Um Novo Método de Indexação para Consultas por Similaridade Utilizando Mapeamentos Unidimensionais Baseados em Focos Globais

Rafael Lucas Bernardes Lima

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2016

Rafael Lucas Bernardes Lima

Um Novo Método de Indexação para Consultas por Similaridade Utilizando Mapeamentos Unidimensionais Baseados em Focos Globais

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Humberto Luiz Razente

Uberlândia
2016

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

L732n 2016

Lima, Rafael Lucas Bernardes, 1989-

Um novo método de indexação para consultas por similaridade utilizando mapeamentos unidimensionais baseados em focos globais [recurso eletrônico] / Rafael Lucas Bernardes Lima. - 2016.

Orientador: Humberto Luiz Razente.

Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.

Modo de acesso: Internet.

Disponível em: http://dx.doi.org/10.14393/ufu.di.2019.314

Inclui bibliografia.

Inclui ilustrações.

1. Computação. 2. Recuperação de dados (Computação). 3. Recuperação da informação. I. Razente, Humberto Luiz (Orient.) II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Maria Salete de Freitas Pinheiro - CRB6/1262

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Um Novo Método de Indexação para Consultas por Similaridade Utilizando Mapeamentos Unidimensionais Baseados em Focos Globais**" por **Rafael Lucas Bernardes Lima** como parte dos requisitos exigidos para obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, _____ de _____ de _____

Orientador: _____
Prof. Dr. Humberto Luiz Razente
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Enzo Seraphim
Universidade Federal de Itajuba

_____
Pro. Dr. Marcelo Zanchetta do Nascimento
Universidade Federal de Uberlândia

Dedico este trabalho primeiramente a Deus, por ser essencial em minha vida, autor de meu destino, ao meu pai José Maria, minha mãe Perpetua, aos meus irmãos e a minha noiva Mariana.

Agradecimentos

Agradeço aos meus pais, José Maria e Perpetua, pelo carinho, amor, dedicação e atenção que tiveram, por sempre me apoiarem em minhas decisões fazendo com que o caminho para chegar até aqui não fosse tão árduo.

Ao meu orientador Prof. Humberto Luiz Razente, sempre atencioso, presente em todos os momentos em que eu precisei, sua orientação e confiança foram de grande ajuda para minha formação como pesquisador.

À Profa. Maria Camila Nardini Barioni e ao Dr. Marcos Rodrigues Vieira pelos conselhos e sugestões que contribuíram de uma maneira bastante efetiva com o trabalho.

À minha irmã Dayane Cristina Bernardes de Castro pelos conselhos e confiança que me passou.

À minha noiva Mariana Pereira de Borba pelo carinho, compreensão, amizade, apoio e confiança que tem demostrado.

Aos amigos e colegas de laboratório Regis Michel dos Santos Sousa, Sara Luzia de Melo e Adilmar Coelho Dantas, que contribuíram com conselhos e pela amizade.

À FAPEMIG pelo apoio financeiro para a realização deste trabalho e também às agências CAPES e CNPq.

"O sucesso é ir de fracasso em fracasso sem perder entusiasmo."
(Winston Churchill)

Resumo

Para recuperação de dados complexos o mais adequado é que se utilizem consultas por similaridade. Para otimizar a resposta à uma consulta são utilizados os métodos de acesso. Quando um conjunto de objetos é definido por meio de uma função de distância (métrica) pode-se dizer que esses objetos passam a compor um espaço métrico, o que possibilita a elaboração dos Métodos de Acesso Métricos (MAMs). Geralmente os MAMs são representados por meio de uma estrutura hierárquica. Existem diversas variações das árvores métricas, e uma estrutura interessante para se trabalhar é a $B^{+}-Tree$, uma característica útil dessa estrutura é que os nós folhas são armazenados em uma lista duplamente encadeada facilitando a navegação entre os nós. O método *GroupSim* apresenta uma abordagem baseada no mapeamento, indexação e recuperação de objetos. Primeiramente é realizado o mapeamento dos objetos para espaços unidimensionais baseando-se em objetos representativos previamente escolhidos, após o mapeamento são gerados vetores unidimensionais os quais são indexados em uma única estrutura $B^{+}-Tree$, possibilitando posteriormente que consultas mais eficientes sejam aplicadas. Por meio de experimentos efetuados foi possível notar que o método proposto apresenta um desempenho superior a outros métodos que podem ser encontrados na literatura. Realizando-se consultas *knn* com k variando entre 10 e 100, e utilizando diferentes conjuntos de dados foi possível avaliar o método proposto. Alguns dos resultados obtidos foram comparando o método *GroupSim* e o *iDistance* utilizando a função Euclidiana e a base de dados *Sierpinski*, o método proposto consegue um desempenho de tempo médio 3.400% melhor. Comparando com a *OmniB - Forest* o melhor desempenho obtido é utilizando a base de dados *Covertype* e a função de distância Euclidiana, neste caso o método proposto chega a ter um desempenho de tempo médio para consulta 1000% melhor, e comparando com Acesso Sequencial o desempenho também chega a 1000% utilizando a base de dados *Sierpinski* e a função de distância Euclidiana. Com base nos resultados obtidos por meio dos experimentos, é possível afirmar que o método proposto apresenta desempenho superior à alguns métodos presentes na literatura, como o *iDistance* e o *OmniB-Forest*.

Palavras-chave: $B^{+}-Tree$. Métodos de Acesso. *GroupSim*.

Abstract

To recovering complex data the most appropriate is to use similarity queries. To optimize the response of a query the access methods are used. When a set of objects is defined by a distance function (metric) can be said that these objects became part of a metric space, which allows the preparation of Metric Access Methods (MAM). Generally MAM are represented by a hierarchical structure. There are several variations of metric trees, and an interesting structure to work is the $B^{+}-Tree$, a useful feature of this structure is that the leaf nodes are stored in a doubly linked list facilitating navigation between the nodes. The *GroupSim* method presents an approach based on mapping, indexing and retrieval of objects. First is performed the mapping of objects to one-dimensional spaces based on representative objects previously chosen, after mapping are generated one-dimensional vectors which are indexed in a single structure $B^{+}-Tree$, allowing subsequently more efficient queries are applied. Through experiments carried out it was possible to note that the proposed method has a performance superior to other methods may be found in the literature. By performing *KNN* queries with k varying between 10 and 100, using different sets of data it was possible to assess the proposed method. Some of the results were obtained by comparing the *GroupSim* and *iDistance* method using the Euclidian function and *Sierpinski* database, the proposed method achieves an average of 3.400% better performance. Compared to OmniB - Forest the best performance achieved is using the database *Covertype* and the Euclidean distance function, in this case the proposed method comes to have an average performance for query 1000% better and in comparison with sequential access to performance also arrives to 1000% using the database *Sierpinskie* and the Euclidean distance function. Based on the results obtained from the experiments, it is clear that the proposed method has superior performance to some methods in the literature, like the *iDistance* and the *OmniB-Forest.*.

Keywords: $B^{+}-Tree$. Access Methods. *GroupSim.*.

Lista de ilustrações

Lista de tabelas

Lista de algoritmos

Sumário

Capítulo 1

Introdução

A quantidade de informação digital para ser armazenada e manipulada por Sistemas de Gerenciamento de Bancos de Dados (SGDBs) vem aumentando nos últimos anos. Os dados são gerados por diversos dispositivos utilizados em diferentes áreas de atuação. Essa diversidade faz com que os dados se tornem cada vez mais complexos e isso afeta o modo com que os SGDBs manipulam essas informações. Estes novos tipos de dados incluem dados multimídia (imagem, áudio, vídeo e texto longo), informações geo-referenciadas, séries temporais, dados de telemetria, de engenharia, dados estatísticos, impressões digitais, dados genômicos e sequências de proteínas.

Os SGDBs permitem a indexação dos dados por meio de Métodos de Acesso (MAs) com objetivo de acelerar consultas aos dados. Os tipos de dados que são comumente utilizados pelos SGDBs possuem relação de ordem possibilitando empregar operadores relacionais de comparação diretamente sobre eles a fim de identificar alguma procedência. São exemplos de domínios com relação de ordem: domínio de valores numéricos, de tempo (data e hora) e de cadeias de caracteres.

Os dados complexos, em geral, não possuem relação de ordem, impossibilitando empregar operadores relacionais ($\geq$, $>$, $\leq$ e $<$). Para esses tipos de dados é possível estabelecer uma relação de similaridade uma vez que é de pouca utilidade uma pesquisa que busca encontrar objetos que são exatamente iguais ou exatamente diferentes.

As consultas por similaridade levam em consideração as características que são extraídas dos objetos contidos nos conjuntos de dados. Essas características são representadas por conjuntos denominados vetores de características ou assinaturas. A consulta nesses domínios é denominada Recuperação Baseada em Conteúdo. Os principais tipos de consultas por similaridade são: Consulta por Abrangência (*Range Query*) e Consulta aos K-vizinhos mais próximos (*K-Nearest-Neighbor ou K-NN*) (CIACCIA; PATELLA; ZEZULA, 1997). Entretanto, os MAs tradicionais não permitem a recuperação por similaridade. Para solucionar esse problema surgiu uma nova família de MAs capazes de realizar esse tipo de consulta, os Métodos de Acesso Métricos (MAM).

Os MAMs foram propostos para indexar dados multimídia e estão entre as principais

técnicas utilizadas em consultas por similaridade. Essas técnicas indexam os objetos em função apenas do valor de distância entre eles. Os MAMs se dividem em estáticos e dinâmicos.

Alguns MAMs constroem a estrutura de indexação em uma única operação utilizando todos os dados. Por isso, são conhecidos como métodos de acesso estáticos. Um trabalho pioneiro foi proposto em (BURKHARD; KELLER, 1973), em que são apresentadas técnicas de particionamento do espaço métrico de forma hierárquica. A primeira é a decomposição hierárquica multivias. Nesta técnica um elemento representativo é escolhido para o domínio e os demais elementos são agrupados de acordo com a distância para ele. A segunda técnica é responsável por dividir o espaço de dados em relação a um número fixo de representantes. A terceira técnica é similar a segunda, porém, estabelece uma constante c, que limita a distância de dois elementos dentro de um mesmo conjunto.

Existem diversos outros trabalhos e entre eles estão a *VP-Tree* (YIANILOS, 1993) e *MVP-Tree* (BOZKAYA; OZSOYOGLU, 1997; BOZKAYA; OZSOYOGLU, 1999).

Na *VP-Tree* (*Vantage Point Tree*) é escolhido um objeto representativo e o conjunto de dados é particionado baseando-se na distância entre os elementos do conjunto de dados e o objeto representativo. Uma árvore binária é construída recursivamente baseando-se no objeto representativo e nas partições formadas. Essa estrutura depende da escolha do objeto representativo, e não é balanceada ou paginada.

A *MVP-Tree* (*Multiple Vantage Point*) possui as mesmas funcionalidades da *VP-Tree*, porém não utiliza uma árvore binária para indexação. Nessa estrutura são escolhidos múltiplos objetos representativos para particionar os objetos do conjunto. Com essa estrutura é possível diminuir a quantidade dos cálculos de distância necessários para uma busca, e o fato de possuir menor profundidade ajudam a diminuir o tempo gasto nas consultas.

Mesmo com bons resultados de tempo para recuperação de objetos, as estruturas criadas utilizando Métodos de Acesso Estáticos não permitem realizar operações de inserção e remoção após sua construção. Para resolver este problema surgiram os Métodos de Acesso Dinâmicos que possuem algoritmos que efetuam a auto-organização da estrutura. Uma estrutura pioneira entre os Métodos de Acesso Métricos Dinâmicos é *M-Tree* (CIACCIA; PATELLA; ZEZULA, 1997). A M-tree é uma estrutura de dados baseada em páginas de disco de tamanho fixo, tem construção *bottom-up* (garantindo o auto-balanceamento), resolve o problema de *overflow* das páginas por meio da divisão do nó de modo que sua propagação possa refletir até a raiz, mas sem interferir em ramos vizinhos. Tais características foram propostas para dados com relação de ordem pela estrutura *B-Tree* e suas variantes (BAYER; MCCREIGHT, 1970; COMER, 1979), e também estão presentes no Método de Acesso Multidimensional *R-Tree* (GUTTMAN, 1984).

Com a criação dos MAs as consultas podem ser mais rápidas do que quando utilizado o acesso sequencial. Alguns autores propõem o uso de técnicas para mapeamento de

objetos em conjunto com técnicas de indexação com objetivo de otimizar ainda mais as consultas. Algumas dessas técnicas compõem a Família Omni de métodos de acesso (JR et al., 2007).

A Família Omni é dividida nas seguintes técnicas: *Omni-Sequential* (onde é utilizado um arquivo sequencial para gerenciamento das consultas), *OmniR-Tree* (é utilizada uma estrutura *R-Tree* para gerenciamento das consultas) e *OmniB-Forest* (onde são utilizadas algumas estruturas *B-Tree* para gerenciamento das consultas).

Além dessas técnicas propostas por (JR et al., 2007) existem diversas outras técnicas com o mesmo objetivo, uma dessas técnicas é a *iDistance* proposta por (JAGADISH et al., 2005). Na técnica *iDistance* os objetos são divididos em n partirções, o número de partições será definido de acordo com um número de objetos representativos que é previamente escolhido. Cada partição formada será indexada em uma única estrutura B^+ - *tree*, com os objetos indexados basta realizar a recuperação, que pode acontecer por meio de uma consulta por abrangência ou *knn*.

1.1 Motivação

O armazenamento de dados para serem recuperados de forma eficiente é um problema que vem sendo explorado ao longo dos anos. Nos últimos anos diversos trabalhos propuseram algoritmos para armazenar e recuperar dados de uma maneira eficiente.

Os Métodos de Acesso são os recursos responsáveis pelo desempenho dos SGDBs, a recuperação eficiente dos dados armazenados em um SGDB está diretamente relacionada ao MA implementado por ele. Por meio das propriedades aplicadas aos MAs é possível descartar subconjuntos de objetos indexados sem a necessidade de acessar e comparar todos objetos. Para isso, basta uma comparação simples entre um objeto de consulta e um dos objetos armazenados. Suponha que os números de 0 à 10 estão indexados, existe uma relação de ordem para essa indexação, um objeto indexado possui um subconjunto maior e outro menor a ele, por exemplo o número 4 possuí um subconjunto dos elementos que são maiores que ele, (5 até 10) e o subconjunto com elementos que são menores (0 até 3). Uma consulta nessa estrutura permite descartar parte dos objetos indexados, possibilitando por exemplo que ao se pesquisar o elemento de número 4 não seja necessário percorrer toda a estrutura.

Na literatura existem diversos trabalhos que abordam estruturas que utilizam algum tipo de Método de Acesso. Como citado anteriormente para trabalhar com dados complexos o ideal é a utilização de algum Métodos de Acesso Métrico o que gera um ganho de tempo nas consultas, e como citado alguns autores propõem o uso de técnicas para mapeamento de objetos em conjunto com esses MAs.

Uma técnica bastante comum para realizar o mapeamento do objetos é baseada na escolha de pontos representativos (pivôs). São escolhidos n pontos representativos em um

conjunto de dados, após definidos esses pontos é calculada a distância de todos objetos que compõem o conjunto de dados para eles. Para cada ponto representativo será gerado um vetor com as distâncias calculadas, esses vetores representam o mapeamento.

Os vetores que são gerados por meio do mapeamento são indexados em n estruturas, onde n é o número de objetos representativos escolhidos. Cada objeto representativo se torna a raiz de uma estrutura, os outros elementos que compõem o vetor são armazenados nos nós intermediários e filhos. Cada nó da estrutura é composto por um identificador do objeto e a distância para o objeto representativo.

As consultas por similaridade utilizando essas técnicas são mais eficientes, pois é possível realizar a poda dos subconjuntos que com certeza não irão compor o conjunto resposta. Os algoritmos para consulta analisam as n estruturas gerando um conjunto com os objetos que compõem uma intersecção que é formada a partir da resposta de cada estrutura. Após esse resultado é necessário fazer o cálculo da distância entre o objeto de consulta e os objetos que compõem o novo conjunto gerado, a resposta são os objetos mais próximos.

A técnica *GroupSim* realiza as consultas de forma similar aos métodos presentes na literatura, porém, o método propõe o uso de apenas uma estrutura para indexar os vetores unidimensionais gerados pelo mapeamento. Com isso, é possível eliminar alguns passos que são adotados por outras técnicas que podem afetar diretamente o desempenho da consulta.

1.2 Objetivos e Desafios da Pesquisa

Desenvolver um método para indexação e recuperação de objetos utilizando MAMs não é uma tarefa trivial. O método deve ser eficiente e apresentar bom desempenho para conjuntos com diferentes dimensionalidades, cardinalidades, e distribuições, para diferentes medidas de dissimilaridade.

O primeiro passo para elaboração de uma estrategia que seja eficiente na indexação e recuperação de objetos é realizar uma boa escolha de objetos pivôs. A seção 2.5.4.3 apresenta algumas estratégias para efetuar essas escolhas. Após a escolha dos pivôs é realizado o mapeamento dos objetos gerando n (neste caso, n é o número de objetos representativos escolhidos, ou seja os pivôs) vetores unidimensionais baseados nas dissimilaridades de todos os objetos do conjunto de dados para os pivôs escolhidos. Esses vetores que serão indexados em uma estrutura $B^{+}-Tree$.

Para indexar os vetores unidimensionais criados em (JR et al., 2007) é proposta a utilização de uma estrutura para cada vetor. Neste trabalho, é proposta a utilização de apenas uma estrutura para obter um ganho de tempo na recuperação.

Quando os vetores são indexados em n estruturas é necessário verificar cada uma dessas estruturas para saber se um objeto pode fazer parte da resposta de uma consulta. Esse tipo de abordagem precisa de um estrutura auxiliar para salvar os objetos que possivelmente

possam fazer parte da resposta, e indexar esses vetores em apenas uma estrutura pode ser uma forma de otimizar as consultas dos objetos.

Ao optar por indexar os objetos em apenas uma estrutura é possível eliminar uma etapa no processo de recuperação. Ao se efetuar uma consulta utilizando n estruturas é necessário utilizar uma estrutura de armazenamento auxiliar para salvar os objetos que compõem a intersecção (objetos que são candidatos a resposta), após saber quais objetos fazem parte da intersecção é necessário realizar uma nova consulta dessa vez na estrutura criada calculando a distância de cado objeto para o objeto consulta com objetivo de definir quem faz parte da resposta. Utilizando apenas uma estrutura de indexação é possível calcular sem a necessidade de uma estrutura auxiliar a distância do objeto para a consulta eliminando assim um passo, que pode ser definido como filtragem.

1.3 Contribuições

Foi proposta uma nova técnica para indexação e recuperação de objetos denominada de *GroupSim*. A técnica é descrita no capítulo 3, com imagens e algoritmos para facilitar o entendimento.

1.4 Organização da Dissertação

O trabalho é organizado da seguinte forma:

- O capítulo 2 apresenta a fundamentação teórica do trabalho, com os tipos de consultas por dissimilaridade existentes definições de funções de distância e espaço métrico. Também são apresentados nesse capítulo Métodos de acesso, mapeamento e indexação de objetos, estrutura de indexação e alguns algoritmos para escolha de pivôs. As literaturas correlatas também estão presentes nesse capítulo.
- No capítulo 3 é apresentada a técnica proposta, com alguns imagens e algoritmos que auxiliam no entedimento.
- O capítulo 4 é referente aos experimentos realizados, neste capítulo são apresentadas as bases de dados e configurações para os experimentos além dos resultados e análises.
- O capítulo 5 apresenta as conclusões e sugestões de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Para se apresentar o universo de pesquisa desta investigação, esta seção fornece noções fundamentais sobre as vertentes que compreendem este trabalho. Inicialmente, é abordado o tema consultas por similaridade e algumas definições. Em seguida, são apresentados conceitos sobre medidas de dissimilaridade, onde são abordadas funções métricas e não métricas. Por fim, alguns conceitos sobre Métodos de Acesso e tipos de consultas **MAs** serão apresentados.

2.1 Consultas por similaridade

Em um Sistema de Gerenciamento de Banco de Dados (SGDB) tradicional, os tipos de dados comumente utilizados incluem basicamente valores numéricos, data/hora e cadeia de caracteres. Esses dados possuem *relação de ordem* e por essa razão é possível empregar operadores relacionais de comparação diretamente sobre eles a fim de tentar identificar alguma precedência.

OS dados que não possuem *relação de ordem* são em sua maioria conhecidos como Dados Complexos. Diferente dos SGDBs tradicionais não é possível empregar operadores relacionais nesses tipos de dados. Isso vale para operadores $<, \leq, >$ e $\geq$. Nesse contexto é de pouca utilidade uma busca para encontrar objetos que são exatamente iguais ou diferentes, logo não faz muito sentido aplicar os operadores $=$ e $\neq$. Suponha uma consulta entre duas imagens. Para fazer uma comparação entre ambas é necessário observar os *pixels* das mesmas, e a não ser que as imagens sejam exatamente as mesmas, vários *pixels* comparados serão diferentes. Ou seja, basta um *pixel* diferente para que as imagens sejam consideradas diferentes. Por isso, faz mais sentido para esse tipo de consulta avaliar a similaridade entre os objetos.

As consultas por similaridade levam em consideração as características extraídas a partir dos objetos presentes no conjunto de dados. As características extraídas dos dados complexos, formam um conjunto denominado *vetor de características* ou *assinatura*· A recuperação dos objetos baseada nesses conjuntos é conhecida como *recuperação baseada*

em conteúdo. Quando a base de dados é constituída por imagens, essa recuperação é conhecida como *recuperação de imagens baseada em conteúdo* (*content based image retrieval – CBIR*).

2.1.1 Consulta de imagem baseada em conteúdo

Recuperação de imagem baseada em conteúdo utiliza diversas características visuais. Essas características foram classificas por (EAKINS; GRAHAM, 1999) em características primitivas como cor, textura e forma, características lógicas como identidade dos objetos mostrados, e características abstratas, como significado de cenas que são apresentadas nas imagens. Mesmo com diferentes possibilidades de características para se trabalhar, a maioria dos sistemas disponíveis utilizam características primitivas como apresentado em (MÜLLER et al., 2004). Estas características são de ampla aplicabilidade, e podem ser utilizadas em uma cena (imagem em um todo), ou em objetos e regiões de interesse. Conforme definido por (SWAIN; BALLARD, 1991).

> *Surface characteristics such as color and texture will typically have only secondary roles in primal access... we may know that a chair has a particular color and texture simultaneously with its volumetric description, but it is only the volumetric description that provides efficient access to the representation of CHAIR.*

O comentário mostra que a forma geométrica é a identidade mais confiável do objeto, embora deva se levar em consideração o contexto representado, e a consulta que será realizada. Considerando uma imagem, onde o objeto não tem grande representatividade, a cor pode ser um recurso de indexação muito eficiente (SWAIN; BALLARD, 1991).

A característica de cor que é amplamente usada em recuperação de imagem pode ter seu espaço discreto representado por um histograma, o qual representa a frequência de cada cor na imagem. Algumas das características de uso do histogramas são: histogramas de cor são invariantes a algumas alterações que a imagem possa sofrer com relação à transação e rotação, pode sofrer poucas mudanças quando se há sobreposição e deslocamento do objeto, ou a profundidade é alterada, como apresentado em (SWAIN; BALLARD, 1991).

As consultas realizadas para recuperação de imagens geralmente utilizam pares de vetores de características, os quais são comparados usando uma medida de dissimilaridade para avaliar o quão próxima uma imagem de consulta está de uma imagem em uma determinada base de dados. A dissimilaridade é calculada por meio de uma função de distância, onde o valor de retorno representa o quão dissimilar os dois vetores de características estão um do outro. Por exemplo, utilizando a imagem de uma praia para a pesquisa, a consulta pode ser feita por meio das características de distribuição de cores (por exemplo, 35% de branco e 65% de azul). Os resultados para essa consulta podem

trazer tanto cenas de praia, como também falsos resultados, talvez pelo fato de existirem imagens que possuem distribuição de cor semelhante à imagem de consulta, mas possuem significados diferentes. Apesar da característica de cor ser amplamente utilizada e apresentar bons resultados, ela não está diretamente ligada à classe do objeto, podendo apresentar imprecisões, como no exemplo citado. Esta visão é bem representada em (BIEDERMAN, 1987). Uma das formas de diminuir essas imprecisões pode ser por meio do tipo de consulta por similaridade utilizado.

2.2 Tipos de consulta por similaridade

Os principais tipos de consulta por similaridade, a Consulta por Abrangência (*Range Query*) representada na Figura 1(a), e a *KNNQ K-Nearest Neighbor Query* (Consulta aos K-vizinhos mais próximos) Figura 1(b), são definidos como (CIACCIA; PATELLA; ZEZULA, 1997):

A ***Range Query*** (Consulta por Abrangência): esta consulta recupera todos os objetos de uma *query* $Q \in D$ que se encontram a uma distância de um raio *r(Q)*. A *Range Query* $\mathbf{RQ}(Q, r(Q))$ seleciona todos os O_j objetos de um conjunto tal que $d(O_j, Q) \leq r(Q)$.

A ***KNNQ*** *K-Nearest Neighbor Query* (Consulta aos K-vizinhos mais próximos): Dado um objeto de consulta $Q \in D$ e um inteiro k tal que $k \geq 1$, a consulta *KNN* recupera os k objetos mais próximos de Q, ou seja, $KNN(Q, k)$ seleciona os k objetos de um conjunto que possuem a menor distância de Q.

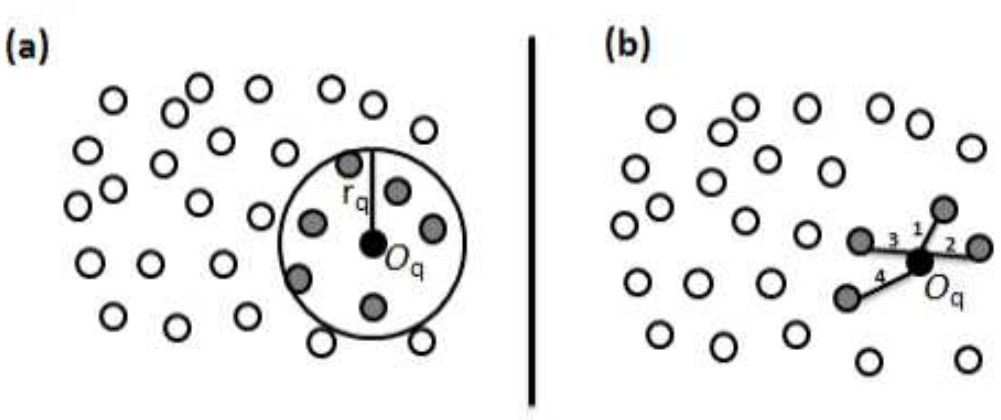

Figura 1 – Exemplos de consultas por similaridade utilizando uma função de distância. O objeto o_q é o objeto de busca enquanto os objetos cinza constituem os objetos do conjunto resposta A. (a) Ilustra uma consulta por abrangência - $RQ(o_q, r_q)$ e (b) ilustra uma consulta aos k-vizinhos mais próximos - $kNNQ(o_q, 4)$.

2.3 Medidas de Dissimilaridade

Para se quantificar a semelhança entre objetos é necessária a definição de uma métrica, ou seja, quanto menor for distância calculada entre dois objetos mais semelhantes eles são, ou diferentes quanto maior for a distância calculada.

A eficácia de uma consulta, ou seja, os resultados recuperados estarem próximos ao que percepção humana considera similar, esta diretamente ligada à medida de dissimilaridade escolhida.

Essa seção apresenta algumas das principais medidas de dissimilaridade, organizadas em *funções métricas* e *não métricas*.

2.3.1 Funções de distância ou métricas

Seja $\mathbf{A} = \{a_1, a_2, a_3\}$ um conjunto de objetos, uma função de distância ou métrica, representada por $d(a_i, a_j)$ deve possuir as seguintes propriedades:

1. **Simetria:** $d(a_1, a_2) = d(a_2, a_1)$;
2. **Identidade:** $a_1 = a_2 \leftrightarrow d(a_1, a_2) = d(a_2, a_1)$;
3. **Não negatividade:** $0 < d(a_1, a_2) < \infty$ *se* $a_1 \neq a_2$ *e* $d(a_1, a_1) = 0$;
4. **Desigualdade triangular:** $d(a_1, a_2) \leq d(a_1, a_3) + d(a_3, a_2)$;

Nas seções seguintes serão apresentados alguns exemplos de funções de distância.

2.3.1.1 Distâncias Minkowski

As funções de distância mais utilizadas são as da família *Minkowski* (norma L_p) (BUGATTI; TRAINA; JR, 2008), geralmente aplicadas em espaços vetoriais. No espaço vetorial, os elementos são definidos por n coordenadas com os valores reais $(x_1, ..., x_n)$. Com isso a família de distâncias de Minkowski (L_p), pode ser definida pela Equação 1:

$$L_p(x, y) = \left(\sum_{1 \leq i \leq D} |x_i - y_i|^p \right)^{1/p}, p \geq 1, x, y \in \mathbb{R}^D. \quad (1)$$

Onde x_i e y_i são dois objetos contidos no conjunto de dados, p é a constante que define o tipo da função, e R é o dominio dos números reais.

Os casos especiais dessa métrica são: distância *Manhanttan* ou distância L_1 (quando $p = 1$) com sua região de cobertura apresentada na Figura 2(a); distância Euclidiana ou distância L_2 (quando $p = 2$), apresentada na Figura 2(b); e a distância *Chessboard*

ou *Chebyshev* ou L_∞ (quando $p = \infty$) apresentada na Figura 2(c). A distância de *Manhanttan* ou distância L_1 é formalmente definida pela Equação 2:

$$d_{L1}(x, y) = \sum_{i=1}^{n} |x_i - y_i|, \tag{2}$$

A seguir a distância Euclidiana (ou distância L_2) também é formalmente definida pela Equação 3. Essa é uma das funções mais usadas para se calcular medidas de similaridade.

$$d_{L2}(x, y) \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}$$

Existem diversas métricas que podem ser usadas para se calcular a similaridade entre objetos, por exemplo, *Manhattan* L_1, Euclidiana L_2, Norma Suprema $L\infty$, dentre outras (DEZA; DEZA, 2009).

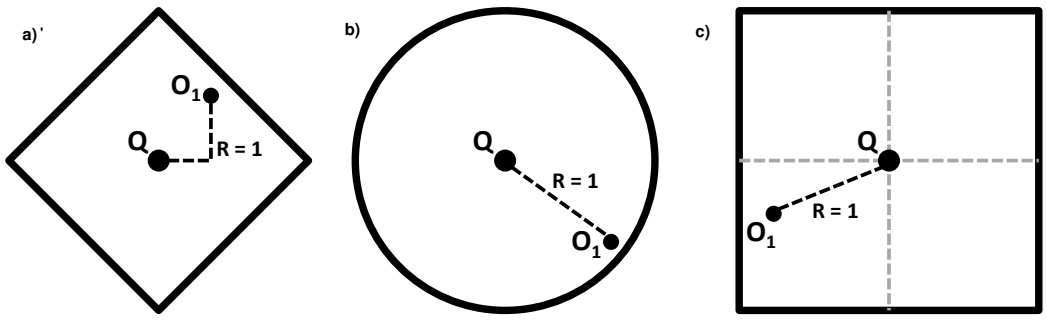

Figura 2 – Região de cobertura das métricas de *Minkowski* (L_p) para raio 1. Onde a) representa a região de cobertura da distância de *Manhattan*, b) representa a região de cobertura da distância euclidiana, e c) representa a região de cobertura da distância *Chessboard*.

2.3.1.2 Quadratic form

A distância *quadratic-form* (ou *Mahalanobis*) é definida na Equação 4:

$$d_{QF}(x, y) \sqrt{(x - y)\Sigma(x - y)^T} \tag{4}$$

onde Σ é definido como uma matriz positiva simétrica, em que algumas correlações entre coordenadas individuais são especificadas. Considere $\Sigma = [x_{ij}]$ uma matriz simétrica n x n com $[x_{ij}]$ sendo os elementos que relacionam a similaridade entre as barras i e j de dois histogramas. O valor de $[x_{ij}]$ é definido como:

$$[x_{ij}] = 1 - \frac{d_{ij}}{max[d_{ij}]}, \text{ onde } d_{ij} = |x_i - y_i|$$

A função de distância *quadratic-form* é bastante aplicada em recuperação de imagens que utiliza histogramas de cores. Essa função pode apresentar bons resultados quando comparada as outras funções de *Minkowski*, pois ela considera a similaridade entre as

barras dos histogramas (LONG; ZHANG; FENG, 2003). De forma geral as métricas de *Minkowski* são bastante aplicadas para recuperação de imagens, principalmente as distâncias L_1 (*Manhanttan*) e L_2 (Euclidiana) (RUBNER et al., 2001).

2.3.1.3 Distância de edição

A Distância de Edição (L_{edit}) ou Distância *Levenshtein*, nome em homenagem ao autor da técnica, foi um dos primeiros algoritmos para comparação de *string* proposto na literatura. Essa técnica é bastante utilizado, Ele calcula o tempo máximo que é gasto para se converter uma *string* em outra.

O Algoritmo 1 mostra um pseudo código da distância, que é calculada por meio de um algoritmo baseado em programação dinâmica, o algoritmo permite inserções, remoções e substituições para se igualar duas *strings*.

Algoritmo 1: Distância de Edição

Entrada: *Strings* A e B.
Saída: M.

```
1  início
2      m = tamanho de A;
3      n = tamanho de B;
4      para i ← 0 até m faça
5          M(i,0) ← i ;
6      fim
7      para j ← 0 até n faça
8          M(0,j) ← j ;
9      fim
       // Calcula cada elemento levando em consideração os resultados
          anteriores.
10     para i ← 1 até m faça
11         para j ← 1 até n faça
12             se A(i) = B(i) então
13                 C ← 0;
14             fim
15             senão
16                 C ← 1;
17             fim
18             M(i,j) ← min { M(i1, j1) + C
                            { M(i1, j) + 1      ;
                            { M(i, j1) + 1
19         fim
20     fim
21 fim
22 retorna M(m, n)
```

A recorrência descrita no Algoritmo 1 tem como ponto chave a linha 16, como a reso-

lução do problema é feita por meio de um algoritmo de programação dinâmica a solução é encontrada utilizando resultados que foram previamente calculados e memorizados.

2.3.2 Funções não métricas

O uso de uma medida de dissimilaridade, a qual não satisfaz pelo menos uma das propriedades que uma métrica possui, pode ser justificado por proporcionar maior robustez, por ser mais resistente a *outliers*, ou seja, a anomalias ou objetos ruidosos (SKOPAL, 2007).

A desigualdade triangular é considerada a propriedade mais importante das medidas de dissimilaridade métricas, pois permite a criação de hierarquias e posterior descarte de elementos em uma busca, gerando oportunidades de otimização (SKOPAL, 2007). Algumas medidas de dissimilaridade não métricas não obedecem a propriedade da desigualdade triangular, podendo gerar um conceito conhecido como *Descontinuidade Semântica.* Por exemplo, na mitologia, semanticamente, a distância do homem para o cavalo é muito maior que a soma das distâncias do homem ao centauro e do centauro ao cavalo, ferindo a desigualdade triangular (Figura 3).

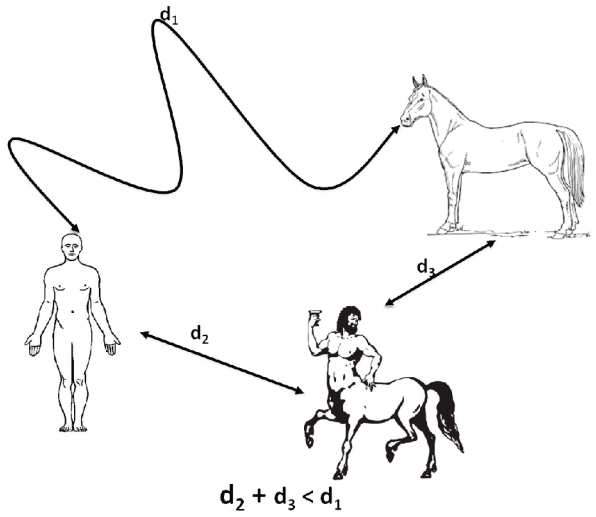

Figura 3 – Descontinuidade semântica e a desigualdade triangular. Adaptado de (SKOPAL, 2007)

Para que uma consulta apresente apenas os resultados esperados, a medida de dissimilaridade aplicada deve seguir a *percepção humana* de similaridade. Portanto, a medida de dissimilaridade aplicada não deve ser limitada por características topológicas (por exemplo axiomas métricos), pois essas características restringem a riqueza da modelagem de similaridade. Seguindo esse pressuposto, uma medida de dissimilaridade mais próxima à percepção humana pode usar uma função não métrica para recuperar os objetos. No entanto, apesar de uma função não métrica apresentar bons resultados com relação a dissimilaridade entre objetos, as medidas métricas são mais aplicadas em recuperação de

dados complexos, uma vez que os objetos podem ser indexados por *Métodos de Acesso Métricos* (subsecção 2.5.1), e, em seguida, recuperados de forma eficiente (SKOPAL, 2007). Muitas das vezes ganhos com eficiência (com relação a tempo de resposta a consulta), podem ser trocados por perda aceitável de eficácia (com relação a precisão da consulta) sendo assim não haveria problemas em se utilizar uma função não métrica para medida de dissimilaridade. Nas seções seguintes, são apresentados alguns exemplos de funções não métricas.

2.3.2.1 Minkowski fracionária

As funções de distância *Minkowski* apresentadas na seção 2.3.1 são métricas. Quando p varia de $0 < p < 1$ tem-se uma distância fracionária L_p que define uma semi-métricas. A Equação 5 apresenta a *Minkowski* fracionária, com $p = \frac{1}{2}$.

$$d_{L\frac{1}{2}}(x, y) \left[\sum_{i=1}^{n}(x_i - y_i)^{\frac{1}{2}}\right]^2 \tag{5}$$

2.3.2.2 Distância Earth Mover's

A medida de dissimilaridade *Earth Mover's Distance* (*EMD*) é baseada na teoria do transporte, nome dado ao estudo do transporte ótimo e alocação dos recursos, que é formalmente definido em (MONGE, 1781). O primeiro trabalho a apresentá-la conceitualmente. O problema do transporte ótimo possui diversas aplicabilidades e tem sido bastante usado ao longo dos anos na teoria da probabilidade, economia e otimização.

O problema consiste em transportar certo volume de massa que está fixado em um local para uma nova região de mesmo volume, para cada unidade de massa transportada existe um custo que deve ser atribuído, ou seja, a massa é transportada de x para y com um custo α para cada unidade transportada. Assim, o problema a ser resolvido é como realizar o transporte gerando um custo total mínimo.

O primeiro trabalho que usou a teoria do transporte apresentada por (MONGE, 1781) como medida de distância foi (PELEG; WERMAN; ROM, 1989), porém sem adotar o nome *Earth Mover's Distance*. Os autores mediam a dissimilaridade entre imagens monocromáticas utilizando o problema do transporte ótimo.

O termo *Earth Mover's Distance* foi sugerido por (STOLFI, 1994), uma analogia feita à alguns programas de CAD[1] para projeto de estradas, os quais têm uma função que calcula o deslocamento ótimo de terra para o preenchimento de buracos em estradas. O termo tem sido usado em diversos trabalhos (RUBNER; GUIBAS; TOMASI, 1997; RUBNER; TOMASI; GUIBAS, 2000; RUBNER et al., 2001), (PELE; WERMAN, 2008;

[1] Nome dado aos sistemas utilizados nas áreas de engenharia, geologia, geografia, arquitetura e design para facilitar projetos e desenhos técnicos.

PELE; WERMAN, 2009), (SKOPAL, 2007), (TANG et al., 2013), dentre outros. A tradução "distância da escavadeira" tem sido usada em trabalhos em língua portuguesa.

A *Earth Mover's Distance* possui a desvantagem de não ser uma métrica para histogramas, que não estão normalizados ao não respeitar as propriedades: de identidade (enquanto o cálculo *EMD* de um ponto é zero para ele mesmo, podem existir pontos distintos em que a *EMD* resulta em zero); e desigualdade triangular (a distância entre dois pontos deve ser menor ou igual a soma das distâncias entre esses dois pontos e um terceiro ponto) (TYPKE; WALCZAK-TYPKE, 2008). A ideia dessa medida de dissimilaridade como mostrado em (TANG et al., 2013) é transformar um histograma em outro com a quantidade mínima possível de trabalho. A medida de dissimilaridade *Earth Mover's Distance* consegue trabalhar bem com *outliers* e pequenas mudanças de valores entre as barras de um histograma, melhorando a qualidade da pesquisa por similaridade em diferentes áreas de domínio, tais como: visão computacional, aprendizagem de máquina, recuperação da informação e recuperação de dados multimídia.

A *Earth Mover's Distance* pode ser usada para medir a dissimilaridade entre dois histogramas, por exemplo, um histograma $q = (q_1, ..., q_n)$ e outro $p = (p_1, ..., p_m)$, onde n e m representam o número de barras, com um Fluxo F representado por $f_{i,j}$, indicando o movimento de q_i para p_j, e uma matriz de custos C, onde $c_{i,j}$ modela o custo do movimento do fluxo da i-ésima barra para a j-ésima barra, pode-se definir o custo total desse movimento na Equação 6:

$$d(q,p) = \sum_{i=1}^{n}\sum_{j=1}^{m} f_{i,j}c_{i,j} \tag{6}$$

2.3.3 Transformação da EMD em uma métrica (FastEMD)

Para solucionar o problema da função *EMD* não respeitar a propriedade da desigualdade triangular, em (PELE; WERMAN, 2008) é apresentado um novo algoritmo baseado no *Earth Mover's Distance* em que por meio de algumas modificações a função passa a respeitar as propriedades das métricas também quando aplicada a histogramas que não estão normalizados. Dados dois histogramas P e Q a *EMD* é definida por (RUBNER; TOMASI; GUIBAS, 2000) como:

$$EMD(P,Q) = \min_{f_{ij}} \frac{\sum_{i,j} f_{ij}d_{ij}}{\sum_{i,j} f_{i,j}} \; s.t \tag{7}$$

$$\sum_j f_{ij} \leq P_i, \sum_i f_{ij} \leq Q_j, \sum_{i,j} f_{ij} = \min(\sum_i P_i, \sum_j Q_j), f_{ij} \geq 0 \tag{8}$$

onde f_{ij} representa o fluxo. Cada f_{ij} equivale ao volume transportado do i-ésimo fornecedor para a demanda j-ésima, d_{ij} é a *ground distance* entre uma barra i e uma barra j de

um histograma. A função proposta por (PELE; WERMAN, 2008) denominada FastEMD é definida como:

$$FastEMD(P,Q) = (min_{f_{ij}} \sum_{i,j} f_{ij}d_{ij}) + |\sum_{i} P_i - \sum_{j} Q_j| \times \alpha \max_{i,j}(d_{ij}) \; s.t \; Eq.8 \qquad (9)$$

Se forem comparados dois histogramas de probabilidade, ou seja, de massa igual a um, as funções *EMD* e *FastEMD* são equivalentes. Quando as massas não são iguais, na *FastEMD* é adicionado um valor ao fornecedor ou à demanda para que as massas tornem-se iguais em ambos os lados. Em (PELE; WERMAN, 2008) também é provado que a técnica proposta é uma medida de dissimilaridade métrica.

A medida de dissimilaridade *FastEMD* compõe o espaço métrico como mencionado anteriormente, porém o fato dessa medida basear-se no problema do transporte ótimo faz com que ela tenha um custo computacional considerável para se avaliar a dissimilaridade entre dois objetos.

Em (PELE; WERMAN, 2009) é apresentado um novo algoritmo que compõe a família *Earth Mover's Distance*. O algoritmo transforma a rede de fluxo da *FastEMD* para que o número de arestas seja reduzido em uma ordem de magnitude. Como resultado é possível calcular a *FastEMD* com uma ordem de magnitude mais rápida do que a proposta por (PELE; WERMAN, 2008), melhorando a escalabilidade para grandes bases de dados. Com essas modificações os autores conseguem algumas vantagens: a) a medida de dissimilaridade se aproxima mais da percepção humana; b) é robusta a ruídos e efeitos de quantização; e c) passa a compor o espaço métrico.

2.4 Espaços métricos

Uma consulta por similaridade pode ser vista como um processo para se obter um conjunto de objetos ordenados pela sua distância para um dado objeto de consulta (*query*). É como uma ordenação, ou classificação, onde o critério é a medida de distância. Embora esse princípio funcione para qualquer medida de distância, ele é melhor representado por funções que satisfazem as propriedades das métricas, (simetria, identidade, não negatividade e desigualdade triangular) (ZEZULA et al., 2006), ao permitir a criação de mecanismos de otimização.

Um espaço métrico $\mathbb{M}$ é definido pelo par $< \mathbf{A}, d() >$, onde *d()* é uma função de distância, como definido na seção 2.3.1, e $\mathbf{A}$ é um domínio de dados. Por meio das propriedades das métricas, principalmente a desigualdade triangular, é possível que os *Métodos de Acesso Métricos* (subseção2.5.1), (assim chamados por se basearem nos espaços métricos) respondam consultas por dissimilaridade de forma mais eficiente, ou seja, mais rápido.

2.5 Métodos de Acesso

Por meio dos Métodos de Acesso (MA) é possível obter um acesso eficiente aos dados, isso utilizando os campos chaves de indexação (ZEZULA et al., 2006), que são usados para construir um índice. Isso possibilita a consulta a um determinado objeto examinando apenas uma pequena fração dos dados armazenados na base (VIEIRA et al., 2004).

Os Métodos de Acesso geralmente utilizam uma abordagem recursiva com uma estrutura hierárquica. Esses MAs são conhecidos como árvores, entre as quais destacam-se:*B-tree, k-d-B-tree, M-tree* e *R-Tree*. A Figura 4 apresenta os exemplos de árvores e algumas definições.

Por definição, uma árvore é um grafo simples acíclico e conexo. Ela é definida em (ZEZULA et al., 2006) como um conjunto finito de um ou mais nós onde:

1. existe um nó denominado *raiz da árvore*, onde somente esse nó não possui ligação de entrada;

2. os demais nós formam n conjuntos disjuntos ($n > 0$), sendo cada um desses conjuntos definidos como $\mathbf{T}$, onde ($i < 1 < n$), formam uma árvore. Essas árvores $\mathbf{T}$ são as *subárvores da raiz*.

Cada nó pode possuir um ou mais elementos conhecidos como entradas de nó, sendo a cardinalidade de um nó, definida como o número de entradas do mesmo. O grau de um nó é definido como o número máximo de ponteiros (ligações de saída) que esse nó pode possuir.

2.5.1 Métodos de Acesso Métricos

Métodos de Acesso Métricos (MAMs) estão entre as principais técnicas usadas em consultas por similaridade. Essas técnicas indexam os objetos apenas em função do valor de distância entre eles.

Os Métodos de Acesso Métricos (MAMs) foram propostos para trabalhar com dados multimídia, com o objetivo de se realizar consultas por similaridade. Logo, se for desejado encontrar um dado objeto de acordo com uma métrica $d()$, pode-se usar os $MAMs$, que organizam um determinado conjunto de dados S de forma que as consultas por similaridade podem ser processadas com mais eficiência, pois o uso do índice métrico faz com que não seja necessário pesquisar todo o conjunto de dados S. Isso ocorre por causa de um principio fundamental por trás de todos os $MAMs$ que é a utilização da *desigualdade triangular*. Quando uma consulta é realizada, apenas as classes candidatas são pesquisadas, tornando a consulta mais eficiente (SKOPAL, 2006).

O principio da desigualdade triangular, que permite que não seja necessário pesquisar todo o conjunto de dados, é definido da seguinte maneira: dado um espaço métrico

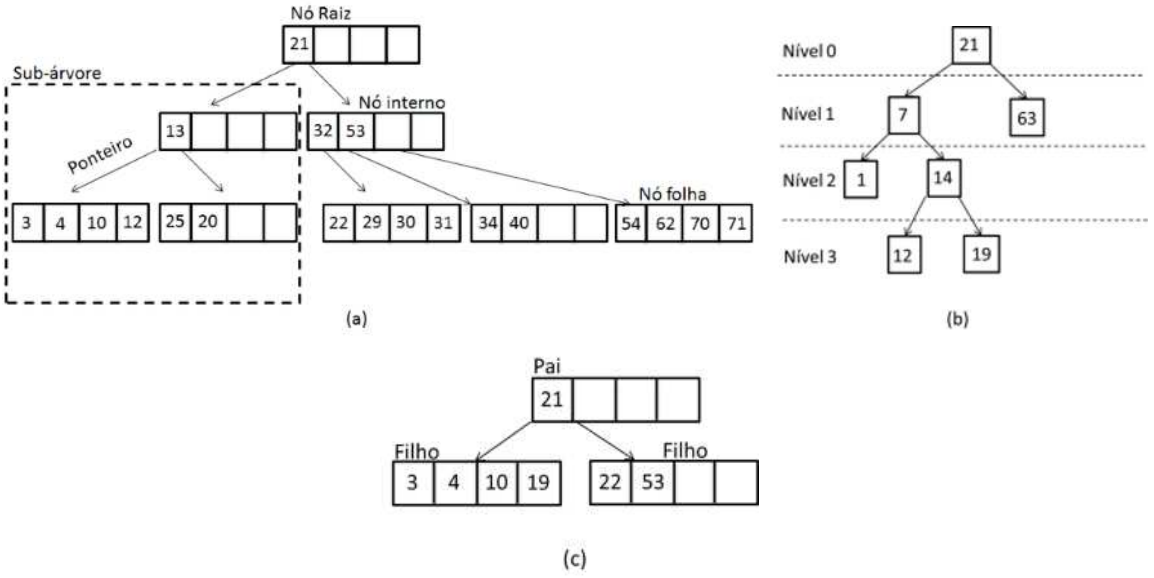

Figura 4 – Exemplo de árvore e algumas definições: (a)Árvore de grau 5, contendo 4 entradas, sendo apresentadas as subárvores, ponteiros entre os nós e a nomenclatura dos nós (raiz, nó interno e nó folha); (b)árvore binária, não balanceada, com altura 4, destacando se os níveis; (c)relação entre um nó pai e seus filhos.

$\mathbb{M} =< \mathbf{D}, d() >$, o conjunto de objetos $\mathbf{O} \subseteq \mathbf{D}$, o objeto de consulta $o_q \in \mathbf{D}$, o raio de consulta r_q e um objeto representativo o_{rep} (veja a Figura 5), um objeto $o_i \in O$ poderá ser descartado se, e somente se, uma das duas condições a seguir for satisfeita:

$$d(o_{rep}, o_i) < d(o_{rep}, o_q) - r_q \tag{10}$$

$$d(o_{rep}, o_i) > d(o_{rep}, o_q) + r_q \tag{11}$$

2.5.2 MAMs Estáticos

Alguns MAMs constroem a estrutura de indexação em uma única operação utilizando todos os dados, por isso, são conhecidos como métodos de acesso estáticos.

O primeiro trabalho encontrado na literatura sobre indexação de dados em domínios métricos de maneira estática foi proposto por (BURKHARD; KELLER, 1973), nele são propostas três técnicas de particionamento do espaço métrico de forma hierárquica.

- A primeira técnica é conhecida como decomposição hierárquica multivias, onde se escolhe, arbitrariamente, um elemento representativo para o domínio e os demais elementos são agrupados de acordo com a distância para o elemento escolhido. É possível separar os elementos com distâncias iguais em grupos, pois os dados são discretos. Isso é feito recursivamente em todos os grupos formados, criando-se vários níveis, definindo-se, assim, uma estrutura hierárquica multivias.

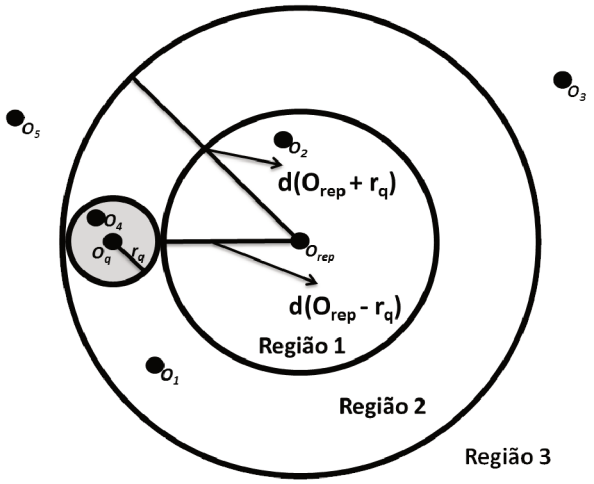

Figura 5 – Pela propriedade da desigualdade triangular, qualquer objeto da região 1 ou 3 pode ser descartado, evitando o cálculo da distância dos mesmos, porém, os objetos que se encontram na região 2 não podem ser descartados apenas com o uso da desigualdade triangular.

- A segunda técnica divide o espaço de dados em relação a um número fixo de representantes. É escolhido arbitrariamente um representante para cada partição e o raio é calculado de acordo com a distância máxima entre os representantes e todos os elementos. Os elementos de cada conjunto são divididos recursivamente de forma igualitária, o que forma uma estrutura hierárquica multívias. Cada nó dessa estrutura guarda os representantes, os raios e ponteiros para os subconjuntos de elementos indexados. Um elemento só pertence a um determinado conjunto, se sua distância ao representante for menor ou igual ao respectivo raio máximo.

- A terceira técnica é semelhante à segunda, porém dois elementos em um mesmo conjunto não devem possuir uma distância máxima entre eles maior do que uma constante c, cujo valor é diferente em cada nível da estrutura, sendo que o grupo que satisfaz a este critério é chamado de *clique*. Para se escolher o valor de c deve se observar os *cliques*, pois, todos os elementos do espaço devem pertencer a pelo menos um clique. Depois se escolhe arbitrariamente um elemento de cada *clique* para ser seu representante.

Um dos métodos de acesso estático mais conhecidos é a *VP-tree(Vantage Point Tree)*. Como é apresentado em (YIANILOS, 1993) esse método particiona um conjunto de dados de acordo com a distância dos elementos com um ponto de referência denominado pivô. Os elementos do conjunto de dados são particionados em dois subconjuntos balanceados, baseando-se no valor médio das distâncias que foram calculadas. Uma árvore binária é construída recursivamente, onde um elemento arbitrário é escolhido como sendo a raiz, e os elementos cuja distância da raiz é menor ou igual à mediana, são inseridos na subárvore

à esquerda, e os elementos cuja distância seja maior, são inseridos a direita. Essa estrutura é dependente da escolha do pivô, e não é balanceada ou paginada.

Outro método de acesso que possui as mesmas funcionalidades da *VP-tree*, porém sem utilizar árvore binária é a *MVP-tree* ou *Multiple Vantage Points Tree*, como mostrado em (BOZKAYA; OZSOYOGLU, 1997) e (BOZKAYA; OZSOYOGLU, 1999). Esse método utiliza múltiplos pontos de vantagem para particionar os elementos. A *MVP-**tree*** também utiliza distâncias já calculadas, armazenando em árvore as distâncias computadas entre os elementos e seus representantes. Com isso, pode aumentar o desempenho da busca, uma vez que o número de cálculos de distância para uma busca vai ser menor. O fato de ser uma estrutura de menor profundidade também ajuda a melhorar o desempenho das consultas.

Existem outros métodos de indexação, que são baseados em estruturas de árvores, sendo os mais conhecidos o *AESA* e o *LAESA*, são variações das estruturas citadas (CHÁVEZ et al., 2001).

2.5.3 MAMs Dinâmicos

Apesar de apresentarem bons resultados com relação a tempo de recuperação após a indexação do objeto, nenhuma das estruturas criadas usando qualquer Método de Acesso Estático é capaz de realizar de modo eficiente operações de inserção e remoção.

O problema dessas estruturas é que sempre que uma dessas operações é executada é necessário uma reorganização que tem um custo considerável, ou então a recriação total da estrutura. Para se resolver esse problema foram propostos Métodos de Acesso Dinâmicos, os quais são algoritmos mais eficientes por trabalhar de forma dinâmica permitindo a auto-organização da estrutura ao serem inseridos novos elementos. Os MAMs dinâmicos (como a *M-tree* e a *Slim-tree*), bem como os métodos de acesso multidimensionais dinâmicos (como a *R-Tree*) têm funcionamento inspirado nos métodos de acesso $B{-}Tree$ e $B^{+}{-}Tree$. Esses métodos são brevemente descritos a seguir.

2.5.3.1 $B{-}Tree$

A $B{-}Tree$ apresentada em (BAYER; MCCREIGHT, 1970) é um tipo de índice muito comum e importante. Essa estrutura ou alguma variação dela pode ser uma forma bastante eficiente para armazenamento e recuperação de dados com relação de ordem. A Figura 6 apresenta um nó de uma árvore do tipo $B{-}Tree$, sendo que esses nós são compostos por chaves C_i, ponteiros P_i e informações $\&_i$, onde:

- as chaves (C_i) representam os atributos pelos quais os dados estão sendo indexados;
- os ponteiros (P_i) são as ligações (ponteiros de memória ou identificador de página de disco), eles apontam para outro nó da árvore; e

- a informação ($\&_i$) representa o endereço onde pode ser encontrado o restante da tupla ligada a chave de pesquisa.

P_1	C_1	$\&_1$	P_2	...	P_i	C_i	$\&_i$

Figura 6 – Organização de um nó em uma $B{-}Tree$.

Uma árvore B de ordem n pode ser definida com uma árvore de busca que possui as seguintes propriedades:

- a raiz deve possuir no mínimo duas e no máximo n subárvores;
- um nó que não seja raiz possui entre $\lceil \frac{n}{2} \rceil$ e n subárvores e também entre $\lceil \frac{n}{2} \rceil - 1$ e $n - 1$ chaves;
- todos os nós folhas estão em um mesmo nível;
- o número máximo de filhos que um nó interno pode conter é de $2n$;

As *B-Trees* são estruturas enárias para armazenamento secundário, possuem altura $O(log(n))$ e páginas de tamanho fixo. A construção de uma $B{-}Tree$ é feita utilizando o modelo *bottom-up* o que garante seu balanceamento.

Busca por elementos em uma $B{-}Tree$

A consulta em uma $B{-}Tree$ é similar à realizada em uma árvore binária, porém na $B{-}Tree$ existem diversos caminhos diferentes a partir de um nó a se percorrer (que varia de acordo com o número de filhos em cada nó), enquanto em uma árvore binária existem apenas dois caminhos em cada nó.

No algoritmo a função de busca recebe um apontador para o nó raiz (r) e a chave C que está sendo procurada. Se a chave C for encontrada o algoritmo retorna o nó ao qual a chave pertence e o índice dentro do nó correspondente, caso contrário, retorna NIL. A figura 7 representa de forma ilustrativa uma consulta.

Inserção de elementos na $B{-}Tree$

A inserção de um elemento em uma $B{-}Tree$ é um pouco mais complicada do que a consulta, pois sempre que uma nova chave é inserida deve se ter o cuidado de não violar as propriedades definidas para $B{-}Tree$, fazendo com que a chave seja inserida no nó correto. Se o nó estiver cheio deve ser feita a separação (split) de acordo com o elemento mediano (elemento que ficaria com a mesma quantidade de chaves a direita e a esquerda), criando assim dois novos nós o quais não violem as definições da estrutura $B{-}Tree$, o

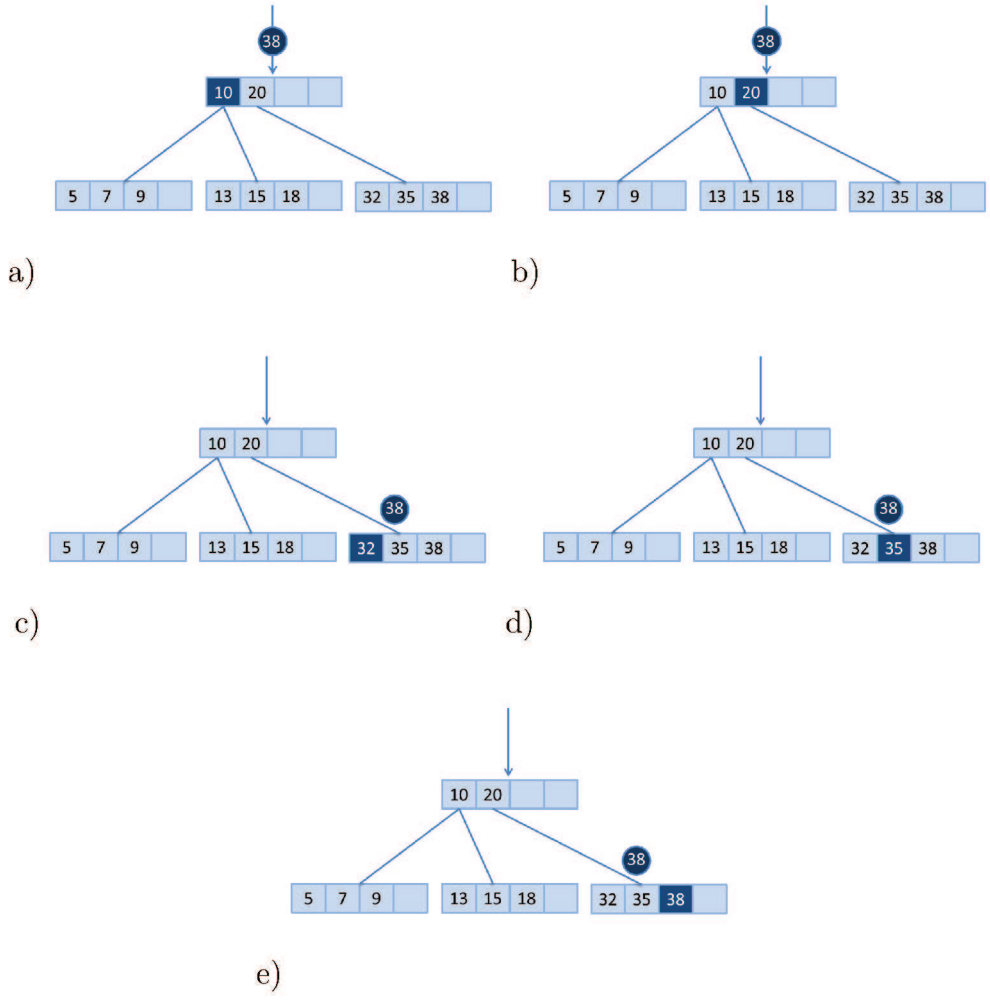

Figura 7 – Ilustração de uma consulta (elemento chave 8) em uma Árvore - B.

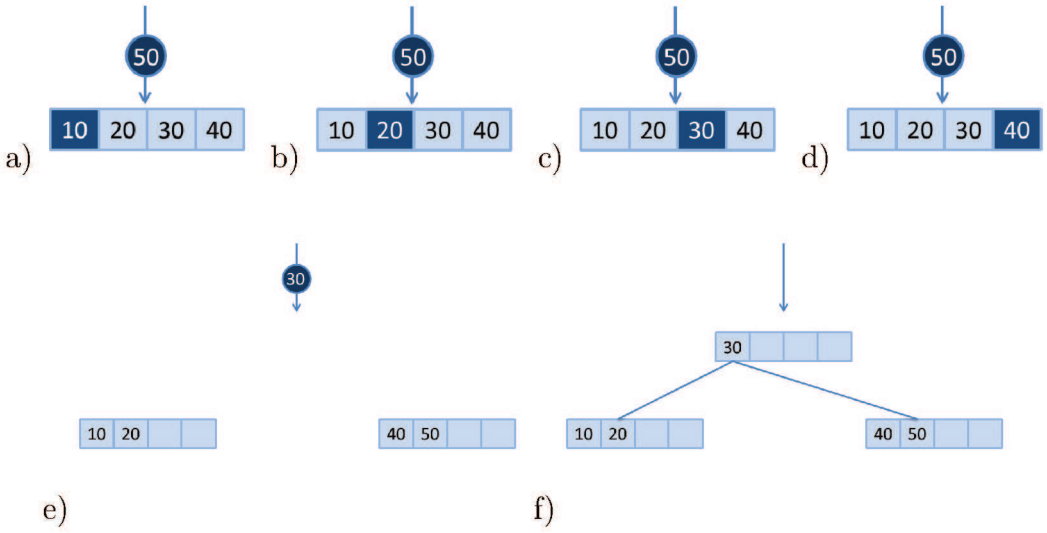

Figura 8 – Ilustração da inserção do elemento chave 50 em uma $B-Tree$.

elemento mediano passa a fazer parte do nó pai. O procedimento de inserção é ilustrado na figura 8.

As evoluções importantes da $B-Tree$ (COMER, 1979) são as estruturas B^+-Tree descrita a seguir e a $B*-Tree$.

2.5.3.2 B$^+$-Tree

A B^+-Tree permite a busca e acesso a uma faixa de valores de modo mais eficiente. Em uma B^+-Tree, todas as chaves se encontram nas folhas. Há ponteiros entre as folhas permitindo que a busca de uma faixa de valores possa encontrar os elementos sem ter que voltar recursivamente nas páginas índices que apontam para essas folhas. Os níveis superiores da estrutura são organizados como na $B-Tree$, consistindo apenas de nós índice que servem como rota para se chegar aos índices chaves (COMER, 1979). As diferenças para a $B-Tree$ são:

- Os dados dos registros ou ponteiros para os mesmos em disco são representados apenas nos nós folhas, as chaves que se encontrarem nos nós internos também estão presentes nos nós folhas.

- Os nós folha são interconectados via uma lista ligada (ou duplamente ligada).

A Figura 9 apresenta um exemplo da estrutura de uma B^+-Tree.

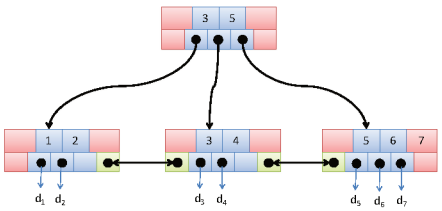

Figura 9 – Exemplo de uma estrutura Árvore - B com ligação dupla entre os nós folhas.

As operações de busca, inserção e remoção são efetuadas de modo similar a $B-Tree$. Na inserção sempre que ocorre a divisão (*split*) o elemento mediano que passará a compor o nó pai, deve continuar na folha, lembrando que a inserção sempre ocorre nos nós folhas. A remoção é um pouco mais complicada, existem dois casos onde no primeiro a chave que deseja remover encontra se em um nó folha, assim basta executar a remoção normalmente. No segundo caso a chave encontra se em um nó interno, para removê-la é preciso que ela seja substituída pelo seu antecessor, basta buscá-lo em seu filho esquerdo, fazer a substituição e depois realizar a remoção como em na $B-Tree$ procurando manter as propriedades da estrutura.

2.5.3.3 M-Tree

A *M-Tree*, possui construção de baixo para cima (*bottom-up*) e não necessita de reestruturações periódicas, o que faz com que o balanceamento da estrutura seja garantido. Todos os elementos são armazenados nos nós folhas, podendo as regiões delimitadas pelo raio de abrangência dos nós irmãos se sobrepor (CIACCIA; PATELLA; ZEZULA, 1997).

Cada estrutura possuí dois tipo de nós, os folhas e os internos. Os nós folhas armazenam os objetos de dados e os nós internos são utilizados para direcionar o processo de busca.

Os elementos sempre são inseridos na subárvore de tal maneira que o seu raio de cobertura não precise aumentar para suportar uma nova entrada. Essa estratégia é adotada para não se criar subárvores com grandes regiões de cobertura o que pode ajudar a evitar sobreposição. Se acontecer empate deve se escolher a subárvore que possui representante mais similar ao objeto que está sendo inserido.

Ao atingir sua capacidade máxima o nó deve ser dividido criando dois objetos representativos um para cada nó. Para a escolha dos objetos representativos existem diversos métodos que podem ser adotados, com o algoritmo MimMax (CIACCIA; PATELLA; ZEZULA, 1997) apresentando os melhores resultados.

A Figura 10 apresenta o exemplo de uma estrutura *M-Tree* com uma distribuição ilustrativa dos dados no espaço. Na Figura 10 (a) é apresentada a estrutura hierárquica *M-Tree* com um nó raiz, no caso desse exemplo, o nó raiz também pode ser considerado um nó interno, a estrutura ilustrada também apresenta três nós folhas. É possível notar por meio da Figura 10 (a) que cada objeto presente no nó raiz é uma representação para um nó folha. Na Figura 10 (b) onde a distribuição dos objetos no espaço é apresentada, é possível notar que o objeto o_5 está na intersecção entre o_1 e o_2, ou seja, o objeto o_5 está dentro do raio de cobertura de ambos os objetos, o mesmo ocorre com o objeto o_8 porém, ele se encontra na intersecção dos objetos o_2 e o_3, quando ocorre de um objeto se encontrar sob dois raios de cobertura, diz se, que há sobreposição na estrutura *M-Tree*.

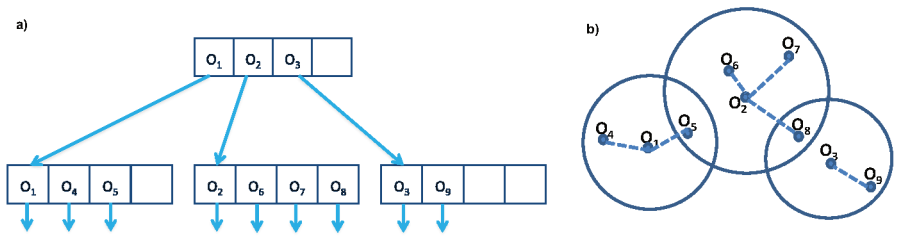

Figura 10 – Representação gráfica *M-Tree*: a) estrutura hierárquica; b) distribuição ilustrativa dos dados no espaço

As consultas efetuadas na estrutura *M-tree* são otimizadas por meio da propriedade da desigualdade triangular, uma vez que apenas os nós que não foram descartados são verificados, com a comparação acontecendo de forma recursiva.

2.5.3.4 Slim-Tree

O método *Slim-tree* proposto em (JR et al., 2000) é similar ao *M-tree*, o objetivo desse método é indexar objetos que compõem um espaço métrico procurando medir e reduzir a sobreposição de nós que podem causar perda de desempenho.

Uma *Slim-tree* possui muitas das características de uma *M-Tree*. Em uma *Slim-Tree* a desigualdade triangular também é usada como uma forma de tentar otimizar as consultas, e como na *M-tree* os elementos são armazenados nos nós folha e há uma hierarquia de representantes para cada nó. A principal diferença entre as duas abordagem está no método de divisão e de escolha das subarvores.

Ao inserir um novo objeto na estrutura *Slim-Tree* deve se verificar se o nó não está com sua capacidade esgotada, se estiver um algoritmo de divisão (*splitting*) deve ser adotado para alocar um novo nó no mesmo nível da árvore dividindo os elementos entre os dois. O algoritmo *MST* (*Minimal Spanning Tree*) (KRUSKAL, 1956) é o mais aplicado para a divisão dos nós pois apresenta bons desempenhos. Ele divide os objetos em dois grupos de acordo com a árvore do caminho mínimo.

2.5.3.5 R-Tree

A estrutura *R-Tree* (GUTTMAN, 1984) é um método de acesso multidimensional dinâmico inspirado pela *B−Tree* (COMER, 1979). Cada nó dessa árvore corresponde a um hiper-retângulo que é armazenado em uma página do disco. A Figura 11, exibe uma representação esquemática do particionamento dos dados com sobreposição. Os nós folhas da estrutura contêm retângulos envolventes mínimos (*Minimum Bounding Rectangle - MBR*) com uma referência para a página de dados.

Na Figura 12 os hiper-retângulos apresentados na Figura 11 aparecem de forma esquemática. Todas as informações referentes aos elementos inseridos na árvore, encontram-se nos nós folhas.

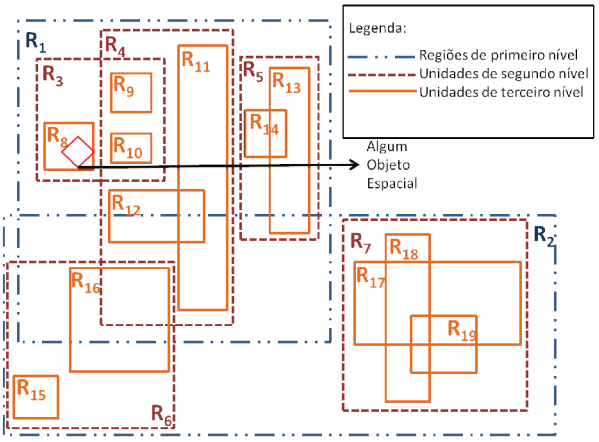

Figura 11 – Retângulos envolventes mínimos (*MBR*) de uma árvore *R-Tree*

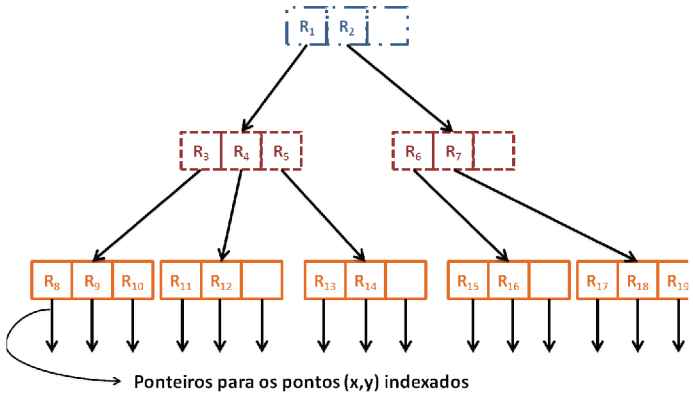

Figura 12 – Representação esquemática dos elementos armazenados na *R-Tree* apresentada na Figura 11

Uma *R-Tree* de ordem (m, M) tem as seguintes características (MANOLOPOULOS et al., 2010):

- Cada nó folha (com exceção do raiz) pode manter até M entradas, enquanto que o número minimo permitido de entradas é de $m \leq M/2$. Cada entrada é da forma $(mbr,\ OID)$, onde o mbr corresponde ao MBR que contém os objetos, e o OID é o identificador do objeto.

- O número de entradas que cada nó interno pode armazenar está entre $m \geq M/2$ e M. Cada entrada é da forma $(mbr,\ p)$, onde p é um ponteiro para o filho do nó, e o mbr corresponde ao MBR que espacialmente contem os $MBRs$ contidos nestes filhos.

- O número mínimo de entradas permitidas em um nó raiz é 2.

- Todas as folhas da *R-Tree* estão em um mesmo nível.

Para realizar consultas na *R-Tree* a intersecção tem um papel muito importante, pois as sub-árvores que têm intersecção com a região de busca serão percorridas. A sobreposição dos $MBRs$ é um dos grandes problemas das *R-Tree*, pois se um ponto pertencer à intersecção de dois $MBRs$, nenhuma das sub-árvores pode ser descartadas.

A estrutura *R-Tree* originalmente proposta por (GUTTMAN, 1984) tem como objetivo realizar um gerenciamento eficiente de grandes coleções de retângulos bidimensionais em aplicações *VLSI* (*Very Large Scale Integration*). A *R-Tree* é uma estrutura dinâmica que organiza os dados dos objetos por meio de uma hierarquia de hiper retângulos. A estrutura suporta inserções, remoções e consultas, utilizando várias heurísticas com o objetivo de minimizar a sobreposição dos $MBRs$ e reduzir seu tamanho. Estas duas propriedades são fundamentais para um processamento eficiente das consultas, porque o desempenho de uma consulta está diretamente ligado ao número de acessos a para determinar a resposta.

2.5.4 Métodos de Acesso baseados no Mapemento de Objetos

Alguns autores propõem o uso de técnicas para mapeamento de objetos em conjunto com técnicas de indexação com o objetivo de otimizar as consultas. Esse mapeamento pode ser feito para espaços unidimensionais, multidimensionais ou métricos. Esta seção apresenta as técnicas de mapeamento e indexação.

2.5.4.1 Métodos de Acesso Omni

O conceito apresentado em (TRAINA et al., 2001; JR et al., 2007) usa um grupo de focos globais para criar mapeamentos unidimensionais, que podem ser aplicados individualmente ou em conjunto com MAs existentes, formando assim os métodos de indexação *Omni-Family*. Os elementos da *Omni-concept* são apresentados a seguir:

- ***base Omni-foci* ($\mathfrak{F}$):** Dado um espaço métrico $\mathbb{M} =< S, d() >$, uma base *Omni-foci* é um grupo $\mathfrak{F} = \{f_1, f_2, ..., f_h | f_g, f_j \in \mathbb{S}, f_g \neq f_j, \forall g \neq j\}$ onde cada f_g é um foco de $\mathbb{S}$, e h é o número de focos na base *Omni-foci*.

- ***Omni-Coordinates***: dados os objetos $s_i \in S$, onde $S \subseteq \mathbb{S}$, e a base *Omni-foci* $\mathfrak{F}$, a *Omni-Coordinates* $C(s_i)$ de um objeto s_i é um grupo de distâncias de s_i para cada foco em $\mathfrak{F}$, onde $C(s_i) = \{< f_g, d(f_g, s_i) >, g = 1, 2, ..., h\}$, onde h é o número de focos na base *Omni*. Para distinguir a distância $d(f_g)$ como uma coordenada, é usada a notação $df_g(s_i) = d(f_g, s_i)$. A cardinalidade de C_i é o mesmo número de focos h na base *Omni-foci* $\mathfrak{F}$.

A Figura 13 exibe um modelo geral de uma estrutura *Omni*. Quando um novo objeto é inserido em uma base de dados, as *Omni-Coordenates* são calculadas e armazenadas. Durante uma pesquisa, as *Omni-Coordenates* são usadas para podar os cálculos de distância por meio da propriedade da desigualdade triangular, apenas os objetos que podem fazer parte da resposta são recuperados do *Arquivo de Acesso Aleatório*. A família *Omni* se divide em *Omni-Sequential* 2.5.4.1, *OmniR-Tree* 2.5.4.1 e *OmniB−Forest* 2.5.4.1.

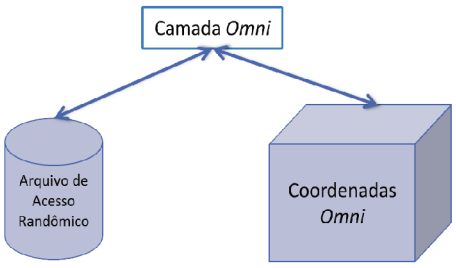

Figura 13 – Representação geral de uma estrutura *Omni*.

Omni-Sequential

Na *Omni-Sequential* os objetos são gerenciados por um Arquivo Sequencial, onde cada entrada é constituída por um vetor das coordenadas *Omni* de um objeto s_i e sua identificação (*OID*). A Figura 14 apresenta uma visão sistemática da estrutura *Omni-Sequential*.

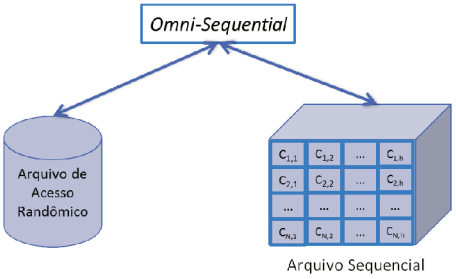

Figura 14 – Representação estrutura *Omni-Sequential*.

A recuperação de objetos nessa estrutura pode ser feita por meio de uma consulta *range query* ou *KNN*. Para consulta *range query* de um objeto s_q com um raio r_q, é feita uma operação de filtragem onde o *Arquivo Sequencial* é lido sequencialmente, comparando as *Coordenadas Omni* de cada objeto s_i armazenadas, com as *Coordenadas Omni* de s_q. O primeiro passo do algoritmo é calcular a distância $d(f_g, s_q)$ do objeto *query* s_q para cada *foci* f_g, criando as coordenadas $C(s_q)$. Então, verifica se $|d(f_g, s_i) - d(f_g, s_q)| > r_q$, para cada *Coordenada Omni* no *Arquivo Sequencial*, verificando todos os arquivos referentes à um *foci* g, onde $1 \leq g \leq h$. Se a comparação for válida para pelo menos um objeto a verificação sequencial é encerrada. A próxima etapa é o cálculo da distância correspondente entre s_i e s_q, neste caso a comparação é com os objetos armazenados no *Arquivo de Acesso Aleatório* e não com as *Coordenadas Omni*. Fazendo essa 'filtragem' inicial por meio do *Arquivo Sequencial*, é possível eliminar diversos cálculos de distância desnecessários, uma vez que a distância só é calculada quando o valor das diferenças das *Coordenadas Omni* de cada s_i para os m *foci* for menor que r_q.

A consulta *KNN* é semelhante a *Range Query*, porém ela possui um raio decrescente, e um conjunto de objetos candidatos a resposta de tamanho **k**, onde o raio da consulta r_q é inicializado com tamanho infinito. O critério de ordenação do conjunto de resposta é a distância entre os vizinhos e o objeto *query* s_q. A lista é preenchida inicialmente com os **k** primeiros objetos encontrados na estrutura *Arquivo Sequencial*, seguindo o mesmo conceito da *Range Query*, ou seja, criando as coordenadas $C(s_q)$ e verificando se $|d(f_g, s_i) - d(f_g, s_q)| > r_q$, se esta comparação não for valida para os h *foci*, r_q é atualizado com relação a $d(s_i, s_q)$ e s_i passa a ser o último objeto do conjunto resposta, essa comparação se repete até que a verificação seja válida, pulando assim para a próxima etapa. Os objetos 'filtrados' na etapa anterior são recuperados do *Arquivo de Acesso Aleatório* por meio do

seu *OID*, as distâncias entre esses objetos são calculadas e ordenadas em ordem crescente, onde os K primeiros objetos compõem a resposta.

OmniR-Tree

Esta técnica aplica os conceitos das *Coordenadas Omni* usando uma estrutura *R-Tree*. A técnica *R-Tree* é um método de acesso desenvolvido para indexar dados vetoriais, onde cada nó índice é representado como um hiper retângulo de limite mínimo (*Minimum Bounding Rectangle* ***MBR***). O *MBR* de um objeto é composto pelos intervalos mínimos de cada dimensão que o limita. Objetos individuais em um espaço métrico não definem uma região, mas é possível imaginar que cada objeto está dentro do *mbOr* de uma *range query* com um raio $r_q = 0$, ou seja, um ponto de consulta centrado naquele objeto. Portanto, o $mbOr(s_i, 0)$ de um objeto s_i é um *MBR* no espaço das coordenadas definidas pela base *Omni-foci*, onde os valores iniciais e finais de cada intervalo são iguais. Assim, o arquivo das *Coordenadas Omni* é substituído por uma *R-Tree* o que pode melhorar a etapa de filtragem. Neste caso, cada entrada é composta pelo $mbOr(s_i, 0)$ e o *OID* do objeto s_i.

A Figura 15 apresenta uma visão esquemática da *OmniR-Tree*, onde um *MBR* da *R-Tree* limita os valores máximos e mínimos para cada *foci*, considerando a região do espaço original (métrico).

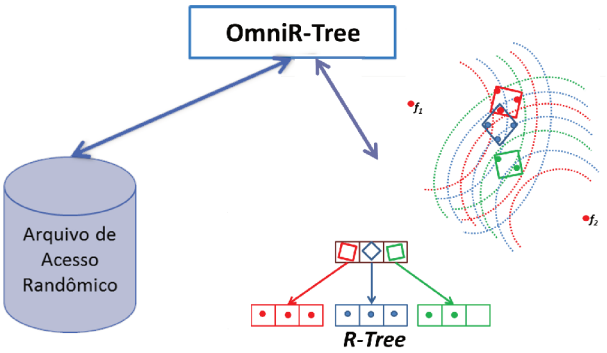

Figura 15 – Representação da *OmniR-Tree*, com o Arquivo de Acesso Aleatório armazenando os objetos, e a *R-Tree* indexando as *Coordenadas Omni* e os *mbOr*s

Os algoritmos de inserção, particionamento de nó, *range query*, dentro outros aplicados à *OmniR-Tree* são iguais aos que são utilizados na *R-Tree*, com exceção do *KNN* (consulta aos K vizinhos mais próximos), diferente dos outros algoritmos esse deve ser substituído. O algoritmo adotado utiliza a mesma abordagem aplicada em árvores métricas. Inicialmente é definido um raio infinito e é realizada uma pesquisa em profundidade para encontrar os k-candidatos a resposta. O raio então é atualizado para o k elemento candidato mais distante, e sempre que esse elemento é substituído o raio é atualizado, essa etapa é executada até que cada entrada que sobrepõe o raio na consulta seja testada.

Os objetos 'filtrados' na etapa anterior são recuperados do *Arquivo de Acesso Randômico* por meio do seu *OID*, as distâncias entre esses objetos são calculadas e ordenadas em ordem crescente, onde os K primeiros objetos compõem a resposta.

Na próxima secção será apresentada a estrutura escolhida como base para essa pesquisa, a $OmniB-Forest$ (TRAINA et al., 2001; JR et al., 2007).

OmniB-Forest

Em (TRAINA et al., 2001; JR et al., 2007) é proposta uma estrutura baseada na B^+-Tree (COMER, 1979) que é denominada $OmniB-Forest$. A estrutura armazena a distância dos objetos da base de dados para cada um dos *foci* e os respectivos *Oids* (identificador de um elemento do conjunto S) de forma que sejam indexados por meio de uma B^+-Tree, com isso a conexão sequencial existente nos nós folhas da B^+-Tree é aproveitada para tentar detectar os objetos mais próximos. A Figura 16 exibe um modelo representativo de uma $OmniB-Forest$, onde a estrutura de acesso aleatório armazena os objetos, e a $B-Forest$ armazena as *Coordenadas Omni*. Essa variação da *Omni-Family* é importante, pois fornece um eficaz suporte a conjuntos de dados métricos utilizando os recursos presentes nos BD comerciais.

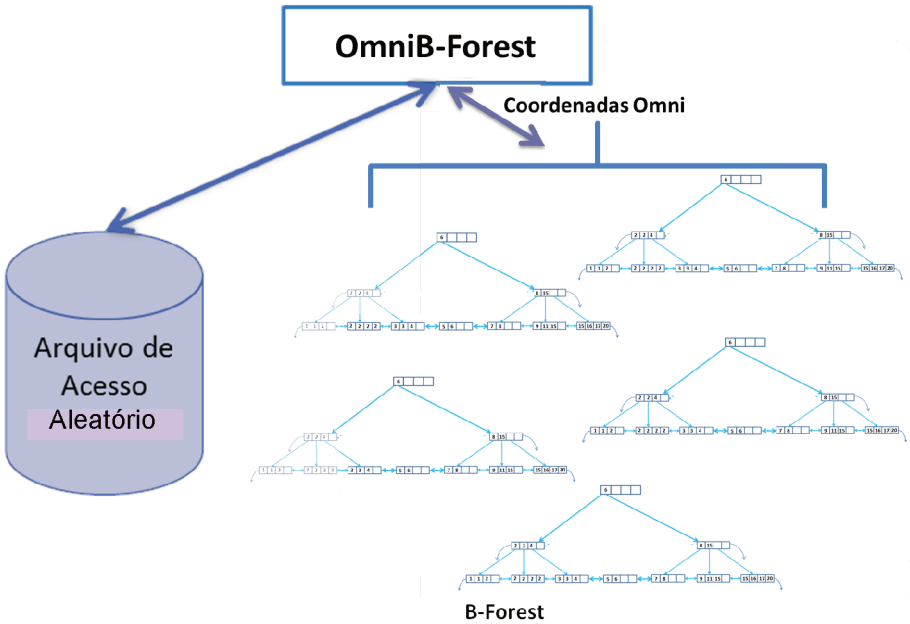

Figura 16 – Representação da *OmniR-Tree*, com o Arquivo de Acesso Aleatório armazenando os objetos, e as estruturas B^+-Tree indexando as *Coordenadas Omni*

Para realizar uma consulta *Range Query* na estrutura $OmniB-Forest$ o primeiro passo é realizar a filtragem, ou seja, obter um conjunto contendo os objetos candidatos a resposta. Para um raio de consulta r_q, os objetos que farão parte desse conjunto são todos os que se encontram no intervalo $I = [d(f_g, s_q) - r_q, d(f_g, s_q) + r_q]$, para cada foco g $(1 \leq g \leq h)$. Para avaliar a intersecção é utilizado um vetor V de contadores com o

tamanho do conjunto de objetos, ou seja, um contador para cada objeto $IOID(s_i)$ do conjunto de dados. Cada posição do vetor armazena quantas vezes um objeto compõe o intervalo I, sendo uma vez para cada foco, assim o número máximo de cada posição é a quantidade de focos m. Para grandes conjuntos de dados, ao invés do vetor de contadores para computar a intersecção pode-se optar por usar uma árvore binária balanceada onde a chave é o identificador e o valor corresponde a contagem. A etapa de refinamento consiste em varrer o vetor V procurando as posições onde o valor é igual ao número de focos, ou seja, igual a m, os objetos são recuperados do Arquivo de Acesso Aleatório e então é feita a validação das distâncias para saber quais objetos constituem a resposta. O Algoritmo 3 descreve o processo da consulta que tem como entrada os objetos do Arquivo de Acesso Aleatório, o conjunto de *B-trees* com as *Coordenadas Omni*, o objeto de consulta s_q e o raio da consulta r_q, e como resposta ele retorna um conjunto de objetos.

Algoritmo 2: *Range Query* PARA A $OmniB{-}Forest$

Entrada: A base de dados S armazenada no Arquivo de Acesso Randômico, o conjunto de $B{-}Tree$s onde estão armazenadas as *Coordenadas Omni*, o objeto de consulta s_q e o raio para consulta r_q.

Saída: Conjunto de resposta.

```
1  início
2      Calcule as Coordenadas-Omni C(s_q) = d(f_g, s_q), 1 ≤ g ≤ h do centro da
        consulta s_q;
3      V um vetor de contadores, um contador para cada objeto s_i no Arquivo de
        Acesso Randômico. Atribua 0 para todas as posições de V;
4      para cada g, onde 1 ≤ g ≤ h faça
           // Etapa de Filtro.
5          Defina o intervalo das chaves validas I_g = [d(f_g, s_q) − r_q, d(f_g, s_q) + r_q];
6          Percorra a B−Tree correspondente ao foco m, procurando a coordenada
            (chaves) que faz parte do intervalo I_m, e:;
7          para cada objeto s_i, onde a coordenada d(f_g, s_i) faz parte do intervalo I_g
            faça
8              pegue o respectivo IOid(s_i);
9              V[IOID(s_i)] ← V[IOID(s_i)] + 1;
10         fim
11     fim
12     para cada posição do Vetor V, onde V[i] = n faça
13         Recupere s_i do Arquivo de Acesso Randômico;
14         se d(s_i, s_q) ≤ r_q então
15             insira s_i no conjunto de resposta;
16         fim
17     fim
18 fim
19 retorna σ(S)
```

Os autores escolheram a estrutura $B^+{-}Tree$ para armazenar as distâncias calculadas, porque com essa estrutura é possível otimizar a consulta *KNN*. Em uma $B^+{-}Tree$ todas

as informações relativas aos objetos estão armazenas em nós folha, e todos os nós que se encontram em um mesmo nível da estrutura estão conectados. A pesquisa se inicia com uma consulta em profundidade para encontrar o primeiro valor chave maior ou igual que $d(f_g, s_q) - r_q$, e busca sequencial para frente até encontrar o limite superior I_g, ou seja, até $d(f_k, s_q) + r_q$.

Uma questão que deve-se atentar na construção da estrutura é a colisão de chaves quando objetos distintos podem se encontrar a uma mesma distância de um foco. A frequência dessas colisões está diretamente relacionada com o domínio da função para medir a dissimilaridade utilizada. A Figura 17 exibe uma B^+-Tree com dupla ligação entre os nós, e colisões entre chaves.

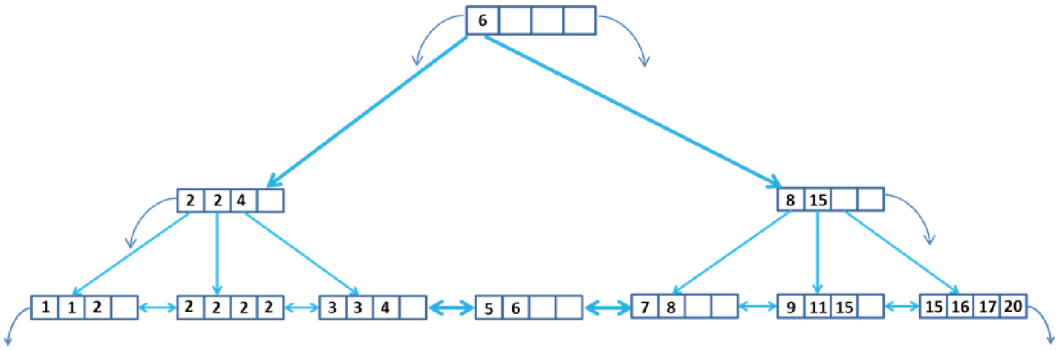

Figura 17 – Representação de uma B^+-Tree usada em uma $OmniB-Forest$ onde todos os nós em um mesmo nível tem uma dupla ligação, com colisões permitidas entre chaves

Para uma consulta *KNN* o algoritmo segue a mesma ideia dos que já foram apresentados com um raio iniciando com valor infinito e sua diminuição até a convergência. O algoritmo encontra os k primeiros objetos que compõem a intersecção e atribui o valor do k-ésimo objeto ao raio. Cada $B^+-Forest$ é pesquisada para encontrar as *Coordenadas-Omni* a partir do objeto de consulta s_q, sendo percorridos um objeto a frente (direita) de s_q e o outro para trás (esquerda) em cada uma das árvores. Este processo é executado para todas as árvores de modo que a intersecção define os objetos candidatos a resposta (lista de identificadores de objetos recuperada das m árvores).

Duas funções foram desenvolvidas para percorrer qualquer uma das $B^+-Forests$ criadas para armazenar as *Coordenadas Omni*, a *step-backward* e a *step-forward*. Essas funções retornam o valor chave $(d(f_g, s_i))$ e o $IOid(s_i)$ para cada $B^+-Forest$ que compõe uma $B-Forest$. Dois cursores definidos em cada $B^+-Forest$, *back-cursor* e *forth-cursor*, permitem percorrer respectivamente as chaves para frente (direita) e para trás (esquerda) de s_q. Um vetor V é usado para fazer o controle dos objetos candidatos. O processo da consulta *KNN* é descrito no algoritmo 3, onde s_q é o objeto de consulta, $C(s_q)$ são as *Coordenadas Omni* de s_q, k é o número de vizinhos desejado, r_q é o raio que começa em infinito, $AS[k]$ é o elemento mais distante do conjunto de respostas já encontrado, I_g^{inf} e I_g^{sup}, são respectivamente os valores de mínimo e máximo do intervalo atual de I_g definido

pelo raio atual de consulta r_q. O passo 4 define um intervalo máximo inicial de nós em uma $B^+-Forest$ onde o algoritmo pode navegar.

A estrutura interna de uma $B^+-Forest$ é usada somente no passo 7 para definir o *back-cursor*$_g$ e o *forth-cursor*$_g$ para cada foco g. Depois disso dois nós folhas de cada $B^+-Forest$ são mantidos na memória principal para executar os passos *step-backward()* e *step-forward()*. Se a estrutura $OmniB-Forest$ for aplicada em grandes conjuntos de dados, o uso de um vetor V para armazenar os objetos candidatos, pode ser substituído por uma árvore binária balanceada ou uma tabela *hash*.

Os algoritmos que compõem a *Família Omni de métodos de acesso* baseiam-se na escolha de elementos pivôs ou *Foci* como definido em (TRAINA et al., 2001; JR et al., 2007) para criarem mapeamentos unidimensionais. Uma boa escolha desses elementos afeta positivamente o resultado de uma consulta. A Seção 2.5.4.3 apresenta alguns métodos encontrados na literatura os quais propõem boas formas para a escolha desses *Foci*s (pivôs).

2.5.4.2 iDistance

O algoritmo *iDistance* foi proposto por (JAGADISH et al., 2005) com o objetivo de otimizar consultas *KNN*.

A primeira etapa do algoritmo é dividir o conjunto de dados em n partições, onde cada partição possui um ponto representativo (pivô). Os objetos são divididos entre as partições de acordo com a relação de proximidade que têm para os pivôs. A Figura 18, exibe uma imagem de mapeamento em um espaço 2-dimensional. As partições são divididas de acordo com os pivôs: O_0, O_1, O_2 e O_3; os pontos A, B, C e D são os dados associados aos pivôs; c_0, c_1, c_2 e c_3, são constantes usadas para gerar intervalos que dividem as partições na estrutura de indexação.

Cada pivô esta associado à uma constante c, calculada com base na maior distância entre pares de elementos do conjunto. O objetivo da constante é separar os objetos de cada partição no mapeamento unidimensional realizado. Na Figura 18, por exemplo, O_0 está associado à c_0, todos os objetos pertencentes a partição referente à O_0 são indexados com relação ao valor de c_0. A equação 12 apresenta o valor a ser indexado, ou seja, a chave indexada para cada objeto de acordo com a equação, seria y.

$$y = i \times c + dist(p, O_i) \tag{12}$$

No segundo passo os objetos que foram mapeados para espaços unidimensionais (espaços que são constituídos pela distância entre os pares objetos e pivôs, com um valor c que é atribuído para gerar intervalos entre as partição), são indexados em uma estrutura B^+-Tree usando a distância (nesse caso o valor resultante da equação 12) como índice. A estrutura B^+-Tree foi escolhida, por ser uma estrutura que é implementada em diversos gerenciadores de bancos de dados comerciais.

Algoritmo 3: *KNN* PARA A $OmniB{-}Forest$

Entrada: A base de dados S armazenada no Arquivo de Acesso Aleatório, o conjunto de $B{-}Trees$ onde estão armazenadas as *Coordenadas Omni*, o objeto de consulta s_q e o número de K vizinhos

Saída: Conjunto dos vizinhos mais próximos em AS

1 **início**
2 Calcule as *Coordenadas-Omni* $C(s_q) = d(f_g, s_q), 1 \leq g \leq h$ do centro da consulta s_q;
3 **V** um vetor de contadores, um contador para cada objeto s_i no Arquivo de Acesso Randômico. Atribua 0 para todas as posições de **V**;
4 Inserir os primeiros k objetos do Arquivo de Acesso Randômico no conjunto de respostas AS, mantendo a lista ordenada de acordo com as distâncias de cada objeto para s_q, com isso o objeto *AS[k]* é o mais distante de r_q, então faça $r_q = d(AS[k], s_q)$;
5 **para** *cada g, onde* $1 \leq g \leq h$ **faça**
6 Defina o intervalo das chaves validas $I_g = [d(f_g, s_q) - r_q, d(f_g, s_q) + r_q]$;
7 Execute uma primeira pesquisa para encontrar o primeiro valor chave $\geq d(f_g, s_q)$ em uma árvore t_g;
8 Set o $back - cursor_g$ e o $forth - cursor_g$ com a posição retornada pela pesquisa;
9 **fim**
10 Set $h \leftarrow 1$;
11 **enquanto** *for possível fazer os passos* ***backward*** *ou* ***forward*** *para alguma das árvores* **faça**
12 /*estas condições podem acontecer em pelo menos uma das árvores: o $back - cursor_h$ não poder ser o início do índice e o $forth - cursor_g$ não pode ser o final do índice, ou seja, $back - cursor_g > I_g^{inf}$ e $forth - cursor_g < I_g^{sup}$.*/;
`// Etapa de Filtro.`
13 Defina o intervalo das chaves validas $I_g = [d(f_g, s_q) - r_q, d(f_g, s_q) + r_q]$;
14 Execute o *step-backward()* na $B - Tree_g$;
15 **se** $back - cursor_g \geq I_g^{inf}$ **então**
16 get i $\leftarrow$ *IOID* $(back - cursor_g)$;
17 incremente V[i];
18 **fim**
19 **se** *V[i]* $\leftarrow$ $\boldsymbol{n}$ **então**
20 recupere s_i do Arquivo de Acesso Randômico;
21 **se** $d(s_i, s_q) < r_q$ **então**
`// Etapa de Refinamento.`
22 Insira s_i na lista de candidatos *AS[]*;
23 Ordene a lista *AS[]*;
24 Set $r_q \leftarrow d(AS[k], s_q)$;
25 **fim**
26 **fim**
27 Repita os passos 14-26 mudando: *step-backward*() para *step-foward*() na etapa 14, e $back - cursor_g \geq I_g^{inf}$ por $forth - cursor_g \leq I_g^{sup}$ no passo 15;
28 Incremente g para a próxima árvore;
29 **fim**
30 **fim**
31 **retorna** $\sigma(S)$

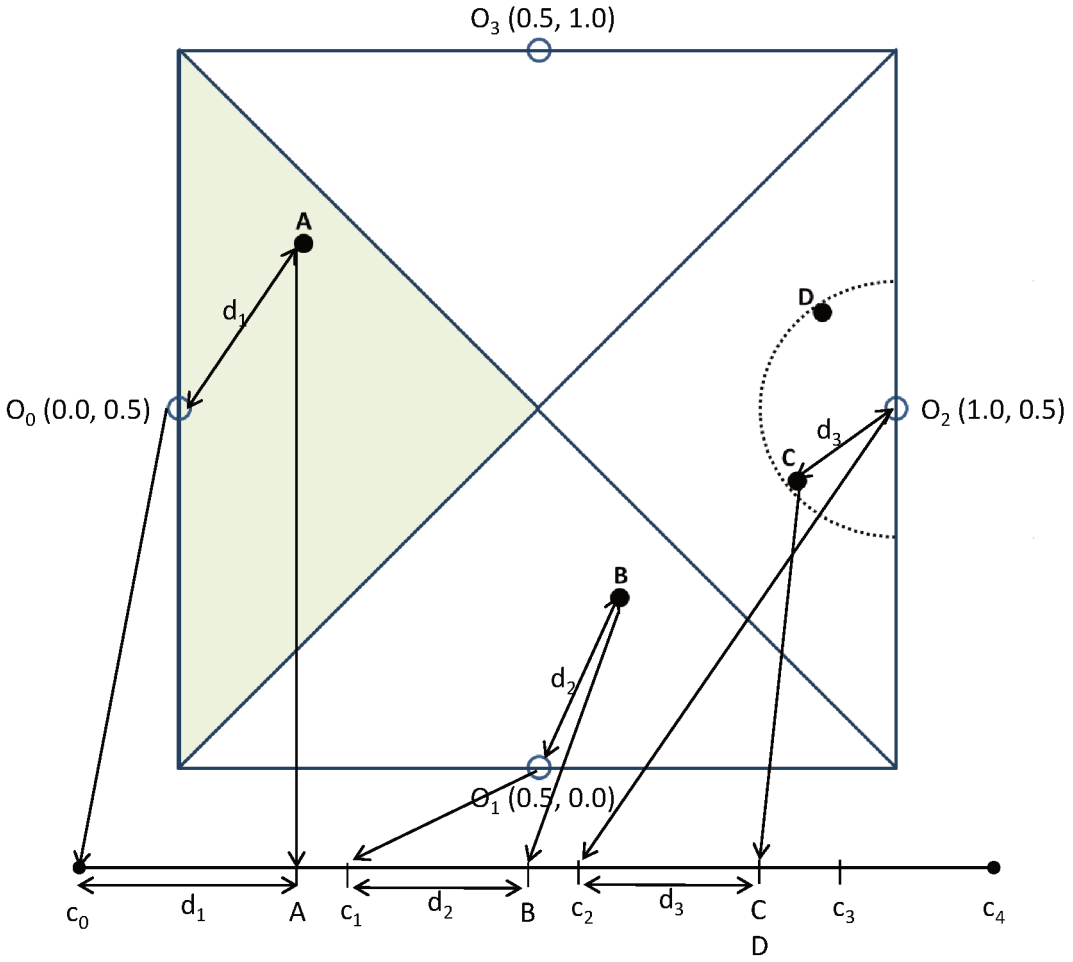

Figura 18 – Mapeamento dos dados em um espaço 2D. Adaptado de (JAGADISH et al., 2005)

O terceiro passo é a consulta *KNN*. A estratégia proposta pelos autores é similar à usada por (TRAINA et al., 2001; JR et al., 2007), baseando-se em uma consulta *range query* onde o raio converge quando os k objetos mais próximos ao dado de consulta são localizados. A diferença é que na estratégia proposta por (TRAINA et al., 2001; JR et al., 2007) o raio começa com um valor infinito e vai decrementando até o algoritmo convergir, já na estratégia adotada no *iDistance* o valor do raio começa em 0 e é incrementado até o algoritmo convergir.

A Figura 19, apresenta a região de cobertura para uma consulta *KNN* utilizando o método *iDistance*. Os pontos O_1, O_2 e O_3 são os pontos representativos das regiões (os pivôs), q é o objeto de consulta, r_0 e r_1 representam o raio crescente. A área sombreada na primeira partição (onde o objeto O_1 se encontra) é relativa a região de cobertura do raio r_1, ou seja, o intervalo $d(O_1, q) - r_1$ e $d(O_1, q) + r_1$. O raio cresce até que os k objetos sejam encontrados ou até atingir o ponto que se encontre mais distante do pivô dentro da partição.

A área sombreada na segunda partição (referente ao pivô O_2) representa uma região de cobertura do raio r_1, porém é considerado somente um pequeno intervalo de $d(O_2, q) - r_1$,

isso porque a distância $d(O_2, q) + r_1$ é maior do que a distância do pivô para o objeto que se encontra mais distante na partição.

Os algoritmos 4, 5 e 6 apresentam a consulta *KNN*. No algoritmo 4 o raio (r) é iniciado com valor 0 e é criada uma variável do tipo *boolean* (*Stopflag*) inicializada com o valor *false* usada no laço principal para definir quando o algoritmo deve encerrar. Dentro do laço a variável r é incrementada em Δ_r (um parâmetro que os autores definem como 0.01) a cada iteração e a função *SearchO* é chamada passando como parâmetro: o objeto de consulta q, a quantidade de objetos a serem recuperados k e o raio r.

No algoritmo 5 é verificado se os K objetos foram encontrados, ou se o valor do raio r é maior do que a distância entre o objeto mais distante dentro da partição e o pivô; se uma dessas condições for atendida o algoritmo convergiu e deve encerrar, caso contrário deve continuar a execução. O laço da linha 6 do algoritmo é responsável por percorrer a B^+-Tree para ambos os lados por meio dos algoritmos *SearchInward* e *SearchOutward*. Porém se a pesquisa estiver dentro de uma intersecção apenas o *SearchInward* é executado porque $d(O_i, q) + r$ ultrapassa o ponto mais distante dentro da partição.

Dado um nó folha como entrada, o algoritmo 5 examina os nós que estão a esquerda, e determina se eles estão entre os k vizinhos mais próximos. Se algum dos nós estiver então o conjunto de resposta é atualizado.

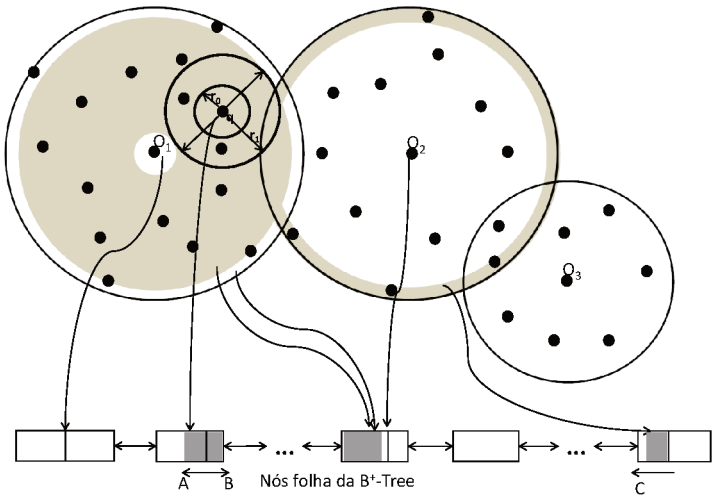

Figura 19 – Região de uma consulta *KNN* utilizando o algoritmo iDistance. Adaptado de (JAGADISH et al., 2005)

2.5.4.3 Algoritmos para escolha de pivôs (*Foci*)

Em algumas seções apresentadas foi adotado o termo *Foci* para se fazer referência aos objetos representativos (pivôs), o que é apenas uma questão de nomenclatura pois ambos tem o mesmo significado. Essa nomenclatura (*Foci*) foi adotada em (TRAINA et

Algoritmo 4: *KNN* Search Algorithm *iDistanceKNN*

Entrada: Objeto de Consulta q, quantidade de objetos a serem recuperados K, e um valor de incremento para o raio $\Delta(r)$
Saída: Conjunto dos vizinhos mais próximos em S

```
1  início
2      r = 0;
3      Stopflag = FALSE;
4      initialize lp[ ], rp[ ], oflag[ ];
5      enquanto Stopflag == FALSE faça
6          r = r + Δr;
7          SearchO(q , r, K);
8      fim
9  fim
10 retorna σ(S)
```

al., 2001; JR et al., 2007) onde os autores utilizam termo para se referirem aos objetos representativos escolhidos para um conjunto de dados. Nesta seção será utilizado apenas o termo pivô.

O algoritmo para escolha de pivôs influencia diretamente na indexação dos objetos e também no resultado das consultas. Existem diversos algoritmos na literatura que procuram fazer uma boa escolha de pivôs. Um bom conjunto de pivôs pode ser formado por objetos que se encontram o mais distante quanto for possível um do outro, com isso, *outiliers* têm grandes possibilidades de serem boas escolhas para pivôs (BUSTOS; NAVARRO; CHÁVEZ, 2003).

Na literatura podem ser encontrados diversos algoritmos para escolhas de pivôs:

- a técnica mais ingênua é a escolha dos pivôs de modo aleatório;
- o *Hull of Foci* (HF) apresentado em (TRAINA et al., 2001; JR et al., 2007) seleciona os pivôs próximos das extremidades do conjunto de dados. Se observado em um plano pode-se notar que o HF gera uma distribuição muito semelhante ao de um hiper-tetraedro;
- os algoritmos *Maximum of Minimum Distances* (MMD) e *Maximum of Sum of Distances (MSD)* são duas técnicas apresentadas em (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011). Estas técnicas escolhem incrementalmente objetos distantes dos que já foram previamente selecionados. Ambas iniciam com a escolha aleatória de um objeto, e incrementalmente realiza-se uma agregação de distância para os outros objetos selecionados.
- O algoritmo *Sparce Spatial Selection* (SSS) foi proposto por (PEDREIRA; BRISABOA, 2007), esse algoritmo é baseado em um parâmetro escolhido de forma empírica, e tem a vantagem segundo os autores de ser um método dinâmico.

Algoritmo 5: *SearchO*

Entrada: Objeto de Consulta q, quantidade de objetos a serem recuperados K, e o raio r

Saída: Conjunto dos vizinhos mais próximos em S

```
1  início
2      Pfurthest = furthest(S,q);
3      se dist(Pfurthest, q) < r e |S| == K então
4          Stopflag = TRUE;
5      fim
6      para i = 0 até m - 1 faça
7          dis = dist(O_i, q);
8          se not oflag[i] então
9              se sphere(O_i, dist_max_i) contem q então
10                 oflag[i] = TRUE;
11                 lnode = LocateLeaf(btree, i * c + dis);
12                 lp[i] = SearchInward(lnode, i * c + dis - r);
13                 rp[i] = SearchOutward(lnode, i * c + dis + r);
14             fim
15             senão se sphere(O_i, dist_max_i) intersecção sphere(q, r) então
16                 oflag[i] = True;
17                 lnode = LocateLeaf(bree, dist_max_i);
18                 lp[i] = SearchInward(Inode, i * c + dis - r);
19             fim
20         fim
21         senão
22             se lp not nil então
23                 lp[i] = SearchInward(lp[i] → leftnode, i * c + dis - r);
24             fim
25             se rp not nil então
26                 rp[i] = SearchOutward(rp[i] → rightnode, i * c + dis + r);
27             fim
28         fim
29     fim
30 fim
31 retorna σ(S)
```

Algoritmo 6: *SearchInward*

Entrada: Nó folha (*node*), distância $d(O_i, q)$ somada de $i * c$ menos o raio atual(*ivalue*)
Saída: Nó folha *node*

```
1  início
2      para cada entrada e no node(e = e_j, j = 1, 2, ..., Número_de_entradas faça
3          se |S| == K então
4              Pfurthest = futhest(S,q);
5              se dist(e, q) < dist(Pfurthest, q) então
6                  S = S − Pfurthest;
7                  S ∪ e;
8              fim
9          fim
10         senão
11             S ∪ e;
12         fim
13     fim
14     se e_1.key > ivalue então
15         node = SearchInward(node → leftnode, i * c + dis − r);
16     fim
17     se o final da partição é encontrado então
18         node = nil;
19     fim
20     return(node);
21 fim
22 retorna node
```

Em (JAGADISH et al., 2005) são propostos métodos para escolha de pivôs, porém, por se tratarem de métodos baseados em espaços multidimensionais, eles não foram utilizados neste trabalho.

Algoritmo *Hull of Foci*

Proposto por (TRAINA et al., 2001; JR et al., 2007) o algoritmo *Hull of Foci* (HF) procura encontrar um bom conjunto de pivôs, com um custo *O(n)* para cálculos de distância. A Figura 20 procura representar de forma ilustrada o algoritmo para escolha dos pivôs. Inicialmente um objeto s_x é escolhido de forma aleatória, então é encontrado o objeto mais distante de s_x o qual será definido como o primeiro pivô (f_1), o próximo passo é encontrar o objeto mais distante de f_1, o qual será definido como o próximo pivô (f_2), a distância entre os dois pivôs ($d(f_1, f_2)$) será armazenada como *edge*. Os próximos pivôs que serão escolhidos deverão ter uma distância semelhante a f_1 e f_2, onde para cada

objeto s_i que ainda não pertence ao conjunto de pivôs, o seguinte erro é calculado:

$$error(s_i) = \sum_{g \in \mathfrak{F}} |edge - d(f_g, s_i)|. \tag{13}$$

O próximo pivô s_i escolhido deve apresentar o menor $error(s_i)$. Este último passo é repetido até atingir o número de pivôs necessários. O Algoritmo *HF* é apresentado em Algoritmo 7. O algoritmo não precisa analisar todo o conjunto de dados para encontrar a base *Omni-Foci* (os pivôs). Os autores mostram que o conjunto de pivôs pode ser escolhido a partir de uma amostra bem distribuída dos dados. Novos objetos podem ser inseridos ou removidos sem alterar a base *Omni-Foci*, apesar dos pivôs serem selecionados a partir do conjunto de dados, um pivô não precisa necessariamente fazer parte do conjunto de dados.

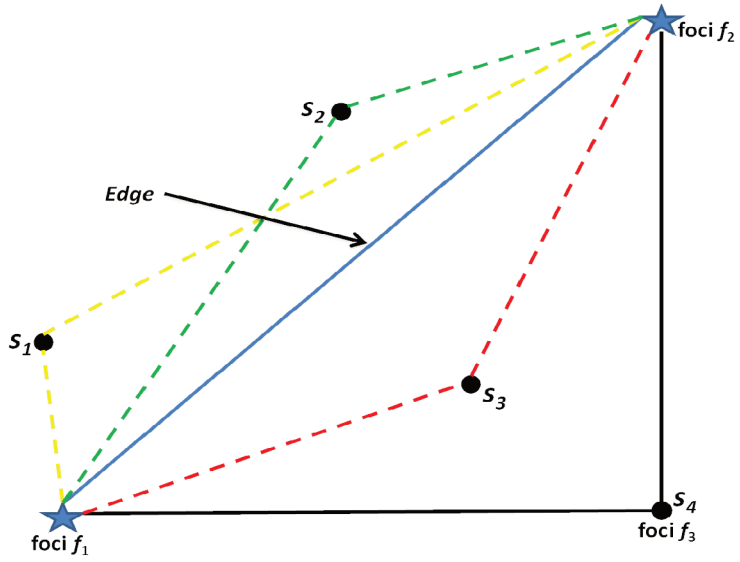

Figura 20 – Dado que f_1 e f_2 já foram escolhidos como pivôs (focos), o algoritmo seleciona o terceiro pivô entre s_1, s_2, s_3 e s_4. A melhor escolha é s_4

Maximum of Minimum Distances (MMD)

A função *Maximum of Minimum Distances* (MMD) é uma técnica incremental que seleciona os objetos que serão pivôs. Cada novo objeto selecionado está localizado o mais distante possível do anterior (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011). O algoritmo inicia escolhendo o primeiro pivô de forma aleatória, e o segundo passo é representado na Equação 14.

$$p_i = \frac{argmax}{s \in T - \{p_1, p_{i-1}\}} \min_{j=1}^{i-1} d(s, p_j). \tag{14}$$

Algoritmo 7: Algoritmo *HF* para encontrar a base *Omni-Foci*(pivôs) em conjunto de dados S

Entrada: A base de dados S e um h de *Foci*.
Saída: Conjunto de *Foci* $\mathfrak{F}$.

início
  Escolha de forma aleatória um objeto $s_i \in S$;
  Encontre o objeto f_1 mais distante de s_i;
  Insira f_1 em $\mathfrak{F}$;
  Encontre o objeto f_2 mais distante de s_i;
  Insira f_2 em $\mathfrak{F}$;
  set $edge \leftarrow d(f_1, f_2)$;
  enquanto *todos os focos não forem encontrados* **faça**
    para *cada objeto* $s_i \in S$, $s_i \neq \mathfrak{F}$ **faça**
      $errors_i \leftarrow \sum_g^{g \in \mathfrak{F}} |edge - d(f_g, s_i)|$;
      Selecione $s_i \in S$ onde $s_i \notin \mathfrak{F}$ e o $erro_i$ é o minimo;
      Insira s_i em $\mathfrak{F}$;
    fim
  fim
fim
retorna $\sigma(S)$

Maximum of Sum of Distances (MSD)

A função *Maximum of Sum of Distances* (MSD) (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) é uma técnica de seleção de pivôs muito semelhante à *MMD* apresentada na seção anterior. Assim como na *MMD* o algoritmo inicia escolhendo o primeiro pivô de forma aleatória, e o segundo passo é representado na Equação 15, mudando apenas que na *MMD* é escolhido o objeto s onde é possível se obter a menor distância $d(s, p_j)$ que maximiza a função, e no *MSD* é feito o somatório de $d(s, p_j)$ para escolher o objeto s que maximiza a função.

$$p_i = \frac{argmax}{s \in T - \{p_1, p_{i-1}\}} \sum_{j=1}^{i-1} d(s, p_j). \tag{15}$$

Sparce Spatial Selection (SSS)

Em (PEDREIRA; BRISABOA, 2007) é apresentado o método para escolha de pivôs *Sparce Spatial Selection* (SSS). Os autores citam como vantagem o fato desse método ser dinâmico, ou seja, em um fluxo contínuo de dados a distribuição do conjunto pode mudar, com isso pode ser necessária a criação de um novo pivô e essa é vantagem apontada pelos autores, da mesma forma pode ser que seja necessária a eliminação de um pivô.

O algoritmo 8 representa o pseudo código da implementação. A entrada do algoritmo é constituída pela base de dados S e uma variável α a qual os autores definem de forma

empírica. O algoritmo inicia encontrando a maior distância entre dois objetos que estão contidos no conjunto S. O conjunto de respostas (*Pivos*) inicia com o primeiro elemento s_1, então o conjunto S é percorrido validando se a distância do elemento s_i atual para cada elemento do conjunto *Pivos* é maior ou igual à maior distância entre dois objetos em S, se essa validação for verdadeira o objeto s_i atual passa a compor o conjunto *Pivos*.

Algoritmo 8: Algoritmo *SSS*, encontra os pivôs por meio do conjunto de dados S

Entrada: A base de dados S, variável peso α.
Saída: Conjunto de pivôs $Pivos$.

```
1  início
2      M← Max(d(s_i, s_j));
3      Pivos← s_1;
4      para cada objeto s_i ∈ S faça
5          se ∀p ∈ Pivos, d(s_i, p) ≥ Mα então
6              Pivos ← Pivos ⋃ s_i;
7          fim
8      fim
9  fim
10 retorna Pivos
```

O parâmetro α do algoritmo SSS depende da dimensionalidade e da distribuição do conjunto para ser definido. Como os autores não determinam como calculá-lo a não ser empiricamente ele não foi considerado nos experimentos.

2.6 Trabalhos Correlatos

Esta seção tem uma breve descrição sobre os trabalhos que serão usados como comparativos para o método proposto. Uma vez que o método proposto busca mapear objetos para n espaços unidimensionais, e armazenar essas informações em um Método de Acesso, para com isso tentar obter uma recuperação mais eficaz (com relação a tempo) dos objetos, é possível comparar os resultados com os métodos *Omni-B-Forest* (TRAINA et al., 2001; JR et al., 2007) e *iDistance* (JAGADISH et al., 2005).

Como o resultado do método proposto busca uma recuperação exata dos objetos, serão avaliados: tempo, número dos cálculos de distância e acesso ao disco.

Em (TRAINA et al., 2001; JR et al., 2007) é proposta uma família de métodos de acesso, baseados nos métodos de acesso já existentes, como as buscas sequenciais, $R{-}trees$ e $B{-}trees$.

A ideia principal da técnica apresentada em (TRAINA et al., 2001; JR et al., 2007), é eleger um grupo de objetos como focos (pivôs), e avaliar a distância de todos os outros objetos para este grupo. Esses focos servem para aumentar a poda no número dos cálculos de distâncias realizados durante um processo de consulta, além de indexar a distância de

cada objeto para o foco reduzindo as comparações realizadas por meio da desigualdade triangular.

As distâncias entre cada objeto e os focos são denominadas *Coordenadas Omni*. Essas coordenadas podem ser indexadas por meio de arquivo sequencial, $R-Tree$ ou $M-Tree$.

Experimentos realizados no trabalho proposto por (TRAINA et al., 2001; JR et al., 2007) mostram que o método obteve resultados iguais ou superiores a métodos existentes, sendo até dez vezes mais rápido e sendo capaz de realizar até dez vezes menos cálculos de distância e acesso ao disco.

O método *OmniB-Forest* apresentado em (TRAINA et al., 2001; JR et al., 2007) apresenta uma limitação na etapa de filtragem. Nessa etapa os autores propõem a utilização de uma estrutura auxiliar para armazenar os objetos que compõem a intersecção, após essa etapa é necessário consultar essa estrutura e recuperar em disco os objetos referentes aos índices salvos. Após recuperar esses objetos é realizado o cálculo da distância entre os objetos recuperados e o objeto de consulta gerando assim o conjunto de respostas.

A limitação do método se encontra no armazenamento e consulta das intersecções, essa abordagem acaba afetando tempo total gasto, tornando a consulta um pouco menos eficiente.

Em (JAGADISH et al., 2005) é proposto o algoritmo *iDistance*, um algoritmo para consultas KNN que pode ser adaptado para diferentes distribuições de dados. O primeiro passo da técnica proposta é particionar os dados e definir um ponto de referência (pivô) para cada partição. Logo em seguida a distância de cada objeto do conjunto de dados para o pivô da partição a qual ele pertence é indexada em uma estrutura B^+-Tree. A eficácia da técnica proposta depende de como os dados são particionados e como os pivôs são escolhidos.

Para realizar uma consulta KNN de um objeto q, é necessário realizar uma consulta por abrangência aplicando um raio r crescente que começa com o valor 0 e é incrementado até que resultado seja obtido. O Algoritmo converge quando os k objetos são encontrados.

Os autores comparam a técnica proposta com outros métodos presentes na literatura como a *M-Tree*, e a *Omni-Sequential*. Os resultados dos experimentos mostram que a técnica proposta superou as outras na maioria dos casos.

O método *iDistance* possuí uma limitação similar ao método *OmniB-Forest*. Esse método também necessita de uma estrutura auxiliar na etapa de filtragem, porém aqui a estrutura armazena os objetos localizados nas partições. Após a etapa de filtragem, os objetos referentes aos índices salvos são recuperados do disco, e a distância entre eles o objeto de consulta é computada gerando o conjunto de resposta.

A limitação desse método também se encontra na etapa de filtragem, onde é necessário o uso da estrutura auxiliar.

Capítulo 3

GroupSim

Na literatura existe uma variedade significativa de trabalhos voltados para recuperação de objetos utilizando medidas de dissimilaridade. Esses trabalhos, em sua maioria, utilizam métodos para indexação e recuperação de objetos, e este trabalho segue o mesmo principio.

A partir do conjunto de dados são escolhidos focos globais, que são denominados pivôs. Os objetos do conjunto de dados são mapeados para espaços unidimensionais por meio desses pivôs e indexados em uma única estrutura $B^{+}-tree$. A partir desse conceito se originou o nome da técnica: *GroupSim*: Um novo método para indexação de objetos utilizando mapeamentos unidimensionais baseados em focos globais, que será descrita em detalhes neste capítulo.

3.1 Mapeamentos Unidimensionais de Dados

Um mapeamento unidimensional de um conjunto de objetos em relação à um pivô é obtido a partir do cálculo da distância de cada objeto em relação à esse pivô, como apresentado na Figura 21.

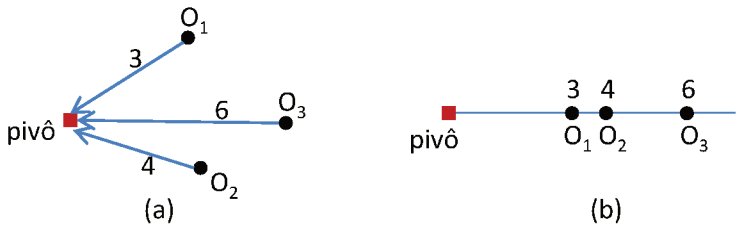

Figura 21 – Mapeamento unidimensional. (a) Pontos em um espaço bidimensional. (b) Mapeamento das distâncias para um espaço unidimensional.

Após a escolha de um conjunto de pivôs é possível criar mapeamentos unidimensionais em relação a cada pivô. Para criação dos vetores que representam os espaços unidimen-

sionais é necessário calcular a distância de cada objeto do conjunto de dados para cada objeto pertencente ao conjunto de pivôs, como apresentado na Figura 22.

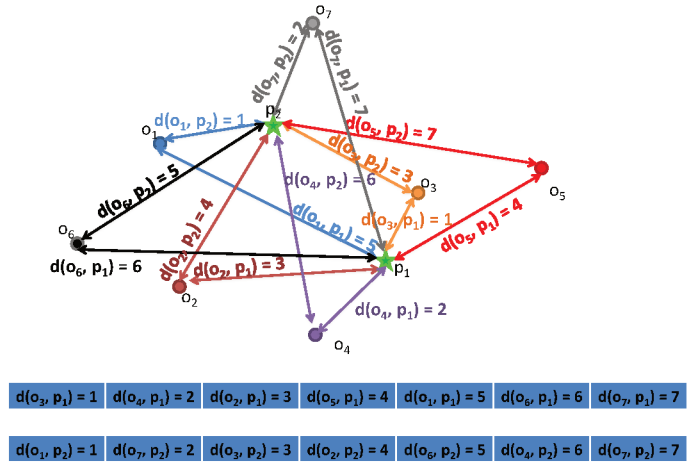

Figura 22 – Ilustração de um conjunto em um espaço bidimensional e os mapeamentos unidimensionais dos objetos com relação aos pivôs p_1 e p_2.

Na Seção 2.5.4.3 foram apresentados vários algoritmos para escolha de pivôs. O comportamento das estruturas criadas varia de acordo com a distribuição e natureza dos conjuntos de dados. Neste trabalho, os algoritmos apresentados foram avaliados comparando-se o tempo de execução, o número de acessos ao disco e o número de cálculos de distância para execução de consultas por similaridade.

3.2 Indexação dos Objetos

Uma consulta por similaridade pode ser efetuada com base na propriedade da desigualdade triangular pelo percurso em um mapeamento unidimensional. Como a chave de indexação é a distância em relação ao pivô tem-se uma relação de ordem total. O método de acesso B^+-*Tree* é uma estrutura de dados eficiente em armazenamento secundário para indexação de dados com relação de ordem.

A B^+-*Tree* é uma estrutura hierárquica que armazena todas as chaves nas folhas da estrutura que resulta em uma lista duplamente ligada de nós, tornando a busca por faixas de valores bastante eficiente. Em uma consulta por abrangência, por exemplo, com a lista sendo duplamente ligada e ordenada pela distância dos elementos para o pivô, o objeto com a menor distância para o elemento de consulta deve ser encontrado e, em seguida, os nós folhas devem ser percorridos para direita e esquerda recuperando as chaves com distância $\leq$ raio. A Figura 23, apresenta um exemplo de consulta em uma $B^+ - Tree$, com as folhas armazenadas em uma lista duplamente ligada. Na figura um objeto com

distância 4 para o pivô é definido como elemento de consulta e o raio de busca igual a 2. A consulta faz uma busca pela chave 4 na árvore e então percorre para ambos os lados para recuperar os identificadores dos objetos que possuem distância ≤ 2 (ou seja, as chaves de 2 a 6). Após recuperar os identificadores associados às chaves, os objetos devem ser recuperados da base de dados e as distâncias para o objeto de consulta devem ser calculadas para a inclusão dos objetos no conjunto resposta. Outra estratégia para a consulta por abrangência é encontrar a chave com distância igual à distância do objeto de consulta ao pivô menos o raio da consulta, e percorrer as folhas em ordem recuperando os identificadores até encontrar uma chave maior que a distância do objeto de consulta ao pivô mais o raio da consulta.

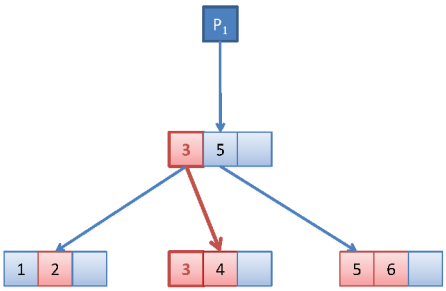

Figura 23 – Exemplo de consulta em uma $B^+ - Tree$, com raio igual a 2 e o objeto de consulta com distância 4 em relação ao pivô.

A indexação dos objetos é similar à $OmniB{-}Forest$ apresentada na Seção 2.5.4.1. Os objetos mapeados em n vetores (onde n é o número de pivôs), são indexados em n estruturas $B^+ - Tree$, sendo que em cada uma das estruturas os objetos são indexados ordenados pela distância para esse pivô. Na $OmniB{-}Forest$, ao computar a intersecção dos identificadores recuperados de cada mapeamento unidimensional permite filtrar os identificadores dos objetos que farão parte do conjunto resposta, reduzindo a quantidade de acessos aleatórios para recuperação dos objetos.

No método proposto, os objetos serão indexados em apenas uma estrutura $B^+ - Tree$, de forma que um dos pivôs é empregado para o mapeamento unidimensional dos objetos. Cada entrada no nó é composto pelo campo chave (distância de um objeto em relação ao pivô), o identificador do objeto e um vetor de distâncias dos mapeamentos unidimensionais para os outros $n-1$ pivôs. Um exemplo do nó é ilustrado na Figura 24. Com isso todos os vetores que foram criados a partir dos n pivôs são armazenados em uma única estrutura $B^+ - Tree$. Esse método foi denominado *GroupSim*.

O Algoritmo 9, apresenta como os objetos são indexados na estrutura proposta. O algoritmo recebe como entrada um conjunto de objetos S e um conjunto de pivôs P, e gera como saída a estrutura com os novos objetos do conjunto S indexados. Das linhas 2 à 7 é realizado o mapeamento dos objetos, onde é calculada a distância de todos os pivôs para todos os objetos, as linhas 8, 9 e 10 representam as informações relativas aos objetos

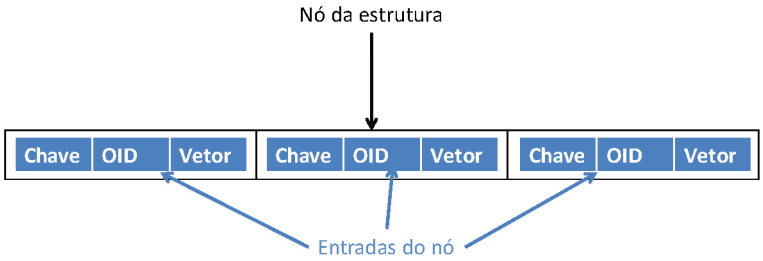

Figura 24 – Exemplos de nó e entrada de um nó presentes na estrutura *GroupSim*, onde a chave é a distância do objeto para o pivô p_0, o OID é o identificador do objeto, e o Vetor é onde as informações referentes aos outros ns pivôs são armazenadas.

sendo indexadas, onde são armazenadas as informações referentes à um objeto por vez, ou seja, é calculada a distância de um objeto o_i para todos os pivôs, e essas informações são armazenadas, isso se repete até que todos os objetos em S sejam percorridos, por se tratar de uma estrutura dinâmica o processo de indexação pode acontecer de forma contínua, ou seja, sempre que um novo conjunto S de objetos chegar ele pode ser indexado sem a necessidade de recriação da estrutura.

Algoritmo 9: *Indexação* NA GROUPSIM

Entrada: Conjunto de objetos S, conjunto de pivôs P .
Saída: Estrutura com novos objetos adicionados.

```
1  início
2      para Cada i = 0, onde i ≤ S.Tamanho faça
           // Etapa de mapeamento dos objetos=========================.
3          dist ← distância entre S_i e P_0;
4          vet_dist[quantidade_pivos − 1] ;
5          para Cada j = 1, onde j ≤ a quantidade de pivô faça
6              vet_dist[j] ← distância entre S_i e P_j;
7          fim
           // ========================================================.
           // Indexa as informações referentes ao mapeamento==========.
           // distância para o pivô zero, que é a chave para cada entrada
8          GroupSim ← dist;
           // Identificador do objeto
9          GroupSim ← OID_O_i;
           // Vetor com as distâncias para os outros n - 1 pivôs
10         GroupSim ← vet_dist;
           // ========================================================.
11     fim
12 fim
13 retorna GroupSim
```

3.3 Consultas por Similaridade

A recuperação dos objetos pode ser feita por meio de uma consulta por abrangência (*range query*) ou de uma consulta aos k-vizinhos mais próximos (*k-nearest neighbor query*). A recuperação baseia-se na propriedade da desigualdade triangular que permite a poda dos identificadores que certamente não farão parte do conjunto resposta.

Para recuperar os objetos são necessárias duas etapas. Na primeira etapa é necessário calcular a chave da consulta e realizar a busca da mesma na estrutura. Para isso, a distância do objeto *query* para o pivô P_0 é calculada, a estrutura *GroupSim* é percorrida até que o elemento com a chave mais próxima a distância calculada seja encontrado. A segunda etapa é responsável por formar e retornar o conjunto de respostas. Nesta etapa, é avaliado se as chaves dos objetos analisados estão dentro do intervalo (o intervalo é constituído pelo valor chave do objeto *query* subtraindo o raio e valor chave do objeto *query* somando o raio), se a validação for verdadeira então é calculada a distância do objeto referente a chave para o objeto *query*, se a distância for menor do que a de algum objeto pertencente ao conjunto de resposta, o último objeto é removido do conjunto e o objeto atual é inserido, ao término da validação das chaves o conjunto de resposta é retornado.

Foram desenvolvidos os algoritmos de consulta por abrangência e por vizinhos mais próximos para a estrutura *GroupSim*, apresentados a seguir.

3.3.1 Consulta por Abrangência (*Range Query*)

Para uma consulta por abrangência, primeiramente, é preciso localizar a chave que possui valor o qual está mais próximo possível de $d(q, p_0) - r$, onde q é o objeto de consulta, p_0 é o primeiro pivô do conjunto e r é o raio da consulta. Depois que localizar a chave os nós folha da estrutura são percorridos até encontrar uma chave maior que $d(q, p_0) + r$. Todas as chaves que estiverem no intervalo $d(q, p_0) - r$ e $d(q, p_0) + r$ devem ser verificadas para saber se entram no conjunto de resposta. Para verificar se há interseção é preciso validar para os $n - 1$ pivôs restantes se $d(o_i, p_j) \leq d(q, p_j) + r$, onde o_i é uma chave da estrutura, e p_j faz parte do conjunto de pivôs $P = \{p_{j+1}, p_{j+2}, ..., p_n\}$.

A distância $d(o_i, p_j)$ está localizada no vetor que compõe uma entrada do nó, se todos os valores do vetor de um nó referente a uma chave o_i, forem $\leq d(q, p_j) + r$, então, possivelmente o objeto referente a chave que esta sendo verificada faz parte da resposta. Para confirmar é necessário buscar o objeto referente a chave na base de dados, depois calcular se $d(o, q) \leq r$, com esta afirmação sendo verdadeira o objeto entra para o conjunto de resposta. O Algoritmo 10 exibe os passos de como a *range query* pode ser implementada.

Algoritmo 10: CONSULTA *range query* EM UMA *GroupSim*

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs P, objeto de consulta q e raio r.
Saída: Conjunto de Resposta.

```
1  início
       // Buscar e apontar para chave mais próxima de d(q, P_0) − r.
2      lista ← GB.abrir_cursor_proximo(d(q, P_0) − r);
       /* retorna a proxima chave a direita (para chaves ordenadas de
          forma crescente)                                          */
3      cursor_dist ← lista.proxima_chave();
4      flag ← true;
       // Executa o laço de d(q, P_0) − r até d(q, P_0) + r
5      enquanto cursor_dist ≤ d(q, P_0) + r faça
6          para i ← 1 até tamanho pivos faça
7              se cursor_dist ≤ d(P_i, q) − r e cursor_dist ≥ d(P_i, q) + r então
8                  flag ← true;
9              fim
10             senão
11                 flag ← false;
12             fim
13         fim
14         se flag então
15             OID ← lista.getOID();
16             objeto ← S.getObjeto(OID);
17             dist ← d(q, objeto);
18             se dist ≤ r então
19                 δ ← δ + objeto;
20             fim
21         fim
22         cursor_dist ← lista.proxima_chave();
23     fim
24 fim
25 retorna Conjunto de respostas δ
```

3.3.2 Consulta aos k-vizinhos mais próximos (*k-nearest neighbor query*)

A consulta *knn* utilizando a estrutura proposta é mais complicada do que a consulta *range query*. Os Algoritmos 11 e 12, exibem os passos de como uma consulta *knn* pode ser implementada. O primeiro passo é apontar para o elemento chave, posicionando um ponteiro para a chave que estiver o mais próximo possível de $d(q, p_0)$, onde q é o objeto *query* e p_0 é o primeiro pivô do conjunto. Mesmo a consulta sendo *KNN* é necessário definir um raio o qual inicia com um valor ∞, a medida que o algoritmo converge, o valor do raio é reajustado até que os k elementos mais próximos de q sejam encontrados.

Enquanto na consulta *range query* o ponteiro é posicionado em $d(q, p_0) - r$ e percorre

a lista da esquerda para direita até chegar em $d(q, p_0) + r$, na consulta *KNN* o ponteiro é posicionado em $d(q, p_0)$ e a lista é percorrida para ambos os lados ao mesmo tempo, olhando para o elemento a esquerda e verificando se ele pertence ao conjunto resposta, depois olhando para o elemento a direita também verificando se ele pertence ao conjunto resposta.

Para saber se o objeto referente a entrada que está sendo analisada faz parte do conjunto resposta, é necessário verificar se os elementos do vetor auxiliar pertencente a entrada relacionada a chave atual satisfazem a condição: $d(o_i, p_j) > d(q, p_0) - raio$ **e** $d(o_i, p_j) < d(q, p_0) + raio$, onde o_i é uma chave dentro do intervalo atual, e p_j é um pivô pertencente ao conjunto $P = \{p_{j+1}, p_{j+2}, ..., pn\}$. Se a condição for verdadeira, então possivelmente o objeto referente à chave que esta sendo verificada faz parte da resposta. Para confirmar é necessário buscar o objeto referente a chave na base de dados, depois calcular se $d(o, q) \leq r$, com esta afirmação sendo verdadeira o objeto entra para o conjunto atual de resposta. Quando o conjunto de resposta atinge tamanho igual $k + 1$, então é necessário remover o maior elemento do conjunto e atualizar o raio para $d(o_k, p_0)$, ou seja, a distância do último objeto do conjunto de respostas para o pivô da estrutura *GroupSim*.

Com o raio atualizado, tem-se uma maior região de poda e pode-se dizer que de certa forma a consulta se tornou do tipo '*abrangência*'. Quanto menor o raio, menor o intervalo $d(q, p_i) - raio$ e $d(q, p_i) + raio$, tornando a quantidade de acesso ao disco e cálculos de distância menores.

Algoritmo 11: CONSULTA *KNN* EM UMA *GroupSim*

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs p, objeto de consulta q e o número k de objetos que deseja recuperar.

Saída: Conjunto de Resposta.

```
1  início
       // Buscar e apontar para chave mais próxima de d(q, p_0).
2      lista ← GB.abrir_cursor_proximo(d(q, p_0));
       /* A lista será percorrida tanto para direita quanto para a
          esquerda, para isso é necessário criar duas variáveis para
          controlar quando se chega no inicio ou final da lista.      */
3      continue_direita_eof ← true, continue_esquerda_bof ← true;
4      raio ← ∞;
       /* Como o raio começa valendo infinito é necessário adotar outro
          critério de parada além de chegar ao inicio ou fim da lista,
          para isso são criadas duas variaveis.                       */
5      dist_continue_direita ← ∞, dist_continue_esquerda ← ∞;
       /* Mesmo que tenha chegado no final ou inicio da lista, pode ser
          necessário continuar pesquisando, pois pode ainda não ter
          encontrado os k objetos, para isso é necessário mais duas
          variaveis, essas variaveis também são usadas como critério de
          parada.                                                     */
6      continue_direita ← true, continue_esquerda ← true;
       /* variavel que controla se o objeto esta dentro do raio de
          consulta para todos os pivôs                                */
7      interseção ← true;
8      enquanto continue_direita ou continue_esquerda faça
9          δ ← Algoritmo 12;
10         se continue_esquerda então
11             continue_esquerda_bof ←!B.bof;
12         fim
13         se continue_direita então
14             continue_direita_eof ←!B.eof;
15         fim
16         dist_continue_direita ← B.dist_direita();
17         dist_continue_esquerda ← B.dist_esquerda();
           // se o raio ainda não foi atingido continuar andando
18         se dist_continue_esquerda < d(q, p_0) − raio ou continue_esquerda_bof
            então
19             continue_esquerda ← true;
20         fim
21         se dist_continue_direita < d(q, p_0) − raio ou continue_direita_eof
            então
22             continue_direita ← true ;
23         fim
24     fim
25 fim
26 retorna Conjunto de respostas δ
```

Algoritmo 12: VERIFICA SE UM OBJETO PERTENCE A CONSULTA

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs p, objeto de consulta q e o número k de objetos que deseja recuperar.

Saída: Conjunto de Resposta.

```
1  início
2      se continue_direita for verdadeiro então
3          OID ← GB.elemento_direita_chave;
4          interseção ← true;
5          para i←1 até número de pivôs faça
6              se GB.dist(p_i) < d(q, p_i) − raio ou GB.dist(p_i) > d(q, p_i) + raio então
7                  interseção ← false;
8              fim
9          fim
10     fim
11     se interseção for verdadeiro então
12         objeto ← S.getObjeto(OID);
13         dist ← d(objeto, q);
14         δ.add(Objeto, d);
15         se δ.tamanho > k então
16             remove objetos com maior distância em δ;
17             raio ← ultima distância de δ (maior distância);
18         fim
19     fim
20     se continue_esquerda for verdadeiro então
21         OID ← GB.elemento_direita_chave;
22         interseção ← true;
23         para i←1 até número de pivôs faça
24             se GB.dist(pivo_i) < d(q, p_i) − raio ou GB.dist(p_i) > d(q, p_i) + raio
                então
25                 interseção ← false;
26             fim
27         fim
28     fim
29     se interseção for verdadeiro então
30         objeto ← S.getObjeto(OID);
31         dist ← d(objeto, q);
32         δ.add(Objeto, d);
33         se δ.tamanho > k então
34             remove objetos com maior distância em δ;
35             raio ← ultima distância de δ (maior distância);
36         fim
37     fim
38 fim
39 retorna Conjunto de respostas δ
```

Capítulo 4

Experimentos e Análise dos Resultados

Este capítulo busca apresentar de forma clara e objetiva os experimentos que foram realizados e a análise dos resultados obtidos. Foram feitos diversos testes a partir de diferentes bases de dados, com uma análise comparativa dos resultados obtidos. Esta análise busca avaliar o desempenho da metodologia proposta quando confrontada com outros trabalhos consolidados na literatura.

4.1 Método para a Avaliação

Para se explorar de forma eficaz os métodos propostos neste trabalho, foram realizados experimentos em uma máquina com sistema operacional Linux 64 bits (Ubuntu 14.04), com 8 GB de memória principal, processador *Intel®Core*™ modelo $i5-2310$ de 2.90 GHz e disco rígido de 1 TB.

Uma das bases de dados utilizadas foi obtida a partir do *Content-based Photo Image Retrieval Test-Collection* (BOLETTIERI et al., 2009), que é uma base com imagens oriundas do *Flikr*, contendo dezenas de milhões de arquivos. Os arquivos são formados pelos descritores visuais das imagens, e divididos em *Color Structure*, *Scalable Color*, *Color Layout*, *Edge Histogram* e *Homogeneus Texture*. A característica escolhida para teste foi a *Color Structure* composta de 64 dimensões, e foram selecionadas características de um milhão de imagens.

Também foram utilizados para avaliação os conjuntos de dados: *Colors* (K.FIGUEROA; G.NAVARRO; E.CHÁVEZ, 2007) com 112.682 imagens e 112 dimensões; o conjunto *Corel* (LICHMAN, 2013) com 68.040 vetores de 32 dimensões; *Eigenfaces* (TURK; PENTLAND, 1991) com 11.900 vetores de 16 dimensões; *Nasa* (K.FIGUEROA; G.NAVARRO; E.CHÁVEZ, 2007) com 40.150 vetores de 20 dimensões. Além desses conjuntos foram geradas duas bases sintéticas: a primeira contém agrupamentos com distribuição uniforme em 150.000 instâncias de 16 dimensões denominada *Clusters*; e a segunda formada por pontos que representam os vértices de um triângulo de *Sierpinski* em duas dimensões

mais quatro dimensões com correlações não lineares ($x + y^2$, $\sqrt{5x}$, $log_{10}(x + y)$, $x^2 + y$), resultando em 531.441 instâncias com 6 dimensões.

O método de acesso *GroupSim* foi avaliado nos experimentos apresentados a seguir em comparação aos métodos de acesso sequencial, $OmniB{-}Forest$ e *IDistance*, levando em consideração o tempo de execução, o número de acessos ao disco e o número de cálculos de distância.

As funções de distância (métricas) utilizadas foram: Euclidiana e a *FastEMD*. Os experimentos com as duas funções de distância permitem avaliar dois cenários diferentes:

- um cenário onde uma quantidade grande de cálculos de distância tem execução mais rápida que um acesso a disco (como ocorre com a distância Euclidiana);
- um cenário onde uma pequena quantidade de cálculos de distância tem execução mais lenta que um acesso a disco (como ocorre com a distância FastEMD).

A distância FastEMD apresenta complexidade de tempo de execução $O(N^2)$ com relação ao número de dimensões. A Tabela 1 apresenta os tempos de execução de 10.000 cálculos de distância com as funções Euclidiana e EMD quando os vetores têm de 8 a 256 dimensões. É possível notar que o tempo de processamento da distância Euclidiana tem crescimento linear com o aumento da dimensionalidade enquanto a *FastEMD* tem crescimento quadrático.

Tabela 1 – Tempo para execução de 10.000 cálculos de distância.

Função de distância	Tempo em segundos					
	8 dim	16 dim	32 dim	64 dim	128 dim	256 dim
FastEMD	0.060579	0.276256	1.505861	9.501836	74.514578	577.575193
Euclidiana	0.001396	0.002486	0.004498	0.008752	0.017139	0.033920

Os objetos de cada conjunto de dados foram indexados utilizando os algoritmos: $OmniB{-}Forest$, *GroupSim*, *iDistance* e Acesso Sequencial. Foram feitas consultas kNN com k variando entre 10 e 100, utilizando as funções de distância Euclidiana e *FastEMD*.

Em cada experimento é possível avaliar o número médio de acessos ao disco, o número médio dos cálculos de distância realizados, e o tempo médio gasto para cada quantidade k de elementos retornados nas consultas. Para cada conjunto de dados foram selecionados aleatoriamente 100 objetos de consulta.

A Tabela 2 apresenta de modo sintético os conjuntos de dados utilizados nos experimentos e suas principais características, bem como os parâmetros empregados de número de pivôs e algoritmo de seleção de pivôs. Os dados apresentados na tabela foram obtidos por meio de alguns experimentos iniciais, foram realizados experimentos com o objetivo de avaliar qual o número de pivôs adotar e qual o melhor método para escolha dos pivôs, ou seja, os experimentos serviram para determinar qual a configuração ideal para cada conjunto de dados, e são apresentados no Apêndice A.1.

Tabela 2 – Conjuntos de dados e parâmetros de execução.

Conjunto	Número objetos	Número dimensões	Seleção pivôs	Número de pivôs	Métrica
Colorstructure	1.000.000	64	MMD	4	L_2
Colors	112.682	112	aleatório	4	L_2
Corel	68.040	32	aleatório	4	L_2
Eigenfaces	11.900	16	MMD	4	L_2
Nasa	40.150	20	MSD	4	L_2
Clusters	150.000	16	MMD	4	L_2
Sierpinski	531.441	6	MMD	4	L_2
Covertype	4.000.000	42	MMD	4	L_2
Kdd Cup	581.012	54	MMD	4	L_2
Corel	68.040	32	aleatório	4	FastEMD
Eigenfaces	11.900	16	aleatório	4	FastEMD
Nasa	40.150	20	MMD	4	FastEMD
Clusters	150.000	16	Hull of Foci	4	FastEMD
Sierpinski	531.441	6	MMD	4	FastEMD

As Figuras 25 à 38 apresentam os gráficos dos experimentos utilizando as funções de distância Euclidiana e FastEMD. Em cada um dos experimentos a seguir a função para escolha dos pivôs juntamente com quantidade de pivôs que apresentaram melhores resultados nos experimentos apresentados pelo Apêndice A.1 foram utilizados e descritos.

Os testes utilizando as base de dados *Colorstructure* e *Colors* foram feitos somente para a função de distância Euclidiana, como a base de dados *Colorstructure* possui uma quantidade significativa de objetos e a *Colors* tem alta dimensionalidade o tempo de execução para realizar os experimentos com a função de distância FastEMD se tornou inviável.

Os experimentos apresentados foram realizados com o objetivo de avaliar os algoritmos para indexação e recuperação de objetos. As Figuras 25 à 31 apresentadas na Seção 4.2 são referentes aos experimentos utilizando a função de distância Euclidiana e as Figuras 34 à 38 apresentadas na Seção 4.3 são referentes aos experimentos realizados utilizando a função de distância FastEMD.

Para um melhor entendimento sobre os experimentos realizados deve-se levar em consideração que para os métodos *iDistance*, *OmniB-Forest* e *GroupSim*, foram computadas as quantidades de acesso ao disco para a estrutura B^+-Tree, ou seja, quantas vezes foi necessário acessar o disco para recuperar uma entrada armazenada em um nó da estrutura.

Para o acesso sequencial são computados os acessos ao disco necessários para recuperar os objetos não apenas uma referência para os mesmos. Para os três métodos baseados na estrutura B^+-Tree, os gráficos de acesso ao disco apresentam apenas a quantidade dos acessos necessários para recuperar os identificadores dos objetos, os acessos não sequenciais que foram realizados para recuperação dos objetos não são computados, porém refletem no tempo total gasto para realizar uma consulta.

4.2 Experimentos com a distância Euclidiana

A Figura 25 exibe os gráficos dos experimentos utilizando 4 pivôs para indexação escolhidos por meio da função *Maximum of Minimum Distance*. Nesse experimento foi usada a base de dados *Colorstructure* e o melhor resultado obtido foi com o Acesso Sequencial.

Na Figura 25 (a) são apresentado os resultados de acesso ao disco, o algoritmo *iDistance* é o que apresenta melhor resultado, realizando a menor quantidade de acesso ao disco para as consultas. O algoritmo de Acesso Sequencial foi o que apresentou pior desempenho.

A Figura 25 (b) apresenta o desempenho dos algoritmos com relação a quantidade dos cálculos de distância realizados, o algoritmo que apresentou os melhores resultados foi o *GroupSim* com o Acesso Sequencial apresentando os piores resultados.

A Figura 25 (c) apresenta o desempenho dos algoritmos com relação ao tempo médio gasto nas consultas. Nesse gráfico é possível notar que apesar de o Acesso Sequencial apresentar os piores resultados com relação ao número de acessos ao disco realizados e a quantidade dos cálculos de distância efetuados, ele obteve o melhor tempo. A explicação para isso ocorrer é que como os algoritmos que estão sendo usados na comparação realizam um acesso aleatório ao disco, e o algoritmo sequencial acessa do primeiro ao último objeto sequencialmente faz com que obtenha melhor desempenho.

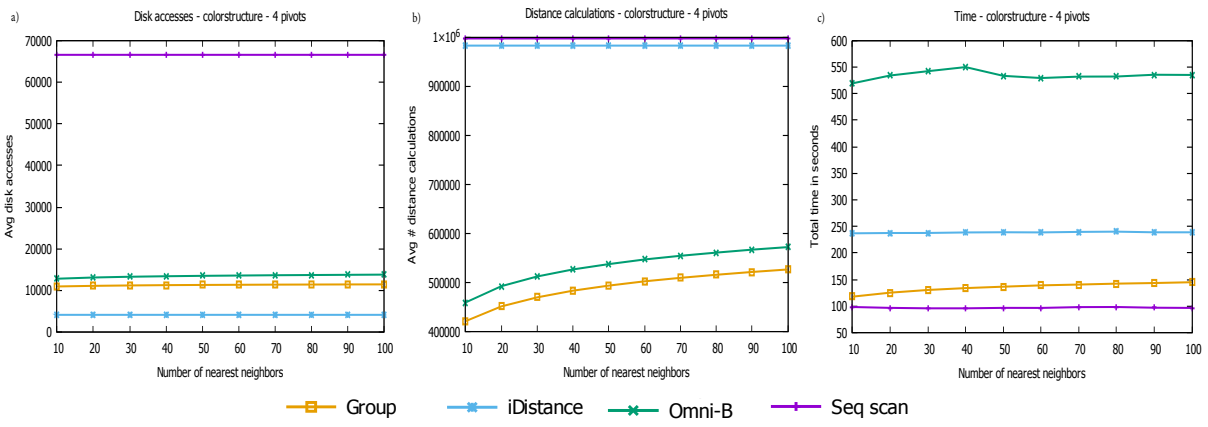

Figura 25 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Colorstructure*.

A Figura 26 exibe os gráficos dos experimentos utilizando 4 pivôs para indexação escolhidos de maneira aleatória. Nesse experimento foi utilizada a base de dados *Colors* e o melhor resultado obtido foi com o algoritmo *GroupSim*.

A Figura 26 (a) exibe informações referentes a quantidade de acessos ao disco realizados por meio de cada função adotada. Para esse experimento o algoritmo *iDistance*

apresenta melhor desempenho, o algoritmo *GroupSim* também tem um bom desempenho com sua curva posicionada muito próxima à do algoritmo *iDistance*.

Na Figura 26 (b) é apresentado o gráfico de desempenho dos algoritmos com relação a quantidade dos cálculos de distância realizado. Nesse gráfico é possível notar que o algoritmo *GroupSim* obteve o melhor desempenho seguido pelo algoritmo $OmniB-Forest$. Aqui o algoritmo *iDistance* obteve um desempenho quase 5 vezes pior do que o *GroupSim*.

A Figura 26 (c) apresenta o desempenho dos algoritmos com relação ao tempo médio gasto nas consultas. É possível notar que o algoritmo *GroupSim* apresentou o melhor desempenho e, apesar dos algoritmos *iDistance* e $OmniB-Forest$ apresentarem melhor desempenho do que o Acesso Sequencial com relação ao número dos cálculos de distância efetuados e quantidade dos acessos ao disco realizados, eles têm um pior desempenho com relação a tempo.

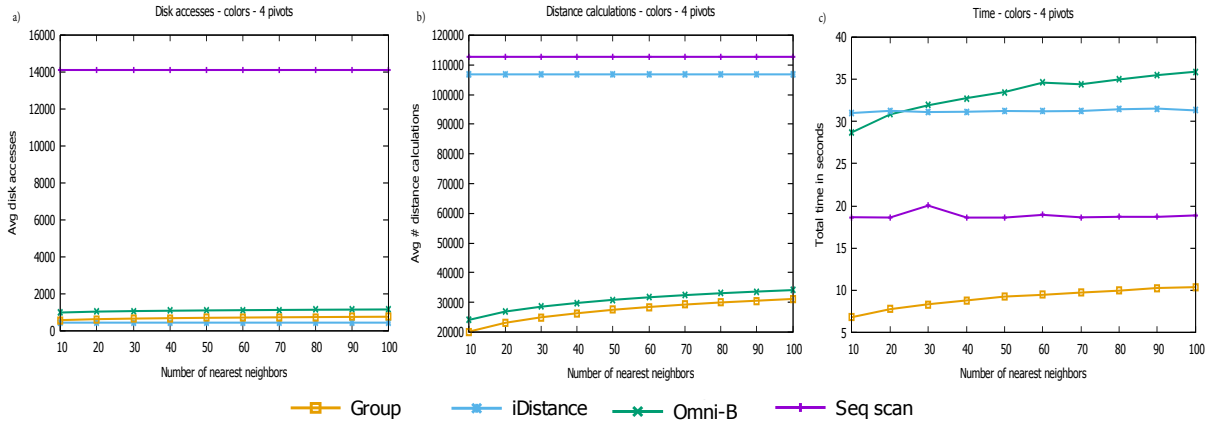

Figura 26 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Colors*.

A Figura 27 exibe os gráficos dos experimentos utilizando 4 pivôs para indexação escolhidos de maneira aleatória. Nesse experimento foi utilizada a base de dados *Corel* e o melhor resultado obtido foi com o algoritmo de Acesso Sequencial.

A Figura 27 (a) exibe informações referentes a quantidade de acessos ao disco realizados por meio de cada função adotada. Para esse experimento o algoritmo *iDistance* apresenta melhor desempenho, o algoritmo *GroupSim* também tem um bom desempenho.

Na Figura 27 (b) é apresentado o gráfico de desempenho dos algoritmos com relação a quantidade dos cálculos de distância realizado. Nesse gráfico é possível notar que o algoritmo $OmniB-Forest$ obteve o melhor desempenho seguido pelo algoritmo *GroupSim*. Os algoritmos *iDistance* e Acesso sequencial obtiveram desempenho similar, realizando praticamente 7 vezes mais cálculos de distância do que o algoritmo $OmniB-Forest$.

A Figura 27 (c) apresenta o desempenho dos algoritmos com relação ao tempo médio gasto nas consultas. É possível notar que o algoritmo *GroupSim* apresenta melhor de-

sempenho para um valor baixo de vizinhos recuperados, porém a medida que o número de vizinhos recuperados aumenta o algoritmo *GroupSim* sofre uma perda de desempenho, tornando assim o Acesso Sequencial que tem uma curva linear o algoritmo de melhor desempenho. Apesar dos algoritmos *iDistance* e $OmniB-Forest$ apresentarem melhor desempenho do que o Acesso Sequencial com relação ao número dos cálculos de distância efetuados e quantidade dos acessos ao disco realizados eles têm um pior desempenho com relação ao tempo de execução.

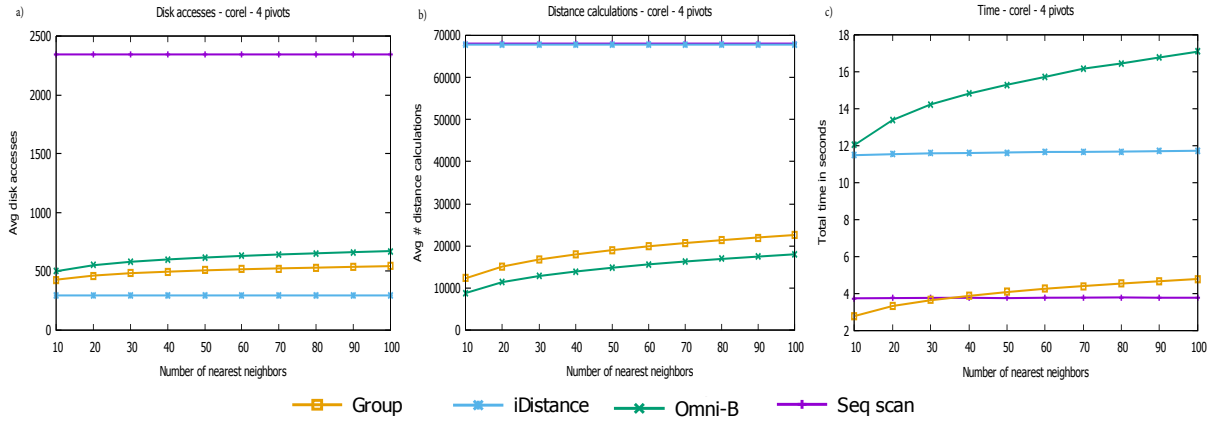

Figura 27 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Corel*.

A Figura 28 exibe os gráficos dos experimentos realizados utilizando a base de dados *Eigenfaces*. Nesses experimentos foram utilizados 4 pivôs escolhidos por meio da função *Maximum of Minimum Distance*, e o melhor desempenho obtido para indexação e recuperação dos objetos foi utilizando o algoritmo *GroupSim*.

A Figura 28 (a) apresenta o algoritmo *GroupSim* como o método que realiza a menor quantidade de acesso ao disco. Os algoritmos $OmniB-Forest$ e *iDistance* apresentam um desempenho razoável.

Na Figura 28 (b) é apresentado o gráfico de desempenho dos algoritmos com relação a quantidade de cálculos de distância realizado. Nesse gráfico os algoritmos *GroupSim* e $OmniB-Forest$ apresentam um desempenho muito similar, sendo que os dois algoritmos realizam as menores quantidades de cálculos de distância dentre os quatro avaliados. O algoritmo de Acesso Sequencial é o que apresenta pior desempenho.

A Figura 28 (c) apresenta o desempenho dos algoritmos com relação ao tempo médio gasto nas consultas. É possível notar que o algoritmo *GroupSim* apresenta melhor desempenho, seguido pelo algoritmo Acesso Sequencial.

A Figura 29 apresenta os gráficos dos experimentos que foram realizados utilizando a base de dados *Nasa*. Para esse experimento foram selecionados 4 pivôs utilizando o algoritmo *Maximum of Sum of Distances*.

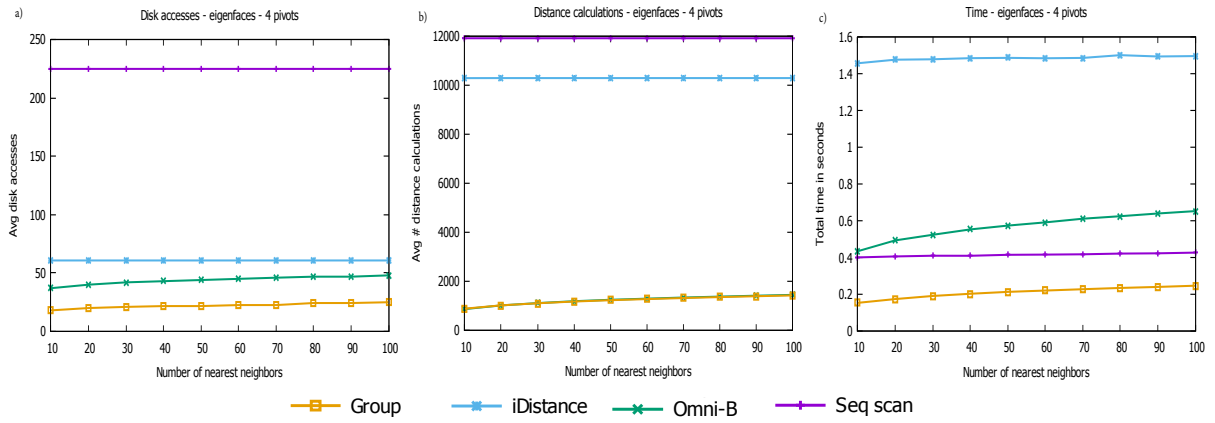

Figura 28 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Eigenfaces*.

Na Figura 29 (a) o algoritmo *iDistance* obteve o melhor desempenho realizando o menor número de acessos ao disco. O algoritmo com de pior desempenho foi o Acesso Sequencial.

A Figura 29 (b) é referente ao número dos cálculos de distância realizados. O algoritmo $OmniB-Forest$ foi o que apresentou melhor resultado, com o algoritmo *GroupSim* tendo desempenho muito próximo à ele. Os algoritmos *iDistance* e Acesso Sequencial ambos obtiveram o pior desempenho.

Na Figura 29 (c) é possível notar que o algoritmo Acesso Sequencial obteve melhor desempenho, porém analisando os gráficos (a) e (b) pode-se notar que ele obteve o pior desempenho em ambos experimentos. O fato desse algoritmo apresentar um bom desempenho em (c) está diretamente ligado ao fato dele acessar os objetos armazenados em disco de forma sequencial, enquanto os outros algoritmos acessam de forma aleatória.

A Figura 30 apresenta os gráficos dos experimentos que foram realizados utilizando a base de dados *Clusters*. Para esse experimento foram selecionados 4 pivôs utilizando o algoritmo *Maximum of Minimum Distances*.

Na Figura 30 (a) o algoritmo *iDistance* obteve o melhor desempenho realizando o menor número de acessos ao disco. O algoritmo com de pior desempenho foi o Acesso Sequencial.

A Figura 30 (b) é referente ao número dos cálculos de distância realizados. O algoritmo *GroupSim* foi o que apresentou melhor resultado, com o algoritmo $OmniB-Forest$ tendo desempenho muito próximo à ele. Os algoritmos *iDistance* e Acesso Sequencial ambos obtiveram o pior desempenho.

Na Figura 30 (c) é possível notar que o algoritmo Acesso Sequencial obteve melhor desempenho, porém analisando os gráficos (a) e (b) foi mostrado que ele teve o pior desem-

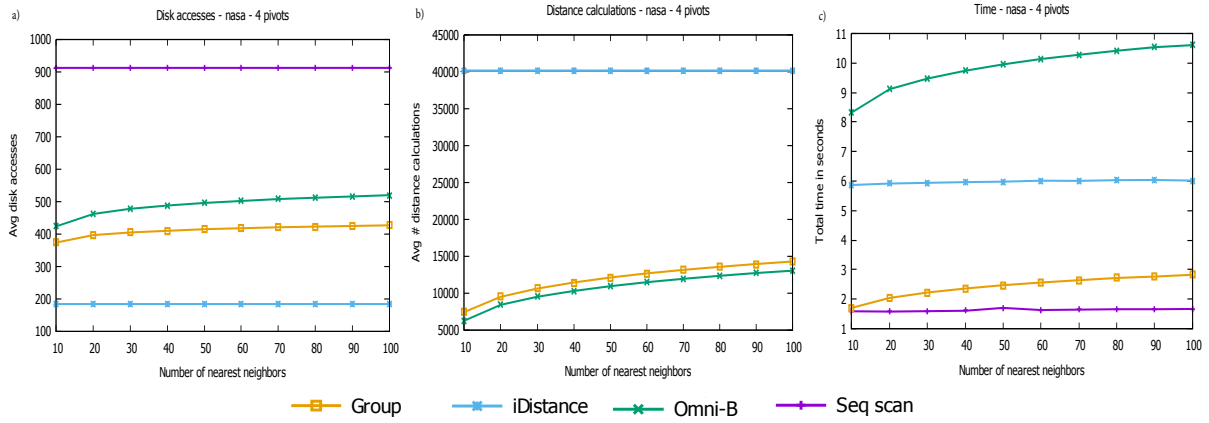

Figura 29 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , *OmniB−Forest*, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Nasa*.

penho em ambos experimentos, o fato desse algoritmo apresentar um bom desempenho em (c) está diretamente ligado ao fato dele acessar os objetos armazenados em disco de forma sequencial, enquanto os outros algoritmos acessam de forma aleatória.

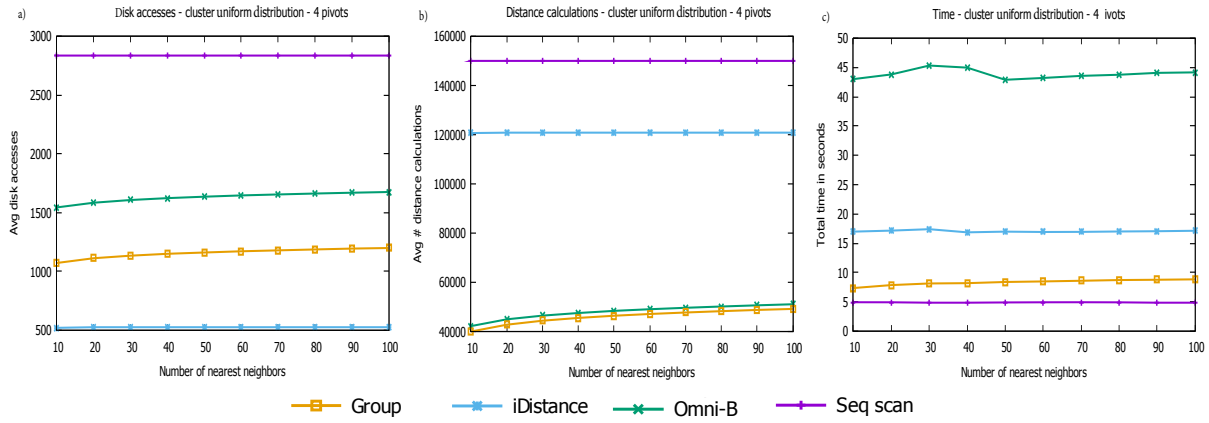

Figura 30 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , *OmniB−Forest*, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Cluster*.

Na Figura 31 os gráficos exibem informações referentes experimentos realizados com a base de dados *Siepinski*, utilizando 4 pivôs escolhidos por meio do algoritmo *Maximum of Minimum Distances*.

A Figura 31 (a) apresenta o gráfico com os resultados dos experimentos realizados para avaliar a quantidade de acesso ao disco, observando esse gráfico é possível notar o algoritmo *GroupSim* apresentou o melhor desempenho, com o algoritmo *OmniB−Forest*

obtendo um desempenho muito próximo à ele. O algoritmo Acesso Sequencial foi o que apresentou o pior desempenho.

A Figura 31 (b) apresenta o desempenho dos algoritmos com relação ao número dos cálculos de distância realizados. Os algoritmos *GroupSim* e $OmniB-Forest$ apresentaram o melhor desempenho, ambos os algoritmos possuem uma curva muito próxima a zero. O Algoritmo Acesso Sequencial aqui também apresenta o pior desempenho.

A Figura 31 (c) apresenta o gráfico do desempenho dos algoritmos com relação ao tempo total médio gasto para as consultas realizadas. O algoritmo *GroupSim* foi o que apresentou melhor desempenho gastando um tempo médio muito próximo a zero para as consultas realizadas. O Algoritmo *iDistance* foi quem apresentou pior desempenho.

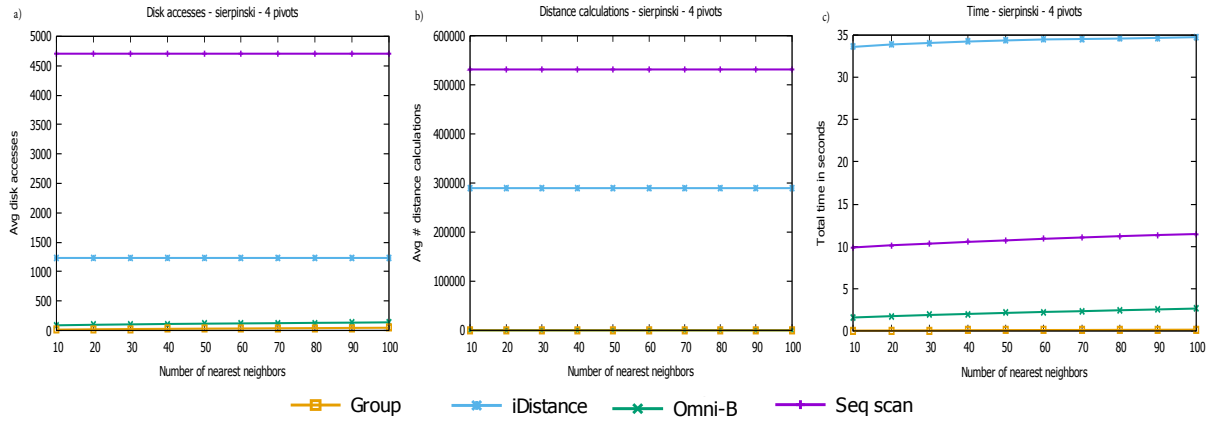

Figura 31 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Sierpinski*.

Os experimentos apresentados nas Figuras 32 e 33 são referentes as bases de dados *Covertype Data Set* (BLAKE; MERZ, 1998) e *KDD Cup 1999 Data Data Set* (CUP, 1999). A base de dados *Covertype* possui 4.000.000 com 42 dimensões e a base de dados *KDD* possui 581.012 atributos com 54 dimensões. Os experimentos para essas bases de dados foram os últimos experimentos realizados, e baseando no conhecimento obtidos com experimentos anteriores, foi adotada a configuração de 4 pivôs escolhidos por meio da função *Maximum of Minimum Distances*, utilizando a medida de dissimilaridade Euclidiana.

Na Figura 32 os gráficos exibem informações referentes experimentos realizados com a base de dados *Covertype*.

A Figura 32 (a) apresenta o gráfico com os resultados dos experimentos realizados para avaliar a quantidade de acesso ao disco, observando esse gráfico é possível notar o algoritmo *iDistance* apresentou o melhor desempenho, com o algoritmo *GroupSim* obtendo um desempenho muito próximo à ele. O algoritmo Acesso Sequencial foi o que apresentou o pior desempenho.

A Figura 32 (b) apresenta o desempenho dos algoritmos com relação ao número dos cálculos de distância realizados. Os algoritmos *GroupSim* e $OmniB-Forest$ apresentaram o melhor desempenho, ambos os algoritmos possuem uma curva bastante similar. O Algoritmo Acesso Sequencial aqui também apresenta o pior desempenho.

A Figura 32 (c) apresenta o gráfico do desempenho dos algoritmos com relação ao tempo total médio gasto para as consultas realizadas. O algoritmo *GroupSim* foi o que apresentou melhor desempenho gastando um tempo médio muito próximo a zero para as consultas realizadas. O Algoritmo $OmniB-Forest$ foi quem apresentou pior desempenho para este experimento.

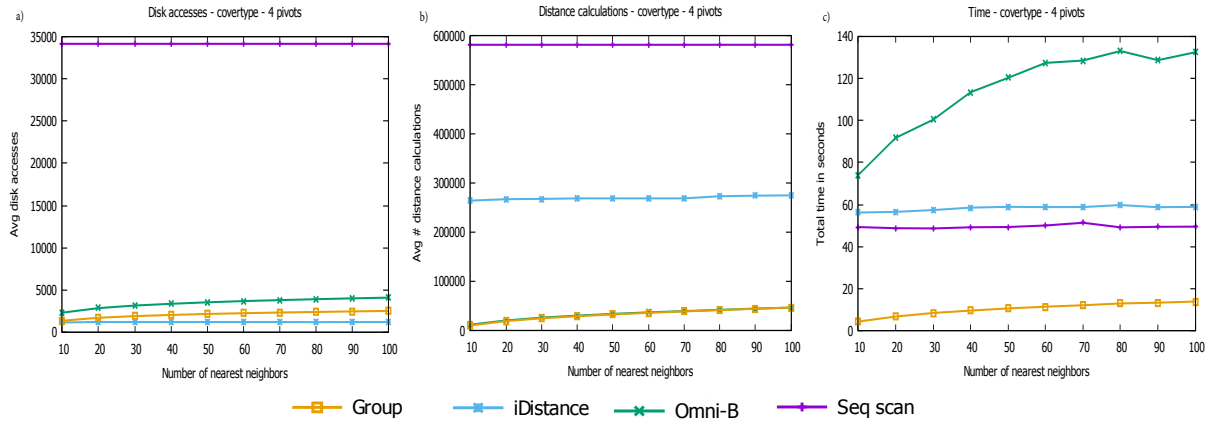

Figura 32 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Covertype*.

Na Figura 33 os gráficos exibem informações referentes experimentos realizados com a base de dados *KDD Cup*.

A Figura 33 (a) apresenta o gráfico com os resultados dos experimentos realizados para avaliar a quantidade de acesso ao disco, observando esse gráfico é possível notar o algoritmo *iDistance* apresentou o melhor desempenho, com os algoritmos $OmniB-Forest$ e *GroupSim* com desempenhos próximos a ele e muito similares, com a *OmniB-Forest* obtendo uma pequena vantagem. O algoritmo Acesso Sequencial foi o que apresentou o pior desempenho.

A Figura 33 (b) apresenta o desempenho dos algoritmos com relação ao número dos cálculos de distância realizados. O algoritmo $OmniB-Forest$ foi o que apresentou o melhor desempenho, seguido pelo algoritmo *GroupSim*. O Algoritmo Acesso Sequencial apresenta o pior desempenho.

A Figura 33 (c) apresenta o gráfico do desempenho dos algoritmos com relação ao tempo total médio gasto para as consultas realizadas. O algoritmo *GroupSim* foi o que

apresentou melhor desempenho. O Algoritmo $OmniB-Forest$ foi quem apresentou pior desempenho para este experimento.

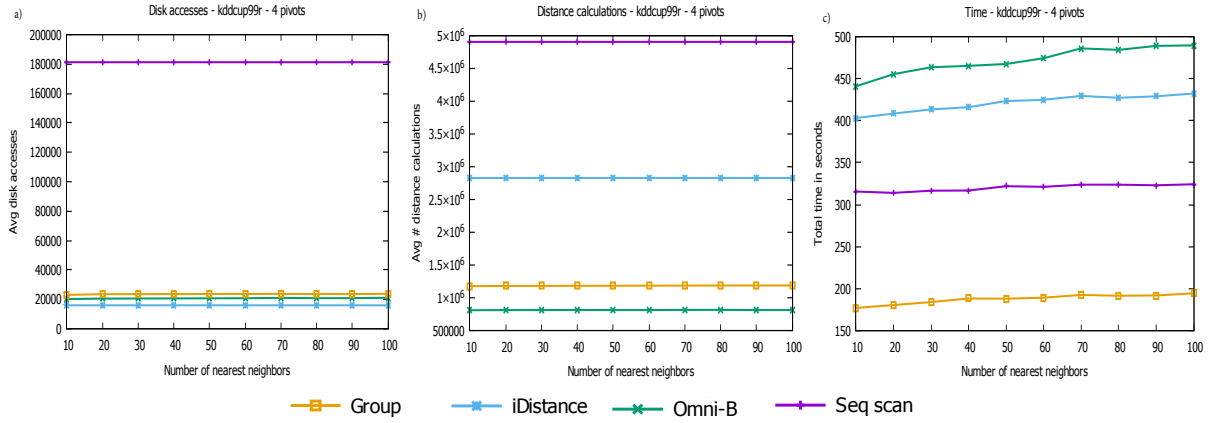

Figura 33 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância Euclidiana e a base de dados *Kdd Cup*.

4.3 Experimentos com a distância FastEMD

Os experimentos apresentados a seguir são referentes as bases de dados *Corel*, *Eingenfaces*, *Nasa*, *Clusters* e *Sierpinski*. Todos esses os experimentos foram realizados utilizando a função de distância FastEMD.

Na Figura 34 são exibidos os gráficos dos experimentos realizados com a base de dados *Corel*, utilizando 4 pivôs escolhidos por meio do algoritmo Aleatório.

A Figura 34 (a) apresenta o gráfico referente a quantidade de acessos ao disco realizados por cada consulta para cada um dos 4 algoritmos analisados. O algoritmo que obteve melhor desempenho foi o *iDistance*, com o algoritmo de Acesso Sequencial apresentando o pior desempenho, com um número de acessos quase 4 vezes maior.

Na Figura 34 (b) observando o gráfico é possível notar que o algoritmo *iDistance* que apresentou o melhor desempenho com relação à quantidade de acesso ao disco apresenta o pior desempenho para os números dos cálculos de distância realizados. Os outros três algoritmos apresentam o melhor desempenho, realizando a mesma quantidade dos cálculos de distância.

A Figura 34 (c) é referente ao gráfico que exibe o desempenho dos algoritmos com relação ao tempo gasto para cada consulta. O algoritmo *iDistance* apresenta um tempo relativamente maior do que os outros algoritmos, isso acontece porque o algoritmo realiza um maior número de cálculos de distância quando comparado aos outros três algoritmos. Nesse experimento o algoritmo de Acesso Sequencial apresenta a melhor média de

tempo, porém os algoritmos *OmniB−Forest* e *GroupSim* também apresentam um bom desempenho, as curvas dos três algoritmos chegam a se encontrarem em alguns pontos do gráfico.

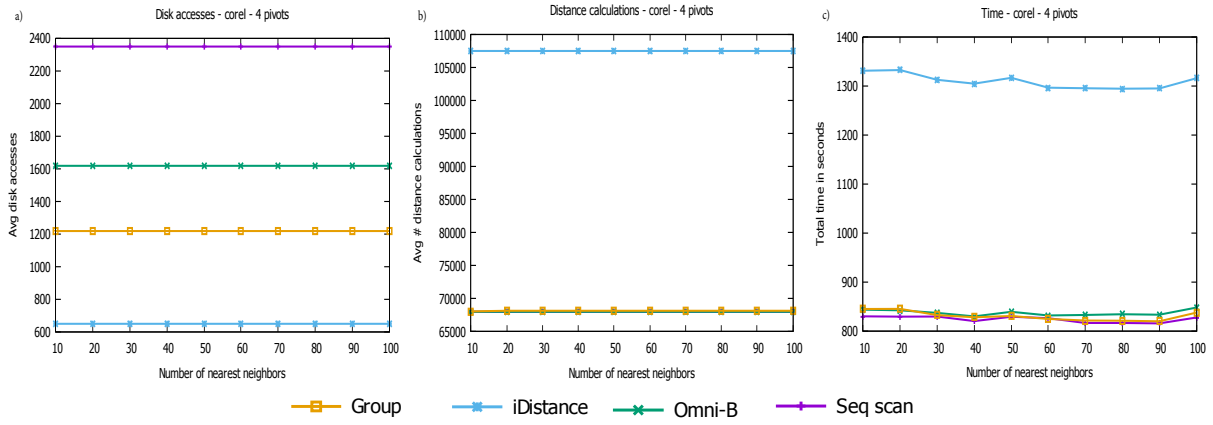

Figura 34 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , *OmniB−Forest*, *GroupSim* e *iDistance*, utilizando a função de distância FastEMD e a base de dados *Corel*.

A Figura 35 exibe os gráficos dos experimentos realizados com a base de dados *Eigenfaces*. Nesses experimentos foram utilizados 4 pivôs escolhidos por meio do algoritmo Aleatório.

Na Figura 35 (a) é possível notar por meio do gráfico que o algoritmo *iDistance* obteve o melhor desempenho com relação a quantidade de acessos ao disco realizados, enquanto o algoritmo *OmniB−Forest* apresentou o pior desempenho.

A Figura 35 (b) apresenta o gráfico com o desempenho dos algoritmos com relação ao número dos cálculos de distância realizados. O algoritmo *iDistance* que foi o que conseguiu o melhor desempenho com relação a quantidade de acessos ao disco realizados, aqui apresenta o pior desempenho, os outros três algoritmos apresentam juntos o melhor desempenho.

Na Figura 35 (c) é avaliado o tempo total médio gasto por cada algoritmo. O Acesso Sequencial apresentou o melhor desempenho, o algoritmo *GroupSim* também apresenta um bom desempenho, analisando sua curva é possível perceber que ela chega em determinado momento se igualar à do algoritmo de Acesso Sequencial.

A Figura 36 apresenta os gráficos dos experimentos que foram realizados utilizando 4 pivôs escolhidos por meio da função *Minimum of Maximum Distance*. A base de dados utilizado nesse experimento foi a *Nasa*.

Na Figura 36 (a) o gráfico apresenta o desempenho dos algoritmos com relação ao número de acessos ao disco que foram efetuados. O algoritmo *iDistance* é o que apresenta melhor desempenho, o algoritmo *GroupSim* possui um desempenho intermediário, com o algoritmo Acesso Sequencial apresentando o pior resultado.

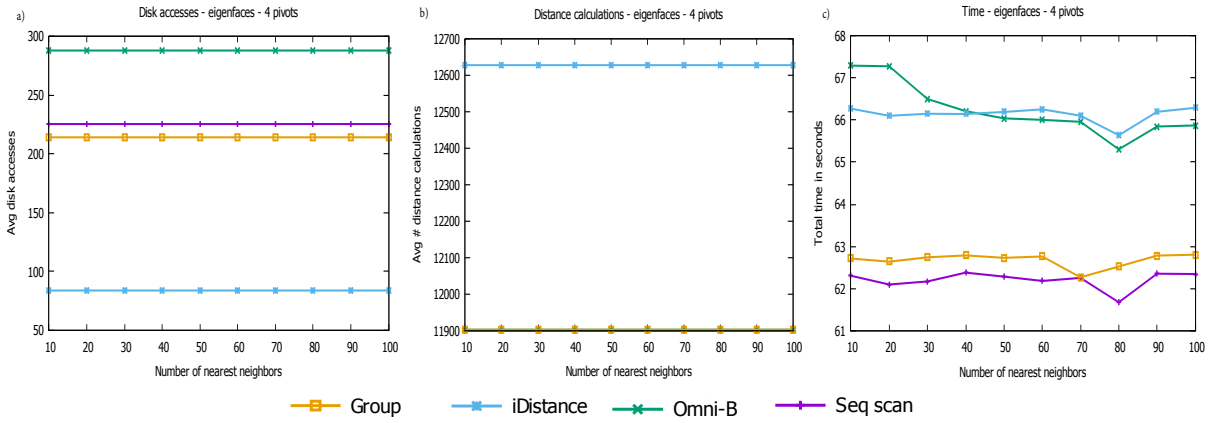

Figura 35 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância FastEMD e a base de dados *Eigenfaces*.

A Figura 36 (b) é referente ao gráfico para análise do número dos cálculos distância que foram realizados por cada um dos algoritmos. Nesse experimento os algoritmos *GroupSim* e $OmniB-Forest$ apresentam um desempenho muito similar, com o algoritmo de Acesso Sequencial obtendo o pior desempenho.

Na Figura 36 (c) é avaliado o tempo total médio gasto por cada algoritmo. O algoritmo *GroupSim* apresentou o melhor desempenho, e o algoritmo de Acesso Sequencial com o pior desempenho.

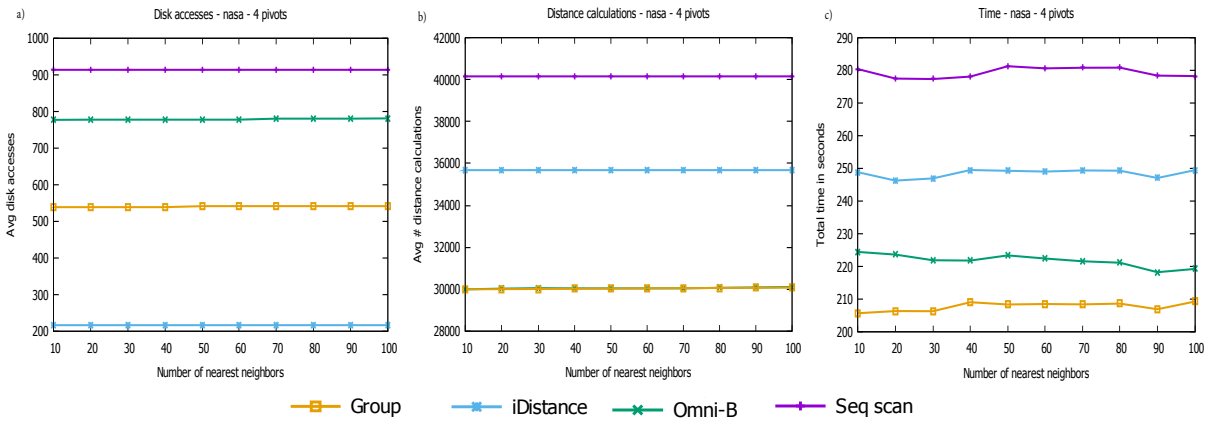

Figura 36 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância FastEMD e a base de dados *Nasa*.

A Figura 37 exibe os gráficos referentes aos experimentos realizados utilizando a base de dados *Clusters*. Para esses experimentos foram utilizados 4 pivôs escolhidos por meio da função *Hull of Foci*.

Na Figura 37 (a) o gráfico apresenta os resultados dos experimentos com relação ao número de acesso ao disco realizado para cada consulta. É possível notar que o melhor desempenho foi utilizando o algoritmo *GroupSim*, com o algoritmo de Acesso Sequencial apresentando o pior desempenho.

A Figura 37 (b) apresenta o gráfico com o desempenho dos algoritmos para o número dos cálculos de distância realizados, por meio desse gráfico é possível notar que os algoritmos *GroupSim* e $OmniB-Forest$ apresentaram juntos o melhor desempenho, com o algoritmo de Acesso Sequencial obtendo o pior desempenho com o algoritmo *iDistance* tendo um desempenho muito similar a ele.

Na Figura 37 (c) é apresentado o gráfico com tempo total médio gasto por cada algoritmo nas consultas realizadas. O algoritmo *GroupSim* apresentou o melhor desempenho, o *iDistance* obteve pior desempenho com o Acesso Sequencial obtendo um desempenho muito próximo à ele.

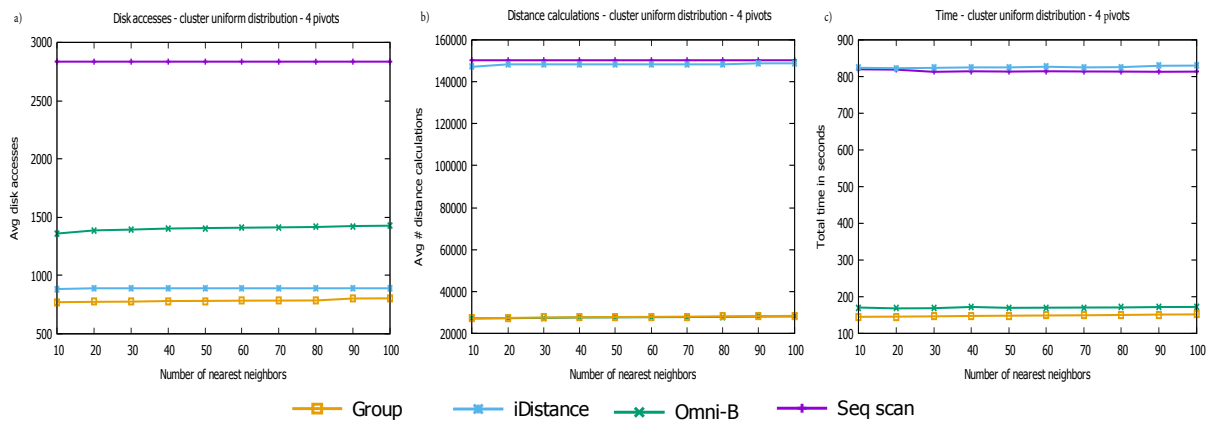

Figura 37 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância FastEMD e a base de dados *Cluster*.

A Figura 38 apresenta os gráficos dos experimentos que foram realizados utilizando a base de dados *Sierpinski*. Para esses experimentos foram utilizados 4 pivôs escolhidos por meio da função *Maximum of Sum of Distance*.

Na Figura 38 (a) é exibido o gráfico com as informações referentes a quantidade de acesso ao disco que é realizada por cada algoritmo para as consultas efetuadas. O algoritmo que apresentou melhor desempenho para essa etapa do experimento foi o *iDistance*. O algoritmo $OmniB-Forest$ foi o que apresentou o pior desempenho dentre os 4 algoritmos analisados.

A Figura 38 (b) apresenta o gráfico referente ao número dos cálculos de distâncias que foram efetuados paras as consultas de acordo com cada algoritmo analisado. O algoritmo *GroupSim* foi o que apresentou o menor número de cálculos de distâncias efetuados, com

o algoritmo *iDistance* realizando 100.000 cálculos de distâcia a mais. O Acesso Sequencial foi o algoritmo que apresentou o pior desempenho.

Na Figura 38 (c) o gráfico apresenta as informações referentes ao tempo total médio de cada algoritmo para as consultas realizadas. O algoritmo *GroupSim* foi o algoritmo que apresentou a melhor média de tempo, o que contribuiu para o bom desempenho desse algoritmo foi ele ter realizado um baixo número de cálculo de distância.

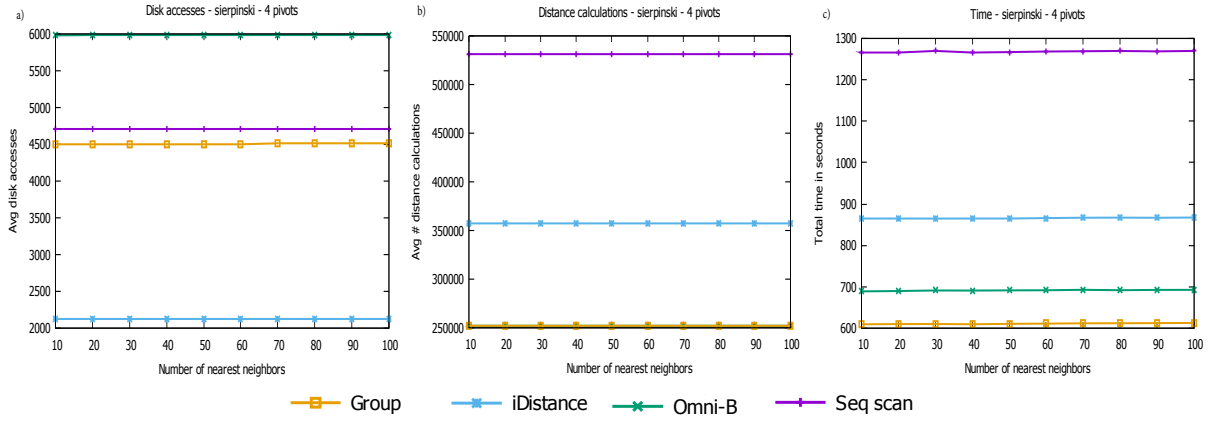

Figura 38 – No gráfico (a) é exibida a comparação do número de acessos ao disco, em (b) a comparação da quantidade dos calculos de distância que foram realizados e em (c) a comparação do tempo total gasto nas consultas para os algoritmos *Sequencial* , $OmniB-Forest$, *GroupSim* e *iDistance*, utilizando a função de distância FastEMD e a base de dados *Sierpinski*.

4.4 Considerações Finais

Analisando os experimentos é possível notar que para funções com baixo custo computacional a quantidade de acesso ao disco pode influenciar mais no tempo gasto pelo algoritmo do que o número dos cálculos de distância realizados. Outro fator que também influencia é maneira que o algoritmo acessa os dados em disco, quanto mais acessos maior o tempo total, e se esses acessos forem muito aleatórios o tempo pode aumentar bastante.

Neste capítulo foram apresentados diversos experimentos com conjuntos de dados sintéticos e reais, de dimensões e tamanhos diferentes, utilizando duas funções de distância com tempo de processamento diferentes, visando a comparação do desempenho das consultas por dissimilaridade da *GroupSim* em relação a $OmniB-Forest$, *iDistance* e Acesso Sequencial.

Por meio dos experimentos foi possível notar que a técnica *GroupSim* apresentou um bom desempenho, principalmente quando comparada com os outros métodos proposto $OmniB-Forest$ e *iDistance*. A técnica *GroupSim* perde em alguns momentos para a técnica de Acesso Sequencial, porém mesmo obtendo pior desempenho ainda é melhor do que as outras duas técnicas.

Os resultados onde o Acesso Sequencial obtém melhor desempenho é totalmente compreensível, pois como já foi falado os resultados referentes ao acesso á disco apresentados para os métodos que utilizam a estrutura B^+-Tree, não estão contabilizando o acesso aleatório ao disco que é realizado na recuperação do objeto, estão contabilizando apenas o acesso a referência que está salva na estrutura. Esses acessos influenciam no tempo total gasto em um consulta.

A Tabela 3 apresenta o percentual relativo ao tempo médio gasto em cada consulta que foi analisada nos experimentos, avaliando o método proposto com relação aos outros métodos que foram apresentados (*iDistance*, *GrouB-Forest*, e Escaneamento Sequencial). Observando essa tabela é possível notar que o método proposto foi superior ao *iDistance* em todos os casos de teste. Observando a comparação com o *OmniB-Forest* o método proposto é superior em praticamente todos os casos de teste com exceção a base de dados *Corel*, onde o método *OmniB-Forest* é 20% superior para a medida de dissimilaridade Euclidiana e empata para medida de dissimilaridade *FastEMD*. Comparado com Escaneamento Sequencial o método proposto não demonstrou tanta superioridade quanto aos outros métodos, mas ainda sim foi superior na maioria dos testes. Na comparação com Escaneamento Sequencial ele perde nos seguintes casos: *Colorstructure* com a medida de dissimilaridade Euclidiana onde o Escaneamento Sequencial é $28,57\%$ superior, *Nasa* com a medida de dissimilaridade Euclidiana onde o Escaneamento Sequencial é $33,33\%$ superior, *Cluster* onde o Escaneamento Sequencial é $37,50\%$ superior, *Corel* com a medida de dissimilaridade FastEMD onde o Escaneamento Sequencial é $0,35\%$ superior e *Eigenfaces* onde o Escaneamento Sequencial é $1,59\%$ superior.

Tabela 3 – Porcentagem média comparando a diferença do tempo médio entre o método *GroupSim* e os outros métodos, para as bases todas as bases de dados usadas e medidas de dissimilaridade adotadas nos experimentos.

Base de Dados	**Medida de Dissimilaridade**	***GroupSim* x iDistance**	***GroupSim* x OmniB**	***GroupSim* x Seq Scan**
Colorstructure	Euclidiana	75,00%	285,71%	-28,57%
Colors	Euclidiana	300,00%	300,00%	137,50%
Corel	Euclidiana	120,00%	-20,00%	180,00%
Eigenfaces	Euclidiana	650,00%	200,00%	100,00%
Nasa	Euclidiana	100,00%	200,00%	-33,33%
Cluster	Euclidiana	112,50%	437,50%	-37,50%
Sierpinski	Euclidiana	3400,00%	100,00%	1000,00%
Covertype	Euclidiana	490,00%	1000,00%	460,00%
Kdd Cup	Euclidiana	127,80%	155,56%	72,22%
Corel	FastEMD	58,58%	0,00%	-0,35%
Eigenfaces	FastEMD	4,76%	3,17%	-1,59%
Nasa	FastEMD	21,85%	9,76%	36,58%
Cluster	FastEMD	350,00%	5,56%	349,44%
Sierpinski	FastEMD	46,67%	16,67%	113,33%

É possível notar que existe uma diferença com relação aos resultados para as funções Euclidiana e *FastEMD*. Observando os gráficos referentes a quantidade dos cálculos de distância, nota-se que os experimentos realizados utilizando a função *FastEMD* apresentam um número muito superior de cálculos de distância do que os que foram realizados utilizando a função *Euclidiana*, isso faz com que o tempo para uma consulta utilizando a *FastEMD* (quadrático) que já seria superior ao de uma consulta Euclidiana (Linear) se torne ainda maior.

Capítulo 5

Conclusão

Dados complexos como multimídia, cadeias de DNA, entre outros, podem ser armazenados em SGDBs. As estruturas de indexação que compõem esses SGDBs são responsáveis por fazer com que as informações armazenadas sejam recuperadas de forma eficiente. Dados complexos como os que foram citados são indexados por meio de Métodos de Acesso Métricos (MAMs) utilizando apenas uma medida de dissimilaridade e um código de identificação do objeto como informação, uma vez que os dados são armazenados em arquivos separados.

Na literatura existem diferentes tipos de Métodos de Acesso cada um com uma abordagem diferente, para diferentes tipos de dados. A avaliação que é feita sobre esses métodos é referente ao tempo gasto para recuperar as informações indexadas, em se tratando de dados complexos uma abordagem que atinge excelentes resultados para determinado conjunto de dados pode ter um desempenho ruim para outro conjunto, e existem alguns fatores que influenciam esse desempenho como a dimensionalidade e a distribuição do conjunto.

A estrutura *GroupSim* visa obter melhor desempenho do que alguns métodos propostos na literatura, uma vez que alguns métodos dependem de uma estrutura auxiliar para que a recuperação dos objetos ocorra e a *GroupSim* não dependa, diminuindo assim uma etapa no processo de recuperação de objeto. Alguns resultados obtidos por meio de experimentos comprovam que a estrutura proposta é mais eficiente do que os métodos avaliados $OmniB-Forest$, *iDistance* e Escaneamento Sequencial na maioria dos casos de teste efetuados.

De uma maneira geral o método proposto obtém um desempenho médio muito bom, os experimentos mostraram que o método proposto é em média $418,3741\%$ mais eficiente do que o *iDistance*, $192,4231\%$ mais eficiente do que o *OmniB-Forest* e $167,6956\%$ mais eficiente do que o Escaneamento Sequencial.

O algoritmo proposto apresenta melhores resultados em conjuntos que possuem distribuição uniforme, assim como os algorimtos $OmniB-Forest$ e *iDistance*. Para conjuntos de dados com outras distribuições nem sempre será viável a aplicação do algoritmo.

5.1 Principais Contribuições

Este tabalho apresenta as seguintes contribuições:

- ❏ foi criado o método *GroupSim* para indexação por similaridade por meio de mapeamentos unidimensionais de dados e com o uso de $B^{+}-Tree$;
- ❏ foram criados os algoritmos de consulta por abrangência e de consulta aos k-vizinhos mais próximos para a estrutura;
- ❏ foram apresentadas as técnicas para escolha de pivôs e foram realizados extensos experimentos com diversos conjuntos de dados para se verificar a adequação das técnicas com os métodos de acesso estudados;
- ❏ foram realizados experimentos para validação da estrutura e algoritmos propostos em relação aos principais trabalhos correlatos.

5.2 Trabalhos Futuros

Um universo de pesquisa a ser explorado a cerca dos métodos de acessa é recuperação de objetos utilizando medidas de dissimilaridades não métricas. Alguns autores como por exemplo (SKOPAL, 2007) defendem que para recuperação de imagens as medidas de dissimilaridades não métricas possuem melhores resultados, uma vez que estabelecem uma relação de similaridade mais próxima à percepção humana.

5.2.1 Recuperação dos Objetos Através de uma Função Não Métrica

Quando se aplica uma função não métrica para indexação e recuperação de objetos, o resultado da consulta pode não ser exato, levando a um resultado aproximado, ou seja, se comparado a uma consulta sequencial, o conjunto de resposta relativo a função não métrica utilizando a estrutura de indexação, pode não apresentar os mesmos resultados que uma consulta sequencial. Esse problema ocorre porque as estruturas de indexação geralmente utilizam da desigualdade triangular para podar os elementos durante uma pesquisa, e pode acontecer da medida de dissimilaridade utilizada violar a propriedade da desigualdade triangular. Para contornar esse problema propõe-se o treinamento de um classificador para estimar um *treshold* α para aproximação da consulta. A Figura 39 exibe o problema gerado pela quebra da desigualdade a triangular, e onde entraria o α.

Para encontrar o *Treshold* α foi proposta uma técnica onde é necessário realizar um treinamento com uma amostra de objetos selecionados da base de dados. Para definir a amostra são escolhidos n objetos do conjunto de dados aleatoriamente, onde o tamanho

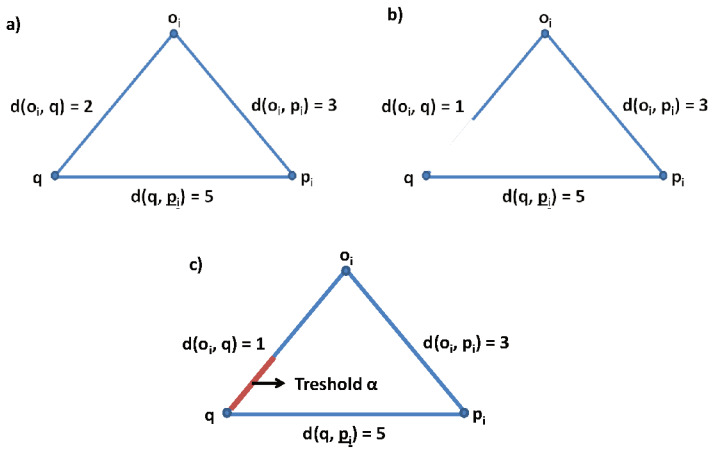

Figura 39 – Em a) o comportamento é normal, em b) existe a quebra da desigualdade e em c) é apresentado o *Treshold*.

de n é definido pelo usuário. É necessário verificar através da medida de dissimilaridade quais dos n objetos selecionados violam a propriedade da desigualdade triangular.

Ao verificar se um objeto viola a propriedade da desigualdade triangular, é necessário armazenar algumas informações sobre ele para que possam ser utilizadas posteriormente. As informações armazenadas são: *OID*s do primeiro objeto, do pivô, e do segundo objeto; distâncias entre: objetos um e dois, objeto um e pivô, e objeto dois e pivô. Essas informações são armazenadas em um arquivo sequencial ordenadas pela distância do objeto um para o pivô.

Além de armazenar as informações em um arquivo sequencial, é feita a indexação através da técnica $OmniB{-}Forest$ (Subseção 2.5.4.1) para um acesso eficiente a essas informações, onde são indexadas a distância do objeto um para o pivô (que será a chave) e também o *OID* do objeto um.

Para acessar as informações é feita uma consulta *knn* com $k = 1$, usando o objeto *query* da consulta principal. O resultado dessa consulta é utilizado para calcular o *treshold* α. O calculo de α é representado pela equação 16,

$$\alpha = (d(o_1, o_2) - (d(o_1, p_i) + d(o2, p_i)))\lambda \tag{16}$$

onde $\lambda = (100 - k)/100$ para k ≤ 90 e $\lambda = 0.10$ para k ≥ 90, o_1 é objeto mais próximo da consulta q, o_2 é objeto que juntamente com o pivô p_i e o_1 deveriam formar um triângulo. O valor p varia de i até n, porque serão calculados n *tresholds* um para cada pivô. O valor de λ varia de acordo com o tamanho do k escolhido pelo usuário para a consulta, sendo que quanto maior o k, maior o raio da consulta se torna, e para não exceder o tempo de uma consulta sequencial o *treshold* deve diminuir. Existe também a possibilidade do usuário definir o tamanho de λ, caso ele deseje maior precisão ele pode aumentar o valor de λ, porém com isso aumentará o tempo computacional.

Os algoritmos para as consultas *range query* e *knn* são praticamente os mesmos. As mudanças no algoritmo *range query* 10, estão na linha 5 onde é adicionado o valor α_1 ao raio, e na linha 7 onde também é adicionado o valor de α_i ao raio. Ja no algoritmo *knn* (11 e 12) é necessário alterar as linhas 6 e 24 do algoritmo 12, e também as linhas 18 e 21 do algoritmo 11. As alterações consistem em adicionar o *treshold* α à variável raio, ou seja, nas comparações tem se $raio + \alpha_i$.

Este

5.3 Contribuições em Produção Bibliográfica

Um artigo científico foi preparado e submetido para apreciação no ACM Symposium on Applied Computing (ACM SAC 2017).

Referências

BAYER, R.; MCCREIGHT, E. Organization and maintenance of large ordered indices. In: **ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control**. New York, NY: ACM, 1970. (SIGFIDET), p. 107–141. Disponível em: <https://doi.org/10.1007/BF00288683>.

BIEDERMAN, I. Recognition-by-components: a theory of human image understanding. **Psychological review**, American Psychological Association, v. 94, n. 2, p. 115, 1987. Disponível em: <http://dx.doi.org/10.1037/0033-295X.94.2.115>.

BLAKE, C.; MERZ, C. J. {UCI} repository of machine learning databases. 1998.

BOLETTIERI, P. et al. CoPhIR: a test collection for content-based image retrieval. **CoRR**, abs/0905.4627v2, 2009. Disponível em: <http://cophir.isti.cnr.it>.

BOZKAYA, T.; OZSOYOGLU, M. Distance-based indexing for high-dimensional metric spaces. In: ACM. **ACM SIGMOD Record**. 1997. v. 26, n. 2, p. 357–368. Disponível em: <http://doi.org/10.1145/253260.253345>.

_____. Indexing large metric spaces for similarity search queries. **ACM Transactions on Database Systems (TODS)**, ACM, v. 24, n. 3, p. 361–404, 1999. Disponível em: <http://doi.org/10.1145/328939.328959>.

BUGATTI, P. H.; TRAINA, A. J.; JR, C. T. Assessing the best integration between distance-function and image-feature to answer similarity queries. In: ACM. **Proceedings of the 2008 ACM symposium on Applied computing**. 2008. p. 1225–1230. Disponível em: <http://doi.org/10.1145/1363686.1363969>.

BURKHARD, W. A.; KELLER, R. M. Some approaches to best-match file searching. **Communications of the ACM**, ACM, v. 16, n. 4, p. 230–236, 1973. Disponível em: <http://doi.org/10.1145/362003.362025>.

BUSTOS, B.; NAVARRO, G.; CHÁVEZ, E. Pivot selection techniques for proximity searching in metric spaces. **Pattern Recognition Letters**, Elsevier, v. 24, n. 14, p. 2357–2366, 2003. Disponível em: <http://doi.org/10.1016/S0167-8655(03)00065-5>.

CHÁVEZ, E. et al. Searching in metric spaces. **ACM computing surveys (CSUR)**, v. 33, n. 3, p. 273–321, 2001. Disponível em: <http://doi.org/10.1145/502807.502808>.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: **International Conference on Very Large Data Bases**. San Francisco, CA: Morgan Kaufmann, 1997. (VLDB), p. 426–435. Disponível em: <https://doi.org/10.1007/978-3-642-03869-3_48>.

COMER, D. Ubiquitous b-tree. **ACM Computing Surveys (CSUR)**, v. 11, n. 2, p. 121–137, 1979. Disponível em: <http://doi.org/10.1145/356770.356776>.

CUP, K. **Intrusion detection data set**. 1999.

DEZA, M. M.; DEZA, E. Encyclopedia of distances. In: **Encyclopedia of Distances**. [S.l.]: Springer, 2009. p. 1–583.

EAKINS, J.; GRAHAM, M. **Content-Based Image Retrieval: A Report to the JISC Technology Application Program University of Northuinbria at Newcastle). Retrieved April 22, 2001**. 1999. Disponível em: <https://doi.org/10.26634/jse.7.2.2038>.

GUTTMAN, A. **R-trees: a dynamic index structure for spatial searching**. [s.n.], 1984. v. 14. Disponível em: <http://doi.org/10.1145/971697.602266>.

JAGADISH, H. V. et al. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. **ACM Transactions on Database Systems (TODS)**, v. 30, n. 2, p. 364–397, 2005. Disponível em: <htpp://doi.org/10.1145/1071610.1071612>.

JR, C. T. et al. The omni-family of all-purpose access methods: av simple and effective way to make similarity search more efficient. **The VLDB Journal**, Springer, v. 16, n. 4, p. 483–505, 2007.

_____. **Slim-trees: High performance metric trees minimizing overlap between nodes**. [S.l.]: Springer, 2000.

K.FIGUEROA; G.NAVARRO; E.CHÁVEZ. **Metric Spaces Library**. 2007. Disponível em: <http://www.sisap.org/sisap2/SISAP/Metric_Space_Library.html>.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. **Proceedings of the American Mathematical society**, JSTOR, v. 7, n. 1, p. 48–50, 1956. Disponível em: <http://dx.doi.org/10.1090/S0002-9939-1956-0078686-7>.

LICHMAN, M. **UCI Machine Learning Repository**. 2013. Disponível em: <http://archive.ics.uci.edu/ml>.

LONG, F.; ZHANG, H.; FENG, D. D. Fundamentals of content-based image retrieval. In: **Multimedia Information Retrieval and Management**. [s.n.], 2003. p. 1–26. Disponível em: <https://doi.org/10.1007/978-3-662-05300-3_1>.

MANOLOPOULOS, Y. et al. **R-trees: Theory and Applications**. [S.l.]: Springer Science & Business Media, 2010.

MICÓ, M. L.; ONCINA, J.; VIDAL, E. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. **Pattern Recognition Letters**, Elsevier, v. 15, n. 1, p. 9–17, 1994. Disponível em: <http://doi.org/10.1016/0167-8655(94)90095-7>.

MONGE, G. **Mémoire sur la théorie des déblais et des remblais**. [s.n.], 1781. Disponível em: <https://doi.org/10.1112/plms/s1-14.1.139>.

MÜLLER, H. et al. A review of content-based image retrieval systems in medical applicationsclinical benefits and future directions. **International journal of medical informatics**, Elsevier, v. 73, n. 1, p. 1–23, 2004. Disponível em: <http://doi.org/10.1016/j.ijmedinf.2003.11.024>.

PEDREIRA, O.; BRISABOA, N. R. Spatial selection of sparse pivots for similarity search in metric spaces. In: **SOFSEM 2007: Theory and Practice of Computer Science**. Springer, 2007. p. 434–445. Disponível em: <https://doi.org/10.1007/978-3-540-69507-3_37>.

PELE, O.; WERMAN, M. A linear time histogram metric for improved sift matching. In: **Computer Vision–ECCV 2008**. [s.n.], 2008. p. 495–508. Disponível em: <https://doi.org/10.1007/978-3-540-88690-7_37>.

_____. Fast and robust earth mover's distances. In: IEEE. **Computer vision, 2009 IEEE 12th international conference on**. 2009. p. 460–467. Disponível em: <http://doi.org/10.1109/ICCV.2009.5459199>.

PELEG, S.; WERMAN, M.; ROM, H. A unified approach to the change of resolution: Space and gray-level. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 11, n. 7, p. 739–742, 1989. Disponível em: <http://doi.org/10.1109/34.192468>.

RUBNER, Y.; GUIBAS, L.; TOMASI, C. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In: **in Proceedings of the ARPA Image Understanding Workshop**. [s.n.], 1997. p. 661–668. Disponível em: <https://doi.org/10.1016/j.comgeo.2006.10.001>.

RUBNER, Y. et al. Empirical evaluation of dissimilarity measures for color and texture. **Computer vision and image understanding**, v. 84, n. 1, p. 25–43, 2001. Disponível em: <https://doi.org/10.1006/cviu.2001.0934>.

RUBNER, Y.; TOMASI, C.; GUIBAS, L. J. The earth mover's distance as a metric for image retrieval. **International journal of computer vision**, v. 40, n. 2, p. 99–121, 2000. Disponível em: <https://doi.org/10.1023/A:1026543900054>.

SKOPAL, T. On fast non-metric similarity search by metric access methods. In: **Advances in Database Technology-EDBT 2006**. [S.l.]: Springer, 2006. p. 718–736.

_____. Unified framework for fast exact and approximate search in dissimilarity spaces. **ACM Transactions on Database Systems (TODS)**, ACM, v. 32, n. 4, p. 29, 2007.

SOCORRO, R.; MICÓ, L.; ONCINA, J. A fast pivot-based indexing algorithm for metric spaces. **Pattern Recognition Letters**, Elsevier, v. 32, n. 11, p. 1511–1516, 2011.

STOLFI, J. Personal communication. 1994.

SWAIN, M. J.; BALLARD, D. H. Color indexing. **International journal of computer vision**, Springer, v. 7, n. 1, p. 11–32, 1991.

TANG, Y. et al. Earth moverś distance based similarity search at scale. **Proceedings of the VLDB Endowment**, v. 7, n. 4, p. 313–324, 2013.

TRAINA, A. et al. Similarity search without tears: the omni-family of all-purpose access methods. In: IEEE. **Data Engineering, 2001. Proceedings. 17th International Conference on**. [S.l.], 2001. p. 623–630.

TURK, M.; PENTLAND, A. Eigenfaces for recognition. **Journal of cognitive neuroscience**, MIT Press, v. 3, n. 1, p. 71–86, 1991.

TYPKE, R.; WALCZAK-TYPKE, A. A tunneling-vantage indexing method for non-metrics. In: **ISMIR**. [S.l.: s.n.], 2008. p. 683–688.

VIEIRA, M. R. et al. Dbm-tree: A dynamic metric access method sensitive to local density data. In: CITESEER. **In SBBD**. [S.l.], 2004.

YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. **Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms**. [S.l.], 1993. p. 311–321.

ZEZULA, P. et al. **Similarity search: the metric space approach**. [S.l.]: Springer, 2006. v. 32.

Apêndices

APÊNDICE **A**

Resultados de Experimentos Complementares

Neste capítulo serão apresentados algumas informações técnicas e teóricas que contribuem para um melhor entendimento sobre o trabalho que foi realizado.

A.1 Experimentos

Esta etapa dos experimentos consiste em determinar o número de pivôs que serão usados no mapeamento dos objetos, e qual função para escolha dos pivôs adotar. Para os experimentos foram utilizadas todas as bases de dados citadas na seção 4.1, para o conjunto *CoPhIR* (BOLETTIERI et al., 2009) foi escolhido o arquivo contendo o primeiro arquivo que contém um milhão de objetos.

Os algoritmos para escolha dos pivôs são os que foram apresentados na seção 2.5.4.3: Aleatório, *Hull Foci*, *Maximum of Minimum Distances*, *Maximum of Sum of Distances*. Todos foram avaliados por meio de uma consulta *knn* com k variando entre 10 e 100, utilizando o algoritmo *OmniB-Forest* para indexação e recuperação dos objetos.

Os gráficos apresentados nas figuras 40 (a), (b) e (c) são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Colorstructure*, observando os gráficos é possível notar que para uma consulta o ideal é utilizar o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) escolhendo 4 pivôs.

O gráfico (a) apresentado na Figura 40 utiliza o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) o número de acesso ao disco é menor quando comparado aos outros algoritmos, nesse caso é possível notar que utilizar um algoritmo para escolher os pivôs de forma aleatória seria a pior opção.

O gráfico (b) da Figura 40 é referente ao número dos cálculos de distância que são realizados, e o algoritmo que apresenta melhor desempenho também é o *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011), aqui o algoritmo de escolha aleatória não é o que apresenta pior desempenho. Os algorit-

mos *Hull Foci* (TRAINA et al., 2001; JR et al., 2007) e o *Maximum of Sum of Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) apresentam o pior desempenho, realizando praticamente o mesmo número de cálculos de distância.

No gráfico (c) apresentado na Figura 40 o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) é o que apresenta o melhor tempo médio de consulta, os outros algoritmos apresentam praticamente o mesmo tempo com apenas pequenas variações de tempo entre eles.

Os gráficos (d), (e), (f), (g), (h) e (i) apresentados na Figura 40, são referentes apenas ao tempo médio gasto para cada consulta. Observando esses gráficos é possível notar que com o número de pivôs aumentando o tempo médio para a consulta também aumenta, e o algoritmo que apresenta o melhor tempo médio para consulta pode variar.

Os gráficos (a), (b) e (c) apresentados pela Figuras 41 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Colors*, observando os gráficos é possível notar que para uma consulta o ideal é utilizar um algoritmo para escolher de forma aleatória 4 pivôs.

O gráfico (a) da Figura 41 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados, é possível notar por meio desse gráfico que o algoritmo de melhor desempenho é o que seleciona os pivôs de forma aleatória, os outros algoritmos apresentam basicamente o mesmo resultado, com uma quantidade de acesso à disco bem superior quando comparados ao algoritmo aleatório.

O gráfico (b) apresentado na Figura 41 mostra que o algoritmo para escolha aleatória dos pivôs também apresenta bons resultados para quantidade dos cálculos de distância efetuados. Os outros algoritmos apresentam basicamente o mesmo desempenho, chegando a fazer praticamente o dobro dos cálculos de distância que o algoritmo aleatório realiza.

Por meio do gráfico (c) da Figura 41 é possível notar que o algoritmo para escolha aleatória dos pivôs é o que apresenta melhor tempo médio para cada consulta. Os outros três algoritmos avaliados apresentam o mesmo tempo médio.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 41 apresentam apenas a informação de tempo médio gasto em cada consulta. Esses gráficos apresentam um aumento de tempo médio gasto para cada algoritmo sempre que a quantidade de pivôs aumenta. Observando esses gráficos é possível notar que independente da quantidade de pivôs o melhor algoritmo é o que realiza a escolha aleatória, com os outros três algoritmos apresentando resultados equivalentes.

Os gráficos (a), (b) e (c) da Figura 42 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Corel*, observando os gráficos é possível notar que para uma consulta o ideal é utilizar um algoritmo para escolher de forma aleatória 4 pivôs.

O gráfico (a) apresentado na Figura 42 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados, é possível notar por meio desse gráfico

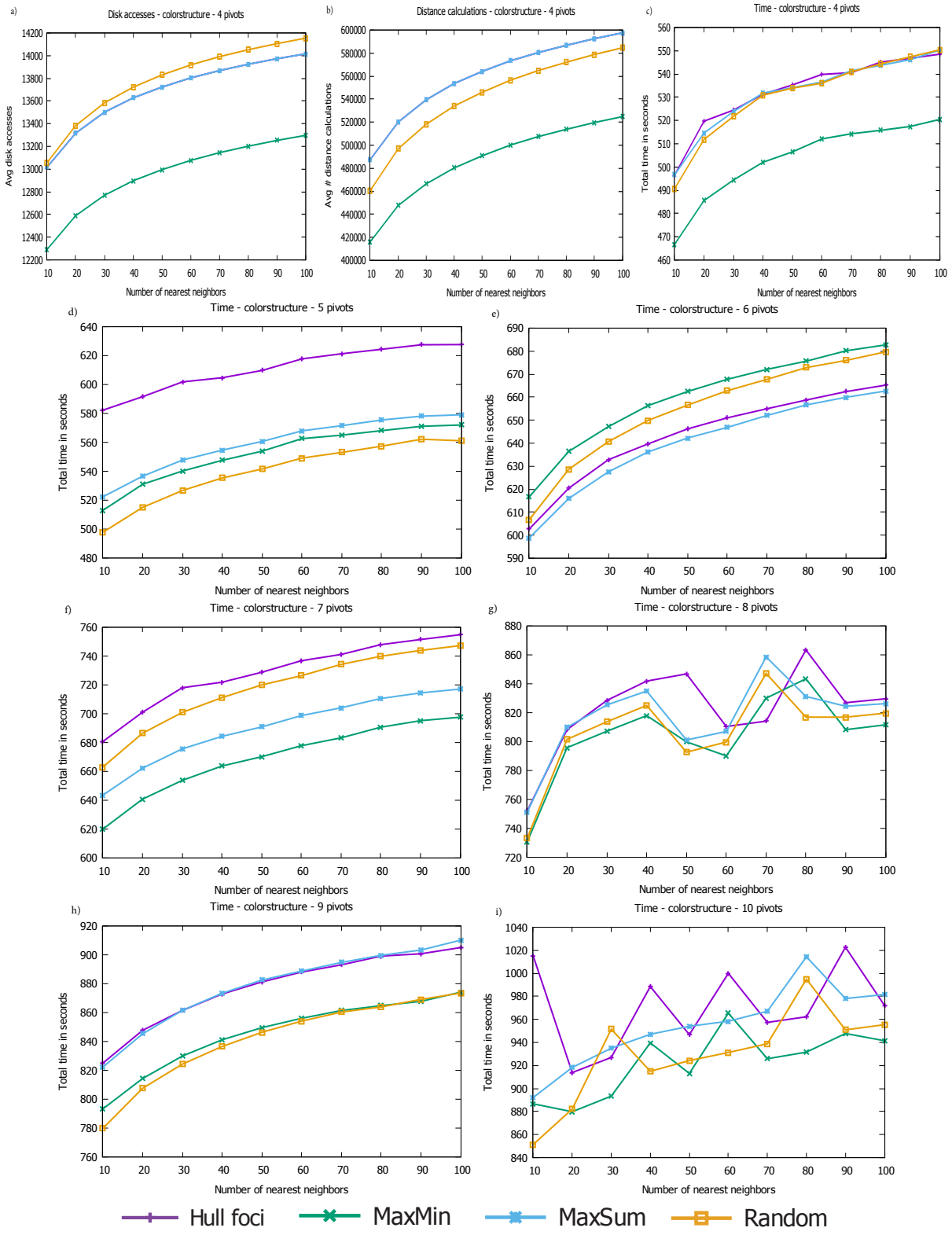

Figura 40 – Gráficos de desempenho dos testes realizados usando a base de dados *Colorstructure* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

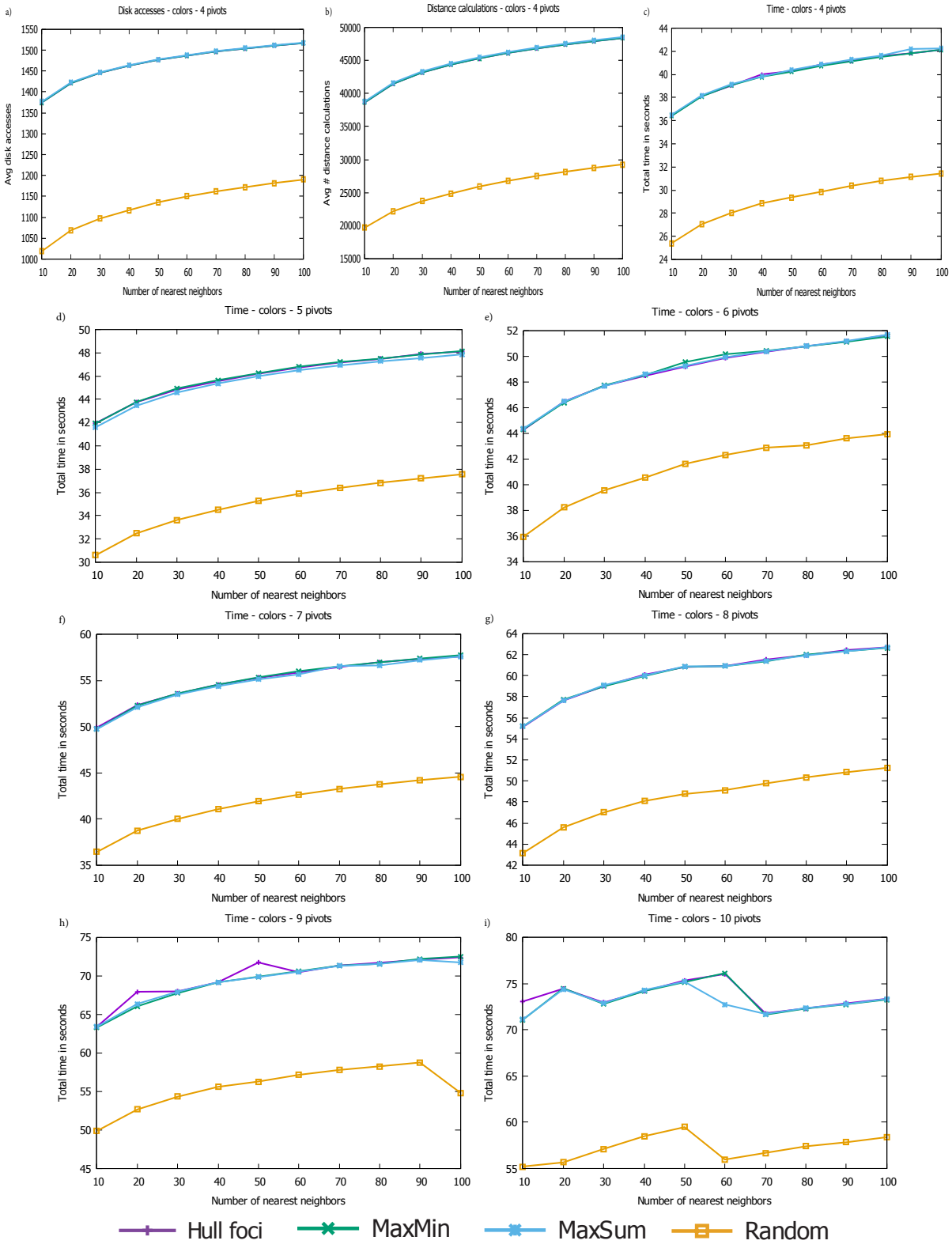

Figura 41 – Gráficos de desempenho dos testes realizados usando a base de dados *Colors* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

que o algoritmo de melhor desempenho é o que seleciona os pivôs de forma aleatória, os outros algoritmos apresentam basicamente o mesmo resultado, com uma quantidade de acesso à disco bem superior quando comparados ao algoritmo aleatório.

O gráfico (b) aprsentado na Figura 42 mostra que o algoritmo para escolha aleatória dos pivôs também apresenta bons resultados para quantidade dos cálculos de distância efetuados. Os outros algoritmos apresentam basicamente o mesmo desempenho, chegando a fazer praticamente o dobro dos cálculos de distância que o algoritmo aleatório realiza.

No gráfico (c) da Figura 42 o algoritmo para escolha aleatória dos pivôs é o que apresenta melhor tempo médio para cada consulta. Os outros três algoritmos avaliados apresentam o mesmo tempo médio.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 42 apresentam apenas a informação de tempo médio gasto em cada consulta. Esses gráficos apresentam um aumento de tempo médio gasto para cada algoritmo sempre que a quantidade de pivôs aumenta. Observando esses gráficos é possível notar que independente da quantidade de pivôs o melhor algoritmo é o que realiza a escolha aleatória, com os outros três algoritmos apresentando resultados equivalentes.

Os gráficos (a), (b) e (c) apresentados na Figura 43 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Eigenfaces*. Para essa base de dados os algoritmos: *Hull of Foci* (TRAINA et al., 2001; JR et al., 2007), *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) e *Maximum of Sum of Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) apresentam um desempenho muito similar, porém o algoritmo *Maximum of Minimum Distances* leva uma ligeira vantagem, apenas o algoritmo para escolha aleatória dos pivôs que apresenta um resultado ruim. Para essa configuração foram utilizados 4 pivôs.

O gráfico (a) da Figura 43 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que o algoritmo de melhor desempenho é o *Maximum of Minimum Distances*, com o *Hull of Foci* e o *Maximum of Sum of Distances* apresentado o mesmo desempenho, e o algoritmo Aleatório apresentando o pior desempenho.

O gráfico (b) apresentado na Figura 43 mostra que o algoritmo *Maximum of Minimum Distances* tem o pior resultado para a quantidade dos cálculos de distância efetuados. Os algoritmos que apresentam o melhor desempenho aqui são: *Hull of Foci* e o *Maximum of Sum of Distances* com quantidade muito similar dos cálculos de distância que foram realizados.

No gráfico (c) da Figura 43 o algoritmo *Maximum of Minimum Distances* é o que apresenta melhor tempo médio para cada consulta, porém os algoritmos *Hull of Foci* e *Maximum of Sum of Distances* apresentam um tempo muito próximo a ele, isso acontece porque no gráfico (a) o algoritmo *Maximum of Minimum Distances* apresentou melhor

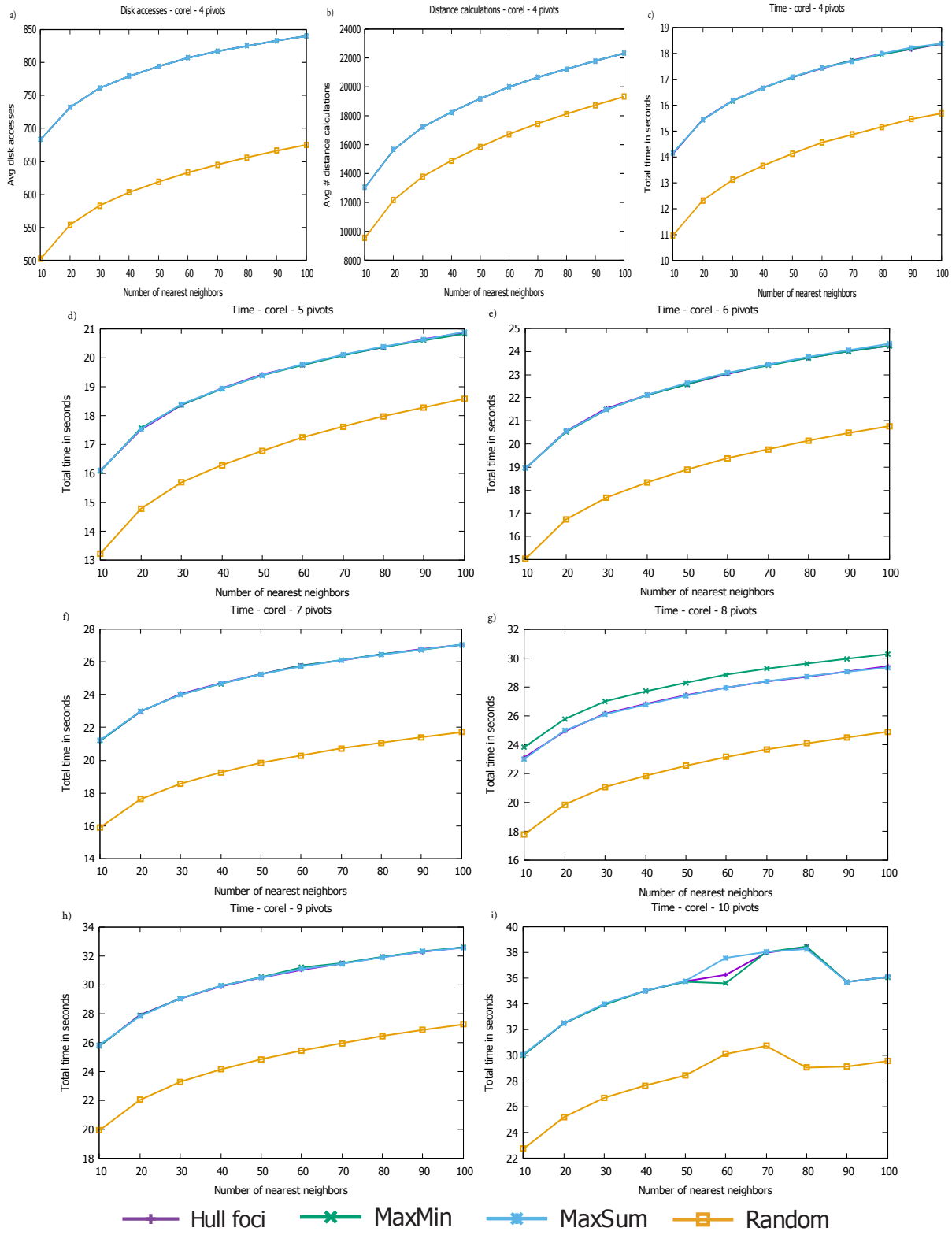

Figura 42 – Gráficos de desempenho dos testes realizados usando a base de dados *Corel* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

resultado e no (b) ele apresentou o pior resultado. Seu desempenho só foi um pouco melhor porque o tempo gasto para acesso à disco é ligeiramente superior ou gasto para os cálculos de distância.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 43 apresentam apenas a informação de tempo médio gasto em cada consulta. No gráfico (d) o melhor algoritmo ainda é o *Maximum of Minimum Distances*, porém nos outros gráficos é possível notar que os algoritmos *Hull of Foci* e *Maximum of Sum of Distances* levam vantagem.

Os gráficos (a), (b) e (c) apresentados na Figura 44 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Nasa*. Para essa base de dados os algoritmos: *Hull of Foci* (TRAINA et al., 2001; JR et al., 2007) e *Maximum of Sum of Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) apresentam um desempenho muito similar, ambos os algoritmos podem ser considerados como uma melhor escolha. Para essa configuração foram utilizados 4 pivôs.

O gráfico (a) da Figura 44 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que apenas o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) apresenta um desempenho inferior, os outros três algoritmos têm um desempenho bem similar.

O gráfico (b) apresentado na Figura 44 mostra que o algoritmo para escolha aleatória tem o pior resultado para a quantidade dos cálculos de distância efetuados. Os algoritmos que apresentam o melhor desempenho aqui são: *Hull of Foci* e o *Maximum of Sum of Distances*, com o algoritmo *Maximum of Minimum Distances* obtendo um desempenho bem próximo ao deles.

No gráfico (c) da Figura 44 os algoritmos *Hull of Foci* e *Maximum of Sum of Distances* apresentam melhor tempo médio para cada consulta, seguidos do algoritmo *Maximum of Minimum Distances*, e com o algoritmo Aleatório apresentando o pior resultado.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 44 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que o algoritmo o qual possui melhor desempenho nem sempre é o mesmo, com o Aleatório sendo o que mais aparece.

Os gráficos (a), (b) e (c) apresentados na Figura 45 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Cluster Uniform Distribution*, com 4 pivôs. O algoritmo que apresenta melhor desempenho para essa base de dados é o *Maximum of Minimum Distances*.

O gráfico (a) apresentado na Figura 45 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) começa com um bom desempenho, porém é

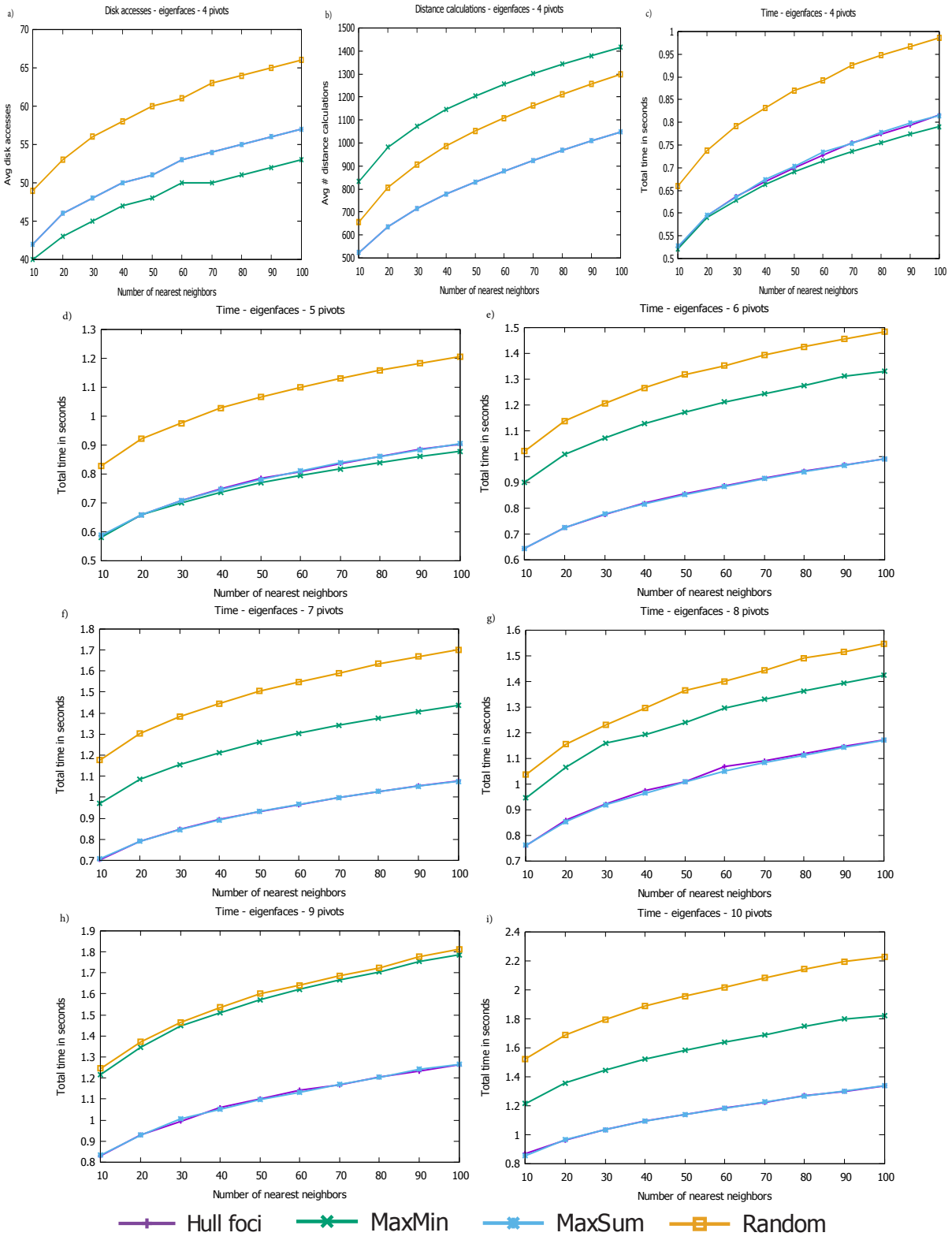

Figura 43 – Gráficos de desempenho dos testes realizados usando a base de dados *Eigenfaces* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

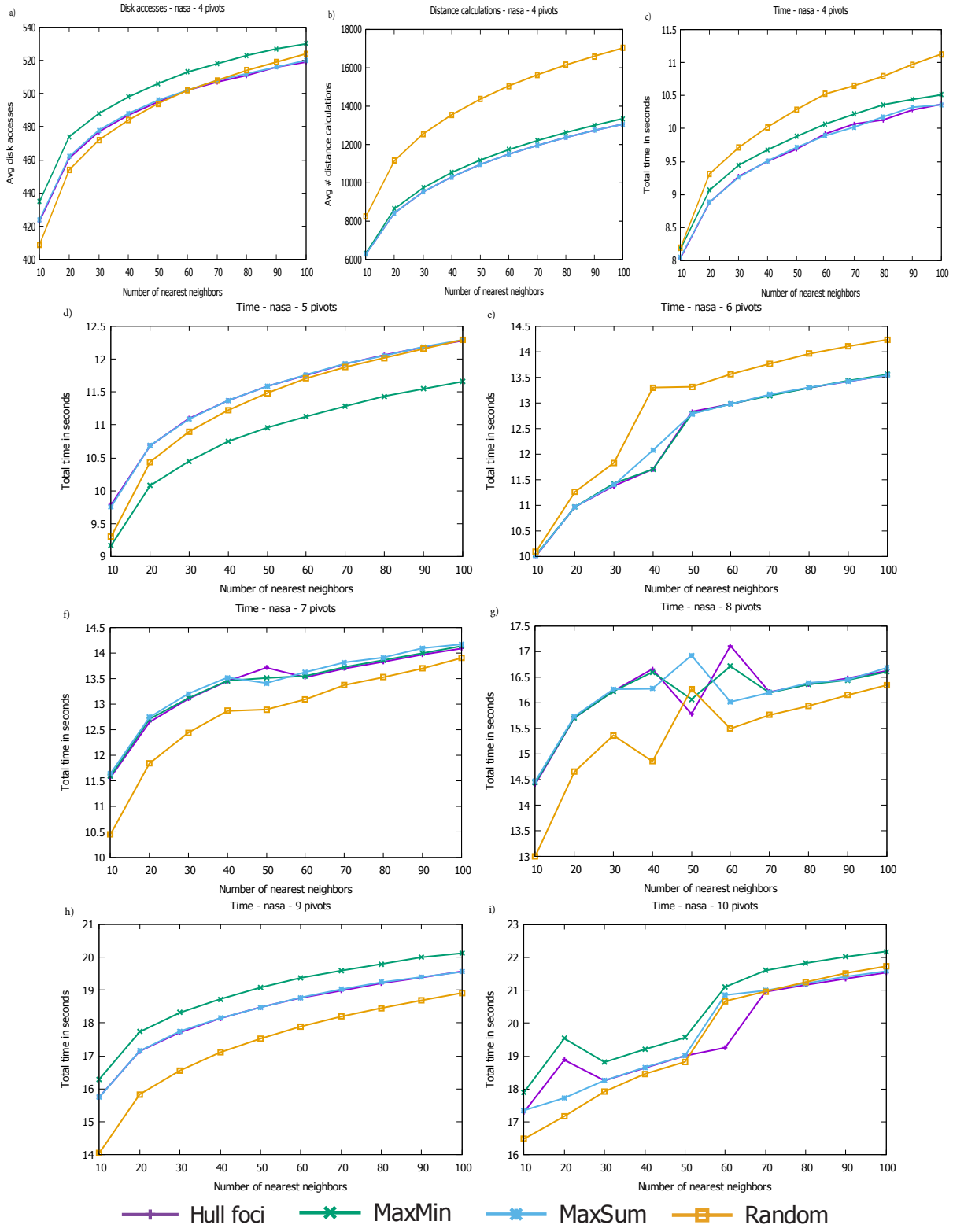

Figura 44 – Gráficos de desempenho dos testes realizados usando a base de dados *Nasa* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

rapidamente substituído pelo algoritmo Aleatório.

O gráfico (b) da Figura 45 mostra que o algoritmo para escolha aleatória tem o pior resultado para a quantidade dos cálculos de distância efetuados. O algoritmo que apresenta o melhor desempenho é o *Maximum of Minimum Distances*.

No gráfico (c) apresentado na Figura 45 o algoritmo que apresenta o melhor desempenho é o *Maximum of Minimum Distances*, seguido pelo algoritmo Aleatório. Aqui é possível notar que a quantidade de acesso ao disco tem um custo mais elevado do que o número dos cálculos de distância efetuados, pois o algoritmo aleatório tem um bom desempenho com relação a quantidade de acesso ao disco e mesmo realizando um número elevado de cálculos de distância ele ainda tem o segundo melhor tempo médio.

Os gráficos (d), (e), (f), (g), (h) e (i) apresentados na Figura 45 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que o algoritmo aleatório é o mais regular.

Os gráficos (a), (b) e (c) da Figura 46 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Sierpinski*, com 4 pivôs. O algoritmo que apresenta melhor desempenho para essa base de dados é o *Maximum of Minimum Distances*.

O gráfico 46 (a) é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que o algoritmo *Maximum of Minimum Distances* (MICÓ; ONCINA; VIDAL, 1994; SOCORRO; MICÓ; ONCINA, 2011) apresenta o melhor desempenho, com o algoritmo Aleatório apresentando o pior desempenho.

O gráfico (b) apresentados na Figura 46 mostra que o algoritmo para escolha aleatória tem o melhor resultado para a quantidade dos cálculos de distância efetuados, com os algoritmos *Hull of Foci* e *Maximum of Sum of Distances* apresentando o pior desempenho.

No gráfico (c) da Figura 46 o algoritmo que apresenta o melhor desempenho é o *Maximum of Minimum Distances*, com algoritmo Aleatório apresentando o pior desempenho. Nesse caso o pior desempenho apresentado pelo algoritmo Aleatório é justificado pelo fraco rendimento com relação ao acesso a disco, mesmo ele tendo um desempenho muito bom com relação ao número dos cálculos de distância a média tempo foi a pior dentre os quatro algoritmos.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 46 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que para todos esses experimentos os algoritmos *Hull of Foci* e *Maximum of Sum of Distances* apresentam desempenho similar, onde foram sempre os melhores.

Os experimentos analisados até o momentos todos utilizam a medida de dissimilaridade Euclidiana, os experimentos a seguir são referentes a medida de dissimilaridade FastEMD.

Os gráficos (a), (b) e (c) apresentados pela Figura 47 são referentes a melhor confi-

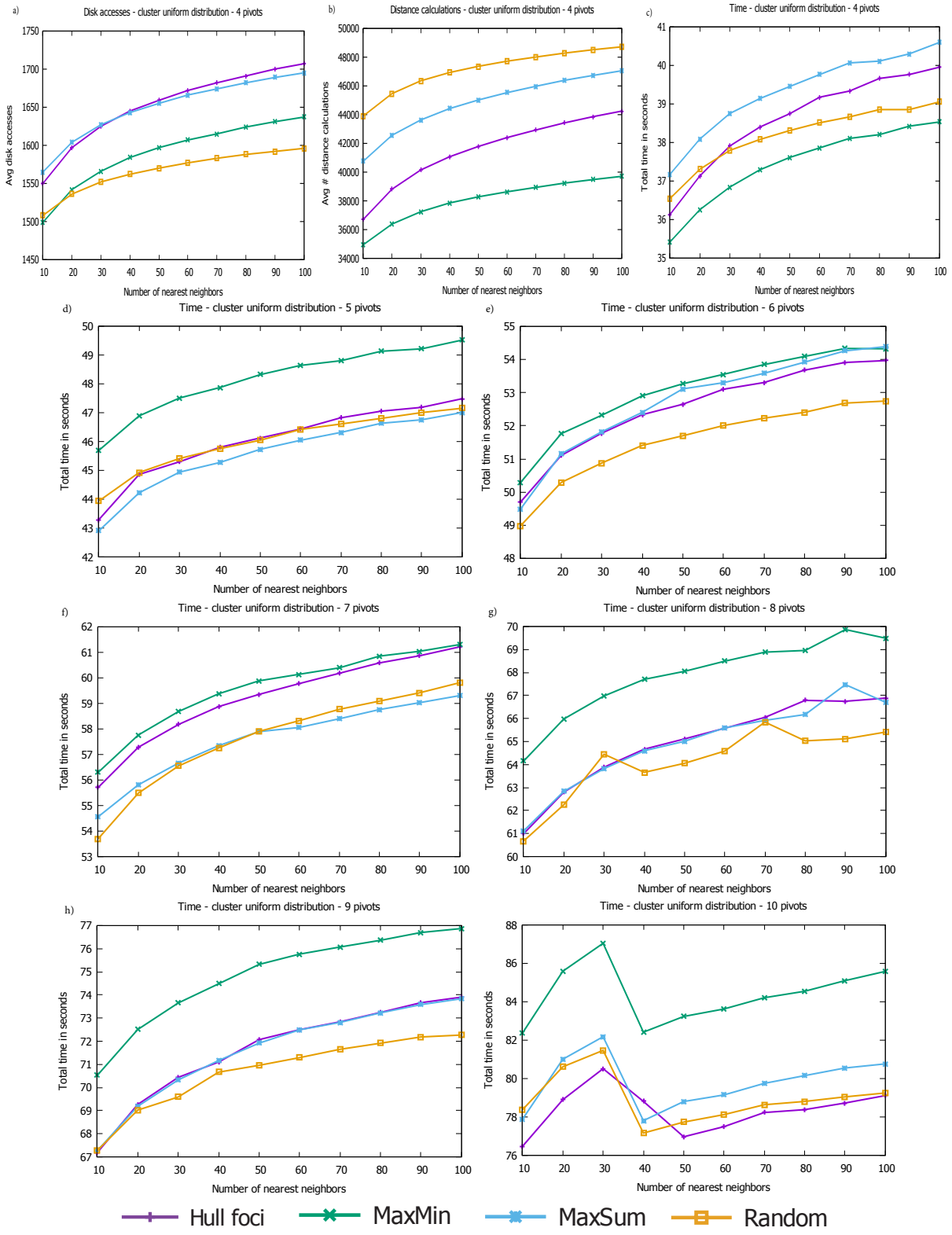

Figura 45 – Gráficos de desempenho dos testes realizados usando a base de dados *Cluster Uniform Distribution* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

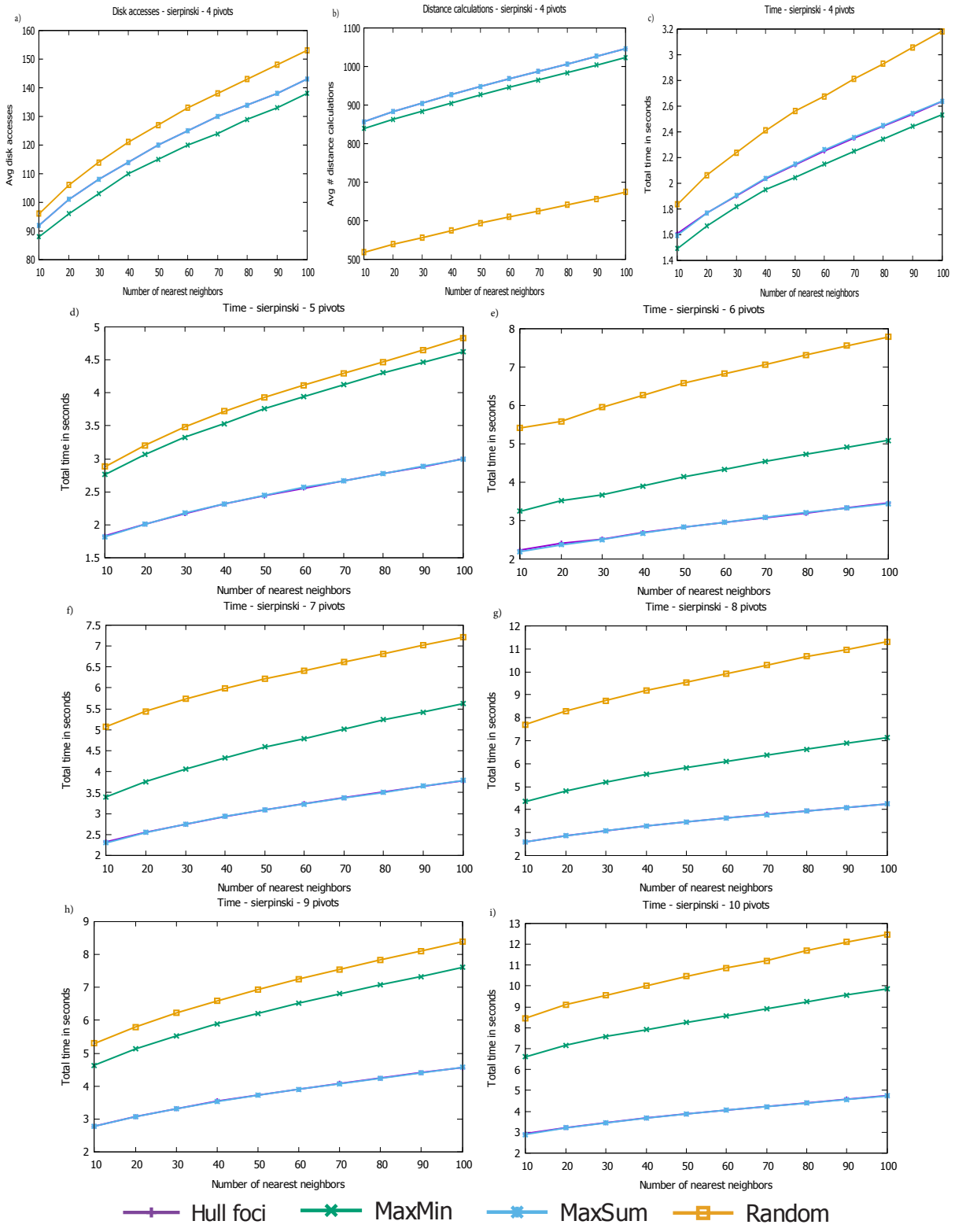

Figura 46 – Gráficos de desempenho dos testes realizados usando a base de dados *Sierpinski* e medida de dissimilaridade Euclidiana. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

guração para uma consulta knn utilizando a base de dados *Corel*, observando os gráficos é possível notar que para uma consulta o ideal é utilizar um algoritmo para escolher de forma aleatória 4 pivôs, porém os outros três algoritmos apresentam um desempenho muito similar ao aleatório.

O gráfico (a) da Figura 47 é referente à quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados, é possível notar por meio desse gráfico que os quatro algoritmos apresentam o mesmo desempenho.

O gráfico (b) apresentado na Figura 47 mostra que o algoritmo *Maximum of Sum of Distances* começa com o desempenho um pouco melhor do que os outros três algoritmos, porém rapidamente o desempenho entre os algoritmos se iguala.

No gráfico (c) da Figura 47 o algoritmo para escolha aleatória dos pivôs é o que apresenta melhor tempo médio para cada consulta, com os outros três algoritmos avaliados com um tempo médio similar, é possível dizer que neste caso poderia ser escolhido qualquer algoritmo para experimentos futuros.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 47 apresentam apenas a informação de tempo médio gasto em cada consulta. Esses gráficos apresentam um aumento de tempo médio gasto para cada algoritmo sempre que a quantidade de pivôs aumenta, com o desempenho entre os algoritmos muito similar.

Os gráficos (a), (b) e (c) apresentados na Figura 48 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Eigenfaces*. Para essa base de dados a melhor configuração foi utilizando 4 pivôs, com os algoritmos apresentando praticamente o mesmo desempenho.

O gráfico (a) da Figura 48 é referente a quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que todos os quatro algoritmos avaliados apresentam o mesmo desempenho.

O gráfico (b) apresentado na Figura 48 mostra o desempenho com relação ao número dos cálculos de distância realizados, aqui todos os quatro algoritmos apresentam o mesmo desempenho.

No gráfico (c) da Figura 48 é possível notar que o desempenho dos quatro algoritmos é muito similar, logo para experimentos futuros pode-se escolher qualquer um dos algoritmos analisados.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 48 apresentam apenas a informação de tempo médio gasto em cada consulta. Esses gráficos apresentam um aumento de tempo médio gasto para cada algoritmo sempre que a quantidade de pivôs aumenta, com o desempenho entre os algoritmos bastante similar.

Os gráficos (a), (b) e (c) apresentados na Figura 49 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Nasa* e 4 pivôs. Para essa base de dados os algoritmos: *Hull of Foci*, *Maximum of Minimum Distances* e *Maximum of Sum of Distances* apresentam um desempenho muito similar, os três algoritmos podem

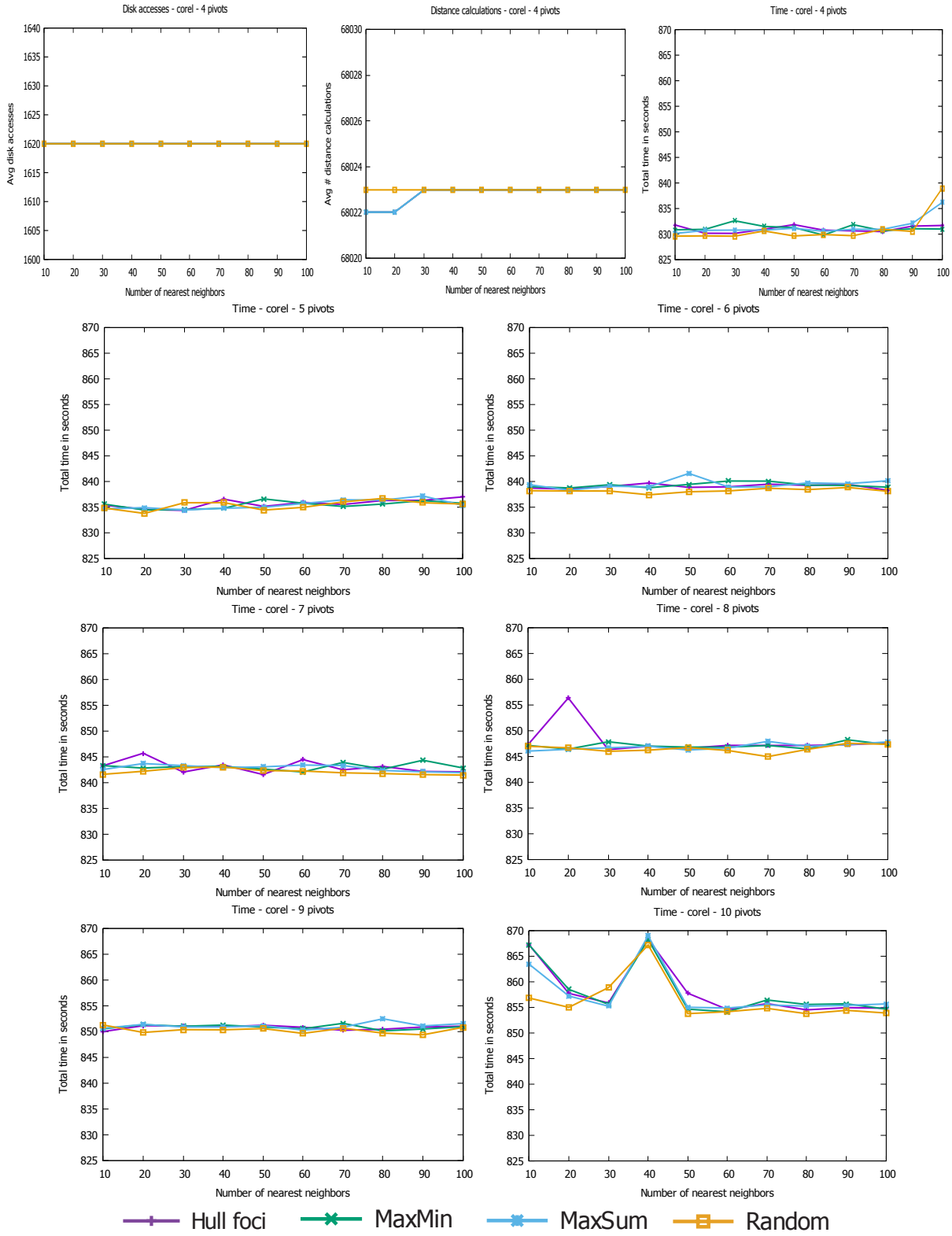

Figura 47 – Gráficos de desempenho dos testes realizados usando a base de dados *Corel* e medida de dissimilaridade *FastEMD*. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

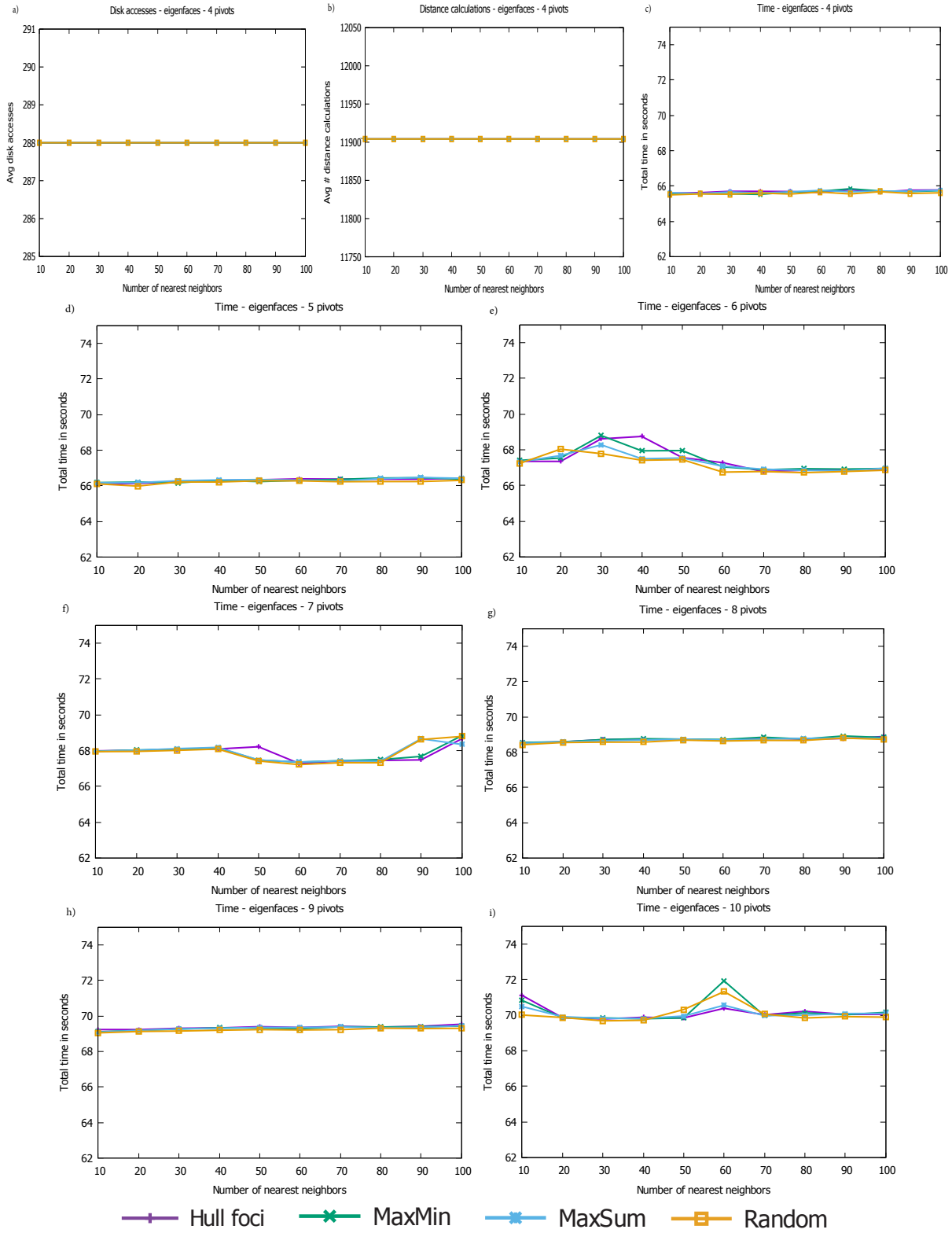

Figura 48 – Gráficos de desempenho dos testes realizados usando a base de dados *Eigenfaces* e medida de dissimilaridade *FastEMD*. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

ser considerados como uma melhor escolha. Para essa configuração apenas o algoritmo aleatório apresenta um desempenho inferior.

O gráfico (a) da Figura 49 é referente a quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que o algoritmo Aleatório apresenta um desempenho superior, os outros três algoritmos têm um desempenho similar, porém muito inferior.

O gráfico (b) da Figura 49 mostra que o algoritmo para escolha aleatória tem o pior resultado para a quantidade dos cálculos de distância efetuados. Os outros três algoritmos são similares e possuem o melhor desempenho.

No gráfico (c) apresentado na Figura 49 os algoritmos *Hull of Foci*, *Maximum of Sum of Distances* e *Maximum of Minimum Distances* apresentam melhor tempo médio para cada consulta, o algoritmo Aleatório apresentando o pior resultado. Para esse experimento o alto número dos cálculos de distância efetuados usando o algoritmo aleatório fez diferença.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 49 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que o algoritmo o qual apresenta melhor resultado varia bastante.

Os gráficos (a), (b) e (c) apresentados na Figuras 50 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Cluster Uniform Distribution*, com 4 pivôs. O algoritmo que apresenta melhor desempenho para essa base de dados é o *Hull of Foci*, com o algoritmo *Maximum of Sum of Distances* obtendo um desempenho muito semelhante.

O gráfico (a) apresentado na Figura 50 é referente a quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que o algoritmo *Maximum of Minimum Distances* tem o melhor desempenho, com os algoritmos *Hull of Foci* e Maximum of Sum of Distances também apresentando um bom desempenho.

O gráfico (b) da Figura 50 mostra que o algoritmo Hull of Foci começa apresentando um bom resultado com o algoritmo *Maximum of Sum of Distances* muito próximo, no meio do gráfico o algoritmo *Maximum of Sum of Distances* obtém desempenho semelhante ao Hull of Foci chegando a ser ligeiramente superior ao final.

No gráfico (c) apresentado na Figura 50 o algoritmo que apresenta o melhor desempenho é o *Hull of Foci*, seguido pelo algoritmo Maximum of Sum of Distances. O algoritmo aleatório como apresentou pior desempenho com relação a quantidade de acesso ao disco e número dos cálculos de distância, apresenta o pior tempo.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 50 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que o desempenho dos algoritmos *Hull of Foci*, Maximum of Sum of Distances e *Maximum of*

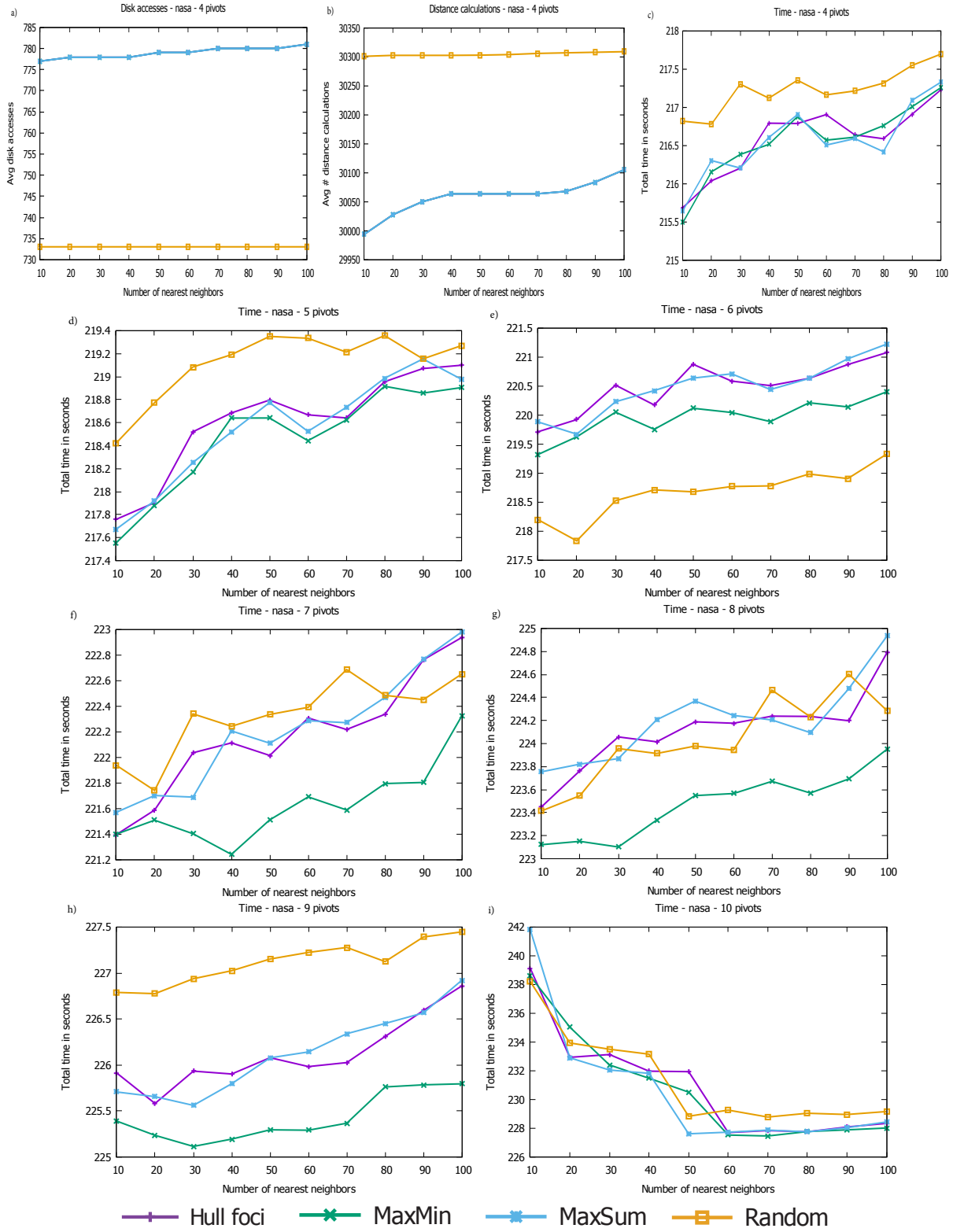

Figura 49 – Gráficos de desempenho dos testes realizados usando a base de dados *Nasa* e medida de dissimilaridade *FastEMD*. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

Minimum Distances apresentam desempenho muito similar, com o Aleatório apresentando sempre o pior desempenho.

Os gráficos (a), (b) e (c) apresentados na Figura 51 são referentes a melhor configuração para uma consulta knn utilizando a base de dados *Sierpinski*, com 4 pivôs. Os algoritmos *Hull of Foci*, *Maximum of Sum of Distance* e *Maximum of Minimum Distance* apresentam desempenho muito similar, é possível dizer que qualquer um desses três algoritmos seria uma boa escolha.

O gráfico (a) apresentado na Figura 51 é referente a quantidade de acesso ao disco que é feita para cada um dos quatro algoritmos avaliados. É possível notar por meio desse gráfico que os algoritmos *Hull of Foci*, *Maximum of Sum of Distance* e *Maximum of Minimum Distance* apresentam o melhor desempenho, com o algoritmo Aleatório apresentando o pior desempenho.

O gráfico (b) da Figura 51 mostra que o algoritmo para escolha aleatória tem o melhor resultado para a quantidade dos cálculos de distância efetuados, com os algoritmos *Hull of Foci* e *Maximum of Sum of Distance* apresentando o pior desempenho.

No gráfico (c) da Figura 51 é difícil distinguir o algoritmo que apresenta o melhor desempenho, observando o gráfico é possível dizer que os algoritmos *Hull of Foci*, *Maximum of Sum of Distance* e *Maximum of Minimum Distance* apresentam desempenho similar e qualquer um dos três algoritmos seria uma boa escolha. Nesse caso o pior desempenho apresentado pelo algoritmo Aleatório é justificado pelo fraco rendimento com relação ao acesso a disco, mesmo ele tendo um desempenho muito bom com relação ao número dos cálculos de distância realizados a média tempo foi a pior dentre os quatro algoritmos.

Os gráficos (d), (e), (f), (g), (h) e (i) da Figura 51 apresentam apenas a informação de tempo médio gasto em cada consulta. Pode-se notar que o tempo gasto aumenta de acordo com a quantidade de pivôs usados na consulta, também é possível notar que para todos esses experimentos os algoritmos *Hull of Foci*, *Maximum of Minimum Distance* e *Maximum of Sum of Distance* apresentam desempenho similar, onde foram sempre os melhores.

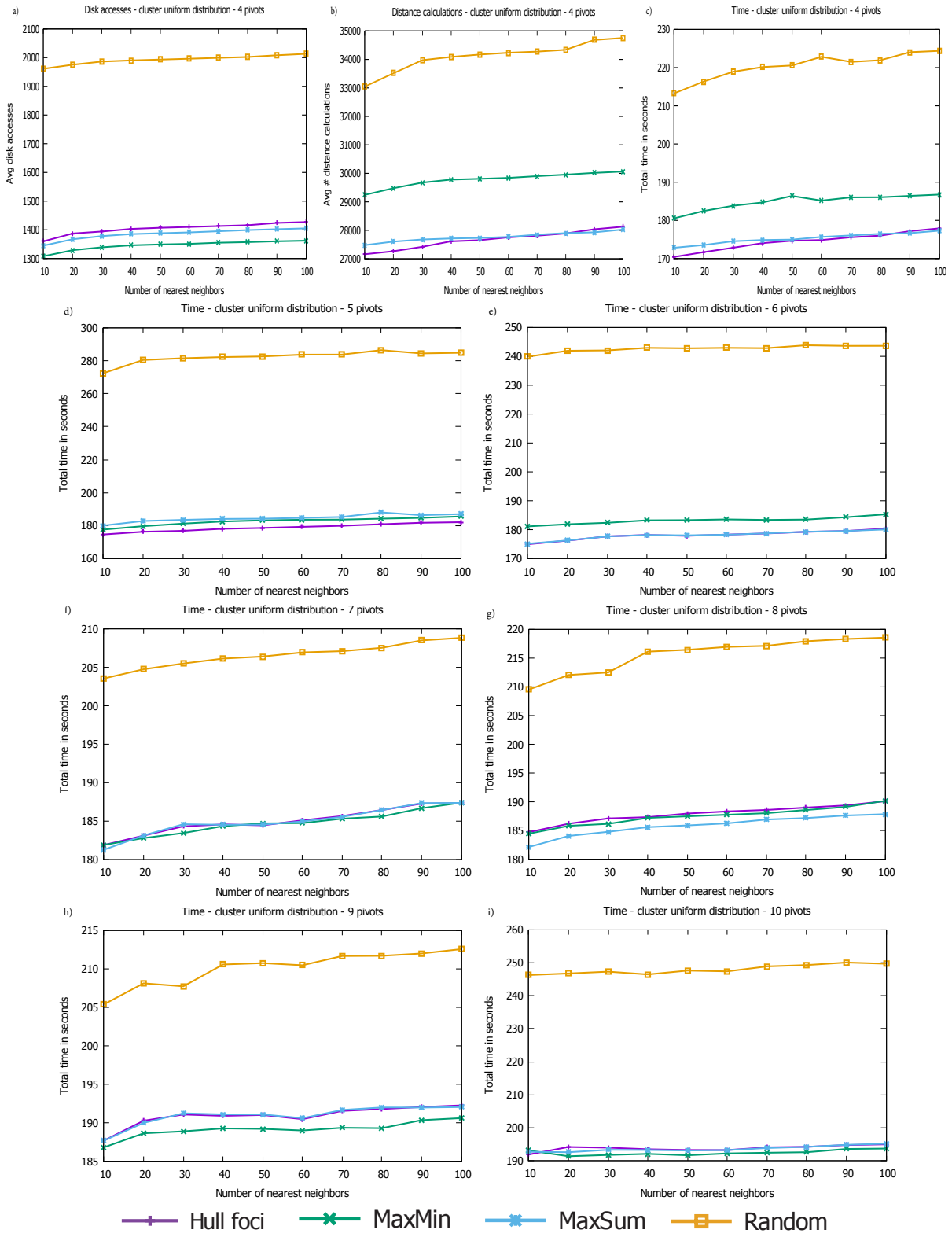

Figura 50 – Gráficos de desempenho dos testes realizados usando a base de dados *Cluster Uniform Distribuition* e medida de dissimilaridade *FastEMD*. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.

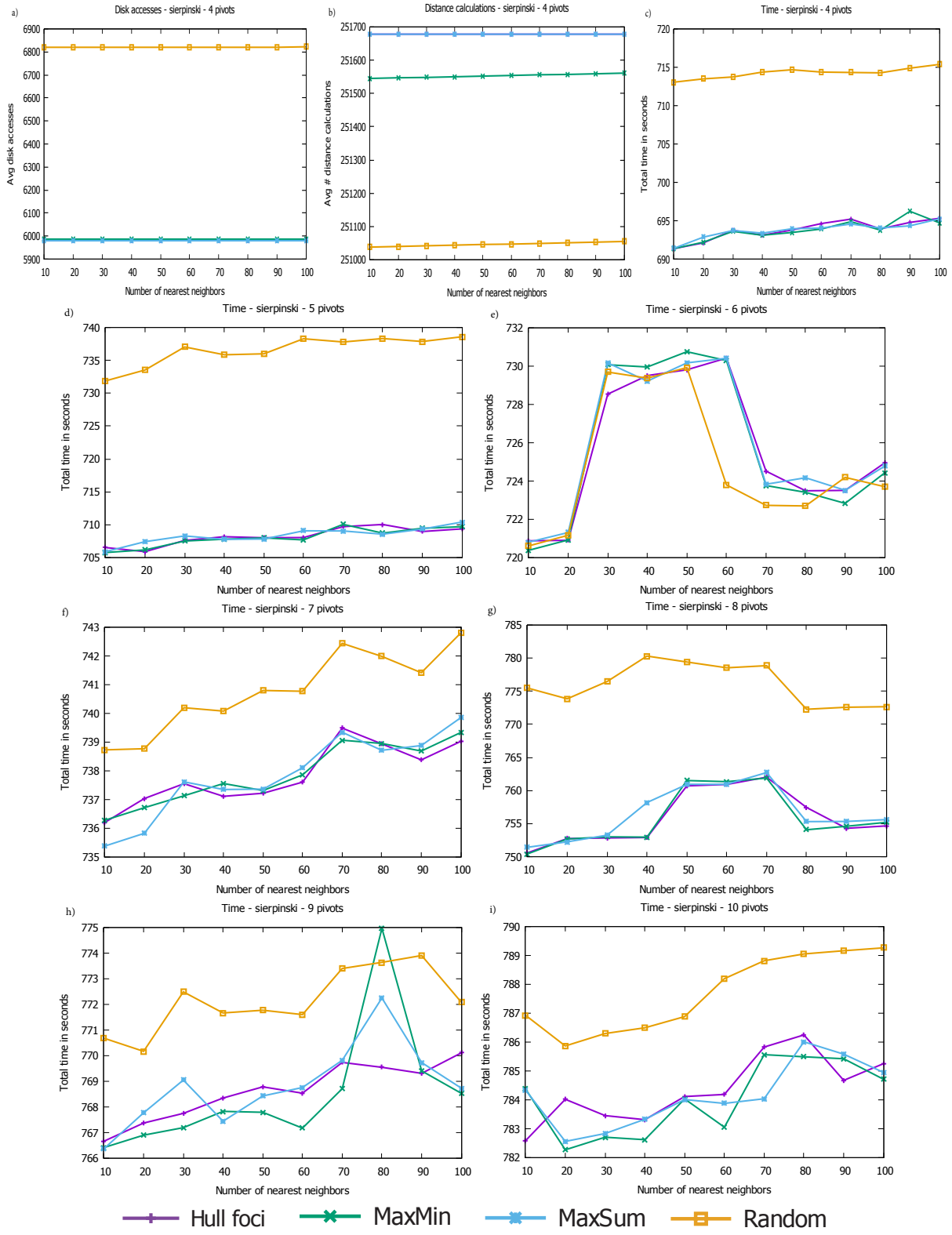

Figura 51 – Gráficos de desempenho dos testes realizados usando a base de dados *Sierpinski* e medida de dissimilaridade *FastEMD*. Em (a) é apresentado o desempenho com relação ao acesso à disco, em (b) desempenho de acordo com cálculos de distância realizados, e nos outros gráficos são apresentados o desempenho com relação a tempo de processamento.