro	Dir. Falso	Folha Verdadeira	Folha Falsa	Geo Verif.
R0	54630	698	1380	219	103	115845
R*	54630	845	2099	233	511	237712
Linear	54630	1092	2230	311	406	202604
Quadrático	54630	880	1626	283	206	135638

Assim como para o *dataset* de Hidrografia, o *dataset* de Malha Viária representada na Tabela 5.6 mostra os resultados seguindo o mesmo padrão ocorrido no *dataset* de Hidrografia, onde se tem a R0 sendo o algoritmo que menos verifica geometrias, seguido do quadrático, linear e da R*.

Tabela 5.7: Resultado de consulta realizada no *dataset* Trecho de Energia.

Algoritmo	Resultado	Dir. Verdadeiro	Dir. Falso	Folha Verdadeira	Folha Falsa	Geo Verif.
R0	1106	253	153	146	146	18103
R*	1106	297	204	102	302	34060
Linear	1106	298	201	101	340	34906
Quadrático	1106	330	230	69	612	43822

O cenário muda para o *dataset* Trecho de Energia apresentado na Tabela 5.7. A R0 ainda continua sendo o algoritmo de *split* que menos verifica geometrias durante a consulta, mas há uma mudança entre os algoritmos linear, quadrático e da R*. Para o *dataset* de Trecho de Energia, que é um *dataset* com poucos objetos, a R* e o linear tornam-se superiores ao quadrático, pois verificam menos geometrias durante a consulta.

5.3 Resultados para Objetos do Tipo Polígono

5.3.1 Tempo de Construção do Índice R-Tree

A Figura 5.11, apresenta três gráficos que representam o tempo de construção de índice R-Tree para os dois *datasets* definidos para os polígonos.

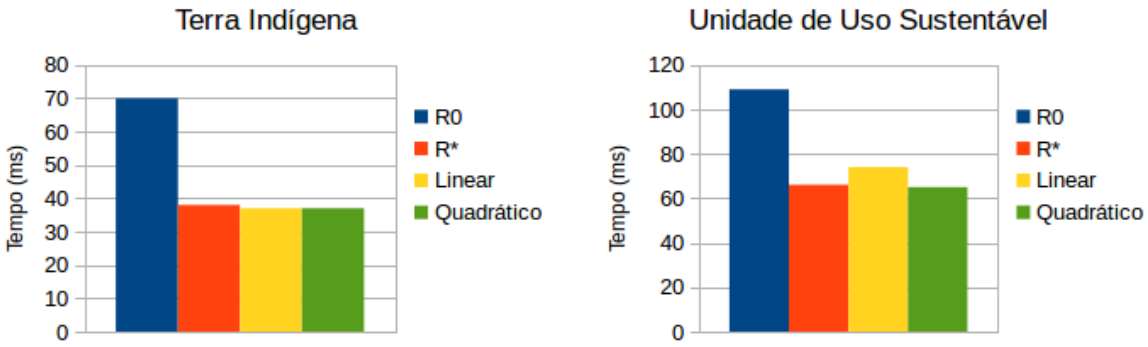

Figura 5.11: Gráficos com a representação do tempo de construção dos indíces da R-Tree para os *datasets* de Terra Indígena e Unidade de Uso Sustentável (polígonos).

Nos dois gráficos o algoritmo de *split* da R0 foi o algoritmo que mais gastou tempo para construir o índice e isso não quer dizer que ele foi o pior, pois pode ocorrer que ele tenha um maior tempo para construir o índice e tenha vantagem em outros critérios, mas nesse ponto de avaliação ele possui o maior tempo. O quadrático realizou o menor tempo para a construção do índice, sobressaindo a R* e ao linear com uma diferença mínima.

5.3.2 Nós Diretórios e Nós Folhas

A seguir, a Figura 5.12 possui dois gráficos para cada um dos *datasets* definidos para os polígonos. Em cada um desses gráficos é apresentado a quantidade de nós diretórios que foram gerados durante a criação do índice.

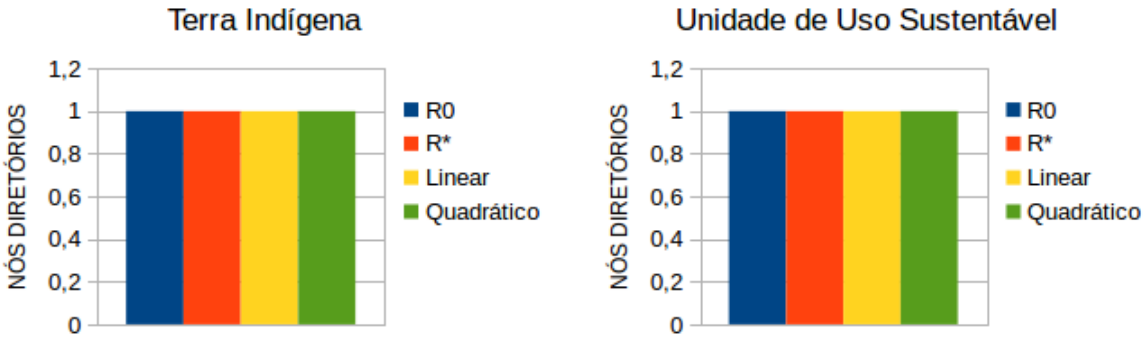

Figura 5.12: Gráficos com a quantidade de nós diretórios para os *datasets* de Terra Indígena e Unidade de Uso Sustentável (polígonos).

Nos dois gráficos todos os algoritmos de *splits* produziram apenas um nó diretório, isso ocorre pelo fato dos dois *datasets* possuírem poucos objetos e a forma que pode se encontrar disposição do objetos em torno do *dataset*. Assim o índice foi gerado com apenas um nível da árvore agregando o nó diretório e o outro nível com todos os nós folhas contendo os objetos.

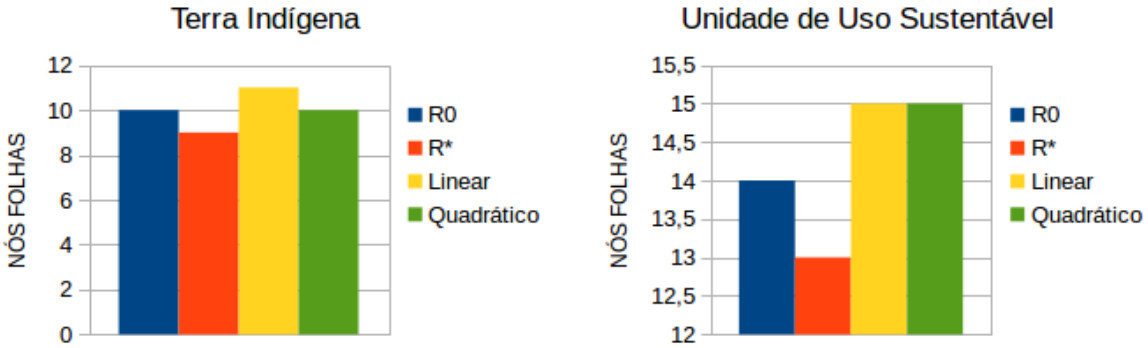

Figura 5.13: Gráficos com a quantidade de nós folhas para os *datasets* de Terra Indígena e Unidade de Uso Sustentável (polígonos).

O algoritmo da R* sobressaiu aos demais algoritmos de *splits* nos dois gráficos, sendo o algoritmo que produziu menos nós folhas. O *dataset* Terra Indígena que possui menos objetos do que o *dataset* Unidade de Uso Sustentável mostra que a diferença entre

os algoritmos de *splits* é baixa e quando o *dataset* possui mais objetos o quadrático tende a produzir mais nós folhas e a R* cada vez mais produzir menos comparado aos demais algoritmos.

Como todos os algoritmos de *splits* produziram a mesma quantidade de nós diretórios, para encontrar o algoritmo que exige mais espaço de armazenamento basta que seja observado novamente a Figura 5.13, visto que a soma de todos os nós é acrescentar +1 nos resultados apresentados nos gráficos da Figura 5.13. Com isso, os algoritmos que mais produziram nós folhas também são os que mais exigem espaço de armazenamento.

5.3.3 Sobreposição de Nós Diretórios e Folhas

O número de nós diretórios produzidos por todos os algoritmos de *splits* foram os mesmos, sendo apenas um e, portanto na Figura 5.14 o gráfico que é apresentado para os *datasets* de Terra Indígena e Unidade de Uso Sustentável também resultaram em apenas uma sobreposição, pois está ligado diretamente ao resultado que teve no número de nós diretórios.

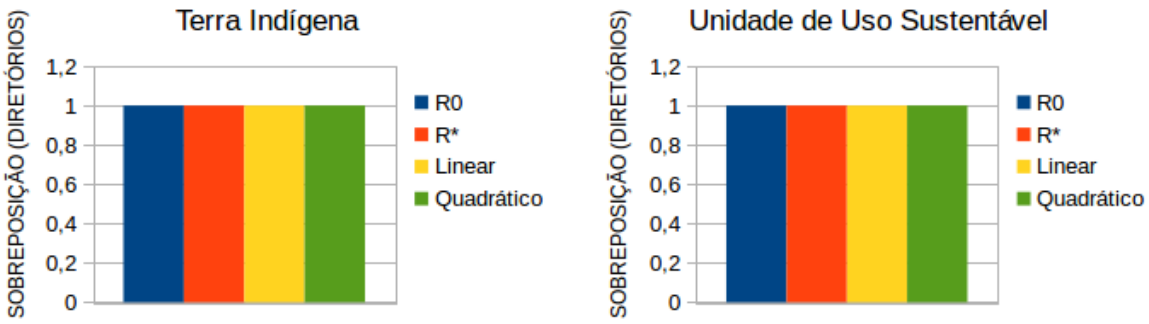

Figura 5.14: Gráficos com a representação das sobreposições nós diretórios para os *datasets* de Terra Indígena e Unidade de Uso Sustentável (polígonos).

A Figura 5.15 apresenta um cenário bastante comum nos trabalhos da literatura para aqueles que na maioria utilizam somente objetos do tipo polígono para os experimentos. Como pode ser visto o algoritmo da R0 e da R* são bastante superiores aos anteriormente propostos por Guttman em relação a quantidade de sobreposições que esses algoritmos produzem. Na Figura 5.15 a R0 sobressaiu aos demais algoritmos produzindo menos sobreposições de nós folhas, logo em seguida veio a R* que também produziu pouca sobreposição, mas porém, maior do que a R0. O pior caso e com grandes sobreposições acontece com o algoritmo Linear, sendo esse o que mais produziu sobreposição e

o quadrático.

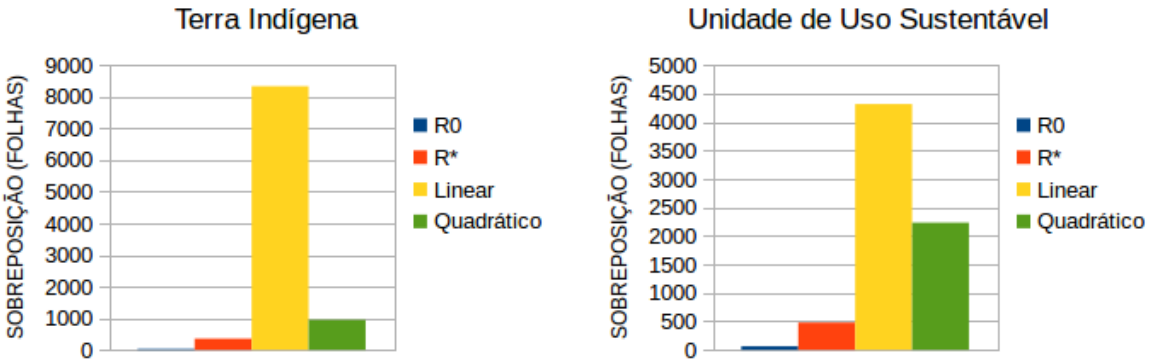

Figura 5.15: Gráficos com a representação das sobreposições nós folhas para os *datasets* de Terra Indígena e Unidade de Uso Sustentável (polígonos).

5.4 Resultados para Objetos do Tipo Ponto

5.4.1 Tempo de Construção do Índice R-Tree

A Figura 5.16, apresenta dois gráficos que representam o tempo de construção de índice R-Tree para os dois *datasets* definidos para pontos.

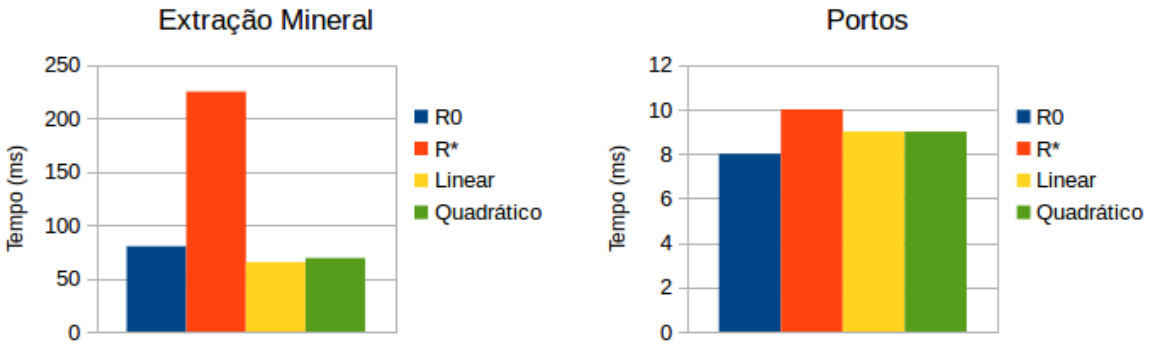

Figura 5.16: Gráficos com o tempo de construção dos indíces da R-Tree para os *datasets* de Extração Mineral e Portos (pontos).

No gráfico para Portos os algoritmos possuem os tempos quase equivalentes, mas o algoritmo da R* possui o pior tempo para criação do índice. O *dataset* Extração Mineral que possui mais objetos do que o de Portos também traz o algoritmo da R* no pior caso para construção do índice. Partindo de um *dataset* com poucos objetos para outro com uma maior quantidade de objetos, percebe-se que a relação de tempo entre

todos os algoritmos de mantém, exceto com a R* que sempre tem um aumento de tempo maior em relação aos outros algoritmos de *split*.

5.4.2 Nós Diretórios e Nós Folhas

A seguir, a Figura 5.17 possui dois gráficos para cada um dos *datasets* definidos para pontos. Em cada um dos gráficos da Figura 5.17 é apresentado a quantidade de nós diretórios que foram criados durante a construção do índice.

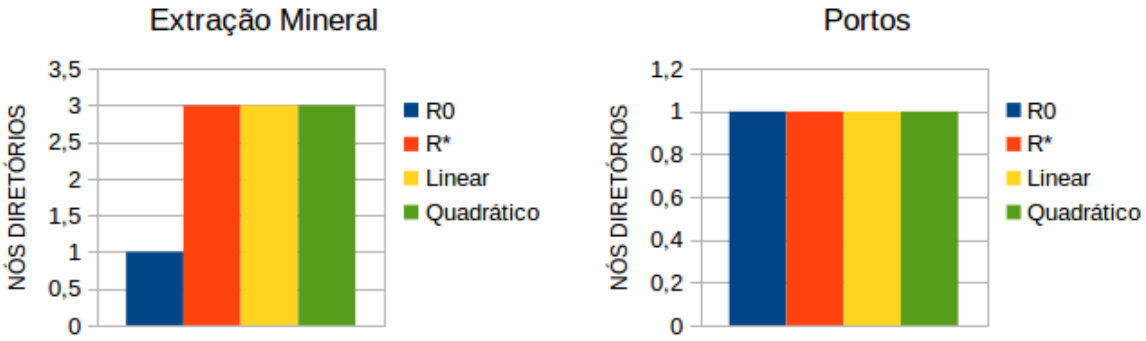

Figura 5.17: Gráficos com a representação da quantidade de nós diretórios para os *datasets* de Extração Mineral e Portos (pontos).

No gráfico do *dataset* de Extração Mineral o algoritmo da R0 produziu apenas uma sobreposição de nós diretórios e os demais cada um produziu três, sendo assim para o *dataset* de Extração Mineral o algoritmo que sobressaiu ao demais foi o da R0. Já para o *dataset* de Portos todos os algoritmos produziram três nós diretórios, pois é um *dataset* que contém poucos objetos.

Observa-se que no *dataset* de Portos que todos os algoritmos produzem uma quantidade de nós folhas quase equivalentes e que o comportamento para encontrar o algoritmo que sobressai aos demais só pode ser visto no *dataset* de Extração Mineral. A media que o número de objetos no *dataset* aumenta a R0 tende a produzir um número menor de nós folhas e o linear tende a produzir um número maior de nós folhas. E isso infere o mesmo para definir o algoritmo que mais exige espaço de armazenamento.

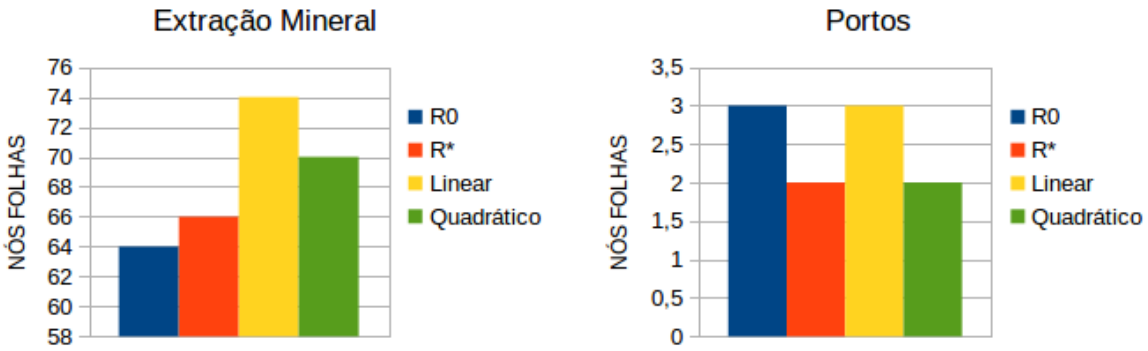

Figura 5.18: Gráficos com a quantidade de nós folhas para os *datasets* de Extração Mineral e Portos (pontos).

5.4.3 Sobreposição de Nós Diretórios e Folhas

O número de nós diretórios produzidos por todos os algoritmos de *splits* foram os mesmos no gráfico de Portos na Figura 5.19 sendo apenas um. Para o gráfico do *dataset* de Extração Mineral o R* tem se o menor número de sobreposições, comparados aos demais algoritmos, embora a diferença seja mínima.

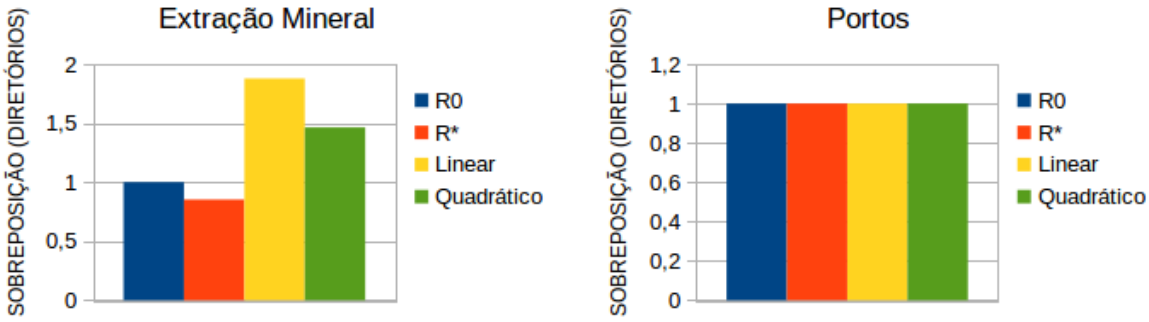

Figura 5.19: Gráficos com a quantidade de sobreposições em nós diretórios para os *datasets* de Extração Mineral e Portos (pontos).

Por meio da Figura 5.20 é possível verificar que os algoritmos da R0 e da R* não produz nenhuma sobreposição e os algoritmos linear e quadrático são os únicos que produz uma sobreposição entre os nós, sendo que o algoritmo linear foi o que mais produziu sobreposição nos dois *datasets* e o quadrático apenas no *dataset* de Extração Mineral. Isso ocorre com o linear pelo fato dele inserir os objetos em seu nó de forma aleatório, sem nenhum critério bem definido como os demais algoritmos de *split*.

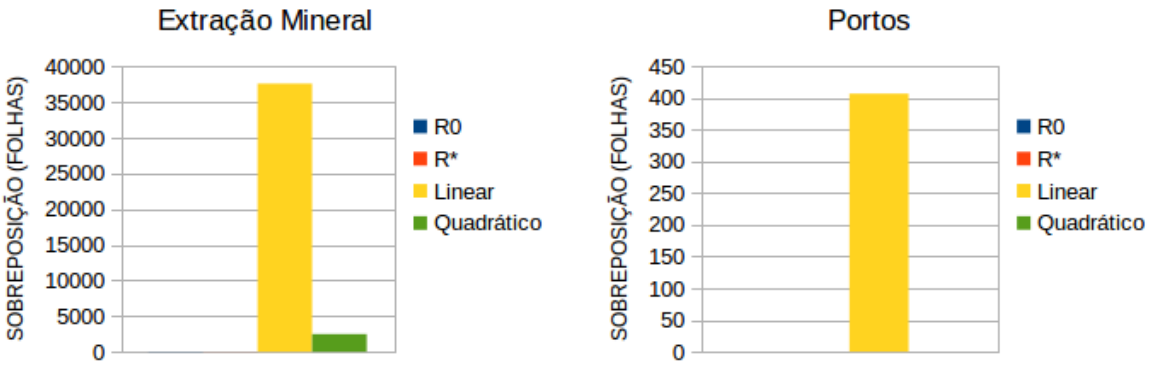

Figura 5.20: Gráficos com a representação da quantidade de sobreposições em nós folhas para os *datasets* de Extração Mineral e Portos (pontos).

6 Conclusão e Trabalhos Futuros

6.1 Conclusão

Este trabalho realiza uma avaliação dos métodos de divisão de nós em árvores R-Tree para objetos do tipo linha, como também de objetos do tipo ponto e polígono afins de controle. Durante esse processo foi realizado experimentos aplicando os algoritmos de *split* linear, quadrático, R0 e R* a um conjuntos da dados espaciais reais.

Foi utilizado duas classes de métricas para avaliar a eficiência de cada algoritmo de *split*, as métricas para avaliação da estrutura multidimensional e as métricas para avaliação de consultas em árvores multidimensional. Com a conclusão dos experimentos não foi possível identificar de forma clara um algoritmo de *split* vencedor, mas foi possível observar que para cada cenário um algoritmo de *split* se destacava em relação aos demais algoritmos.

Analisando o tempo de construção da árvore R-Tree para as linhas conclui-se que o algoritmo da R* possui o pior tempo para construção da árvore em *datasets* com grandes e poucas quantidades de objetos. O melhor tempo para construção da árvore quando o *dataset* possui muitos objetos é o linear, mas para *datasets* com poucos objetos o R0 sobressai aos demais.

A quantidade de nós criados permitiu encontrar o algoritmo que mais exigiu espaço de armazenamento através da soma dos nós diretórios com as dos nós folhas. E nesse cenário foi concluído que o algoritmo da R* exigiu menos espaço de armazenamento quando se indexava *datasets* com grandes quantidades de objetos. Além disso também foi possível determinar que o quadrático é o algoritmo que mais exige espaço de armazenamento em todos os tipos de *dataset*.

As sobreposições entre os nós estão ligadas diretamente com o custo de uma busca, isso foi concluído com o resultado das sobreposições e do tempo de busca. O algoritmo da R0 foi o que menos realizou sobreposição nos nós folhas e diretórios, logo também foi o algoritmo que menos levou tempo para realizar uma busca. Ao contrário, os algoritmos linear e R* foram os que produziram mais sobreposições e tiveram o maior

tempo de busca. Ainda nos experimentos de busca, a R0 sobressaiu aos demais algoritmos de *split* em todos os três *datasets*, sendo o que menos verificou geometrias para retornar a consulta.

A R* foi o algoritmo de *split* que mais buscou maximizar o preenchimento dos nós, e isso fez com que menos nós diretórios e folhas fossem criados. Pois no gráfico que apresenta a quantidade de nós diretórios e folhas percebe-se que a medida que a quantidade de objetos no *dataset* aumentava a R* produzia menos nós comparados aos demais algoritmos. Partindo desse cenário foi possível observar e concluir que ao usar menos espaço de armazenamento ou ter o maior preenchimento dos nós causa o aumento das sobreposições.

Os experimentos realizados com objetos do tipo polígono, o algoritmo linear exigiu mais espaço de armazenamento e o algoritmo da R* menos espaço. Para os experimentos com objetos do tipo ponto, o linear também exigiu mais espaço de armazenamento e a R* menos espaço a medida que a quantidade objetos do *dataset* aumentava. As sobreposições para pontos e polígonos trouxe como resultado o mesmo que acontece em muitos trabalhos da literatura quando se comparam a eficiência do linear e quadrático a outros algoritmos de *split*. A R0 e a R* sobressaem com um elevado desempenho acima dos algoritmos linear e quadrático, tornando-os o pior caso.

Com a conclusão dos experimentos não foi possível determinar qual algoritmo é o melhor em todos os casos, mas sim mostrar qual a vantagem de cada algoritmo e em qual cenário. A escolha de qual algoritmo de *split* utilizar deve ser escolhida conforme os principais critérios para sua finalidade, por exemplo, espaço de armazenamento, tempo de criação da árvore ou tempo de busca.

Além disso outro fato que deve ser levado em consideração é o tipo de objeto que irá se trabalhar, se é ponto, linha ou polígonos. Pois com esse trabalho também foi possível concluir que os algoritmos de *splits* trazem resultados diferentes quando são a aplicados a objetos distintos, como as linhas, polígonos ou pontos. Os experimentos realizados para os objetos do tipo ponto e polígono trouxeram em seus resultados o algoritmo da R* e da R0 como superiores ao linear e ao quadrático em quase todos os cenários, o qual possuíam uma diferença elevada. Essa diferença elevada não pode ser vista nos experimentos para os objetos do tipo linha, o cenário para as linhas são diferentes e isso era o que nossa hipótese afirmava, em que os experimentos apresentados na literatura

não podem generalizar os resultados para objetos do tipo linha, pois as linhas se diferenciam dos pontos e polígonos por serem objetos sinuosos, com formas variadas e tamanhos diferentes.

6.2 Trabalhos Futuros

Este trabalho buscou avaliar os principais algoritmos de *splits* mencionados na literatura. O trabalho pode ser expandido com a realização de novos experimentos junto a outros *datasets* de tamanhos maiores e com a inclusão de novos algoritmos de *splits*.

Além da métrica de de tempo de execução de uma consulta, espera-se acrescentar outras métricas para avaliar com mais precisão uma consulta no índice criado. Com o resultado dos experimentos obtido nesse trabalho espera-se também propôr um novo método de *split* levando em consideração os principais aspectos: tempo de criação do índice, espaço de armazenamento, tempo de busca no árvore e os tipos de objetos para indexar.

Referências Bibliográficas

[Ang e Tan 1997]ANG, C.-H.; TAN, T. C. New linear node splitting algorithm for r-trees. In: *Proceedings of the 5th International Symposium on Advances in Spatial Databases*. London, UK, UK: Springer-Verlag, 1997. (SSD '97), p. 339–349. ISBN 3-540-63238-7. Disponível em: <http://dl.acm.org/citation.cfm?id=647225.718938>.

[Beckmann et al. 1990]BECKMANN, N. et al. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, v. 19, p. 322–331, 1990. ISSN 01635808.

[Bed e Monteiro 2008]BED, F. A.; MONTEIRO, M. V. Comparação do desempenho dos índices R-Tree, Grade fixa e Curvas de Hilbert para consultas espaciais em bancos de dados geográficos. 2008.

[BUSH 1945]BUSH, V. AS WE MAY THINK. *The Atlantic Monthly*, n. JULY 1945, p. 35–47, 1945.

[Caldwell 2005]CALDWELL, D. R. *Unlocking the Mysteries of the Bounding Box*. 2005. Disponível em: <http://purl.oclc.org/coordinates/a2.htm>.

[Camara 2001]CAMARA, G. Representação computacional de dados geográficos. p. 1–44, 2001.

[Cormen et al. 2009]CORMEN, T. et al. *Introduction to Algorithms*. Third edit. Cambridge, Massachusetts London, England: The MIT Press, 2009. ISSN 01605682. ISBN 9780262033848.

[Davis e Rocha 2007]DAVIS, C.; ROCHA, H. B. *Fundamentos de geoprocessamento*. [S.l.: s.n.], 2007. 26 p.

[Filho, Traina e Junior 1999]FILHO, R. F. S.; TRAINA, A. J. M.; JUNIOR, C. T. Indexando Pontos em Espaços com Altas Dimensões Através dos Métodos de Indexação Espacial K-D-B-Tree, R-Tree e TV-Tree. 1999.

[Gaede e Gunther 1998]GAEDE, V.; GUNTHER, O. Multidimensional access methods. *ACM Computing Surveys*, v. 30, p. 170–231, 1998.

[Guttman 1984]GUTTMAN, A. R-trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data - SIGMOD '84*, p. 47–57, 1984. ISSN 01635808. Disponível em: <http://portal.acm.org/citation.cfm?doid=602259.602266>.

[Jaluta 2014]JALUTA, I. Highly Concurrent Multidimensional Index for Spatial Database Management Systems. p. 1–7, 2014.

[Korotkov 2012]KOROTKOV, a. A new double sorting-based node splitting algorithm for R-tree. *Programming and Computer Software*, v. 38, n. 3, p. 109–118, 2012. ISSN 0361-7688.

[Liu, Fang e Han 2009]LIU, Y.; FANG, J.; HAN, C. A new R-tree node splitting algorithm using MBR partition policy. *2009 17th International Conference on Geoinformatics, Geoinformatics 2009*, 2009.

[Oliveira, Costa e Rodrigues 2015]OLIVEIRA, T. B. de; COSTA, F. M.; RODRIGUES, V. J. S. Definição de Planos de Execução Distribuídos para Consultas de Junção Espacial usando Histogramas Multidimensionais. *Proc. of XXX SBBD*, p. 89–100, 2015. ISSN 2316-5170.

[Oliveira et al. 2011]OLIVEIRA, T. B. de et al. DSI-RTree - Um Índice R-Tree Escalável Distribuído. *Sbrc 2011*, p. 719–732, 2011.

[Outsios Evangelos 2011]OUTSIOS EVANGELOS, E. G. Data node splitting policies for improved range query efficiency in k-dimensional point data indexes. *Proceedings - 2011 Panhellenic Conference on Informatics, PCI 2011*, p. 46–50, 2011.

[Sellis, Roussopoulos e Faloutsos 1987]SELLIS, T.; ROUSSOPOULOS, N.; FALOUTSOS, C. Index Multi-Dimensional Objects +. 1987.

[Sleit e Al-Nsour 2014]SLEIT, a.; AL-NSOUR, E. Corner-based splitting: An improved node splitting algorithm for R-tree. *Journal of Information Science*, v. 40, p. 222–236, 2014. ISSN 0165-5515. Disponível em: <http://jis.sagepub.com/cgi/doi/10.1177/0165551513516709>.

[Xia e Zhang 2005]XIA, T.; ZHANG, D. Improving the R*-tree with outlier handling techniques. In: ACM. *Proceedings of the 2005 international workshop on Geographic information systems - GIS '05*. ACM Press, 2005. p. 125. ISBN 1595931465. Disponível em: <http://portal.acm.org/citation.cfm?doid=1097064.1097083>.

[Yannis et al. 2006]YANNIS, M. et al. R-Trees: Theory and Applications. p. 171, 2006.

[Yi 2012]YI, K. The priority R-tree. *SIGSPATIAL Special*, v. 4, n. 2, p. 8–12, 2012. ISSN 19467729.

[Zhang e Xia 2004]ZHANG, D.; XIA, T. A novel improvement to the R*-tree spatial index using gain/loss metrics. In: ACM. *Proceedings of the 12th annual ACM international workshop on Geographic information systems*. [S.l.], 2004. p. 204–213.