Estudo de Técnicas para Indexação e Recuperação de Sequências Numéricas: Segmentação Adaptativa e Processamento de Consultas em Lote

Luiz Fernando Afra Brito

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

Uberlândia
2018

Luiz Fernando Afra Brito

Estudo de Técnicas para Indexação e Recuperação de Sequências Numéricas: Segmentação Adaptativa e Processamento de Consultas em Lote

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. Marcelo Keese Albertini

Uberlândia
2018

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

B862e
2018

Brito, Luiz Fernando Afra, 1991-
Estudo de técnicas para indexação e recuperação de sequências numéricas: segmentação adaptativa e processamento de consultas em lote / Luiz Fernando Afra Brito. - 2018.
105 f. : il.

Orientador: Marcelo Keese Albertini.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Disponível em: http://dx.doi.org/10.14393/ufu.di.2018.253
Inclui bibliografia.

1. Computação - Teses. I. Albertini, Marcelo Keese. 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 – UFU
FACULDADE DE COMPUTAÇÃO – FACOM
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO – PPGCO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada **"Estudo de Técnicas para Indexação e Recuperação de Sequências Numéricas: Segmentação Adaptativa e Processamento de Consultas em Lote"** por **Luiz Fernando Afra Brito** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, ___ de _____ de _____

Orientador: _____

Prof. Dr. Marcelo Keese Albertini
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Humberto Luiz Razente
Universidade Federal de Uberlândia

_____

Prof. Dr. Ricardo Araújo Rios
Universidade Federal da Bahia

Dedico este trabalho a todos os meus familiares, especialmente a minha mãe, que sempre me motivou sendo uma pessoa guerreira e enfrentando os seus próprios desafios de maneira implacável.

Agradecimentos

Agradeço primeiramente a minha família, que esteve sempre ao meu lado mesmo nos momentos mais difíceis. Agradeço também aos meus amigos mais próximos, aos meus colegas de laboratório, aos meus professores e ao meu orientador Marcelo Keese Albertini. Todos foram muito prestativos e sempre estiveram à disposição, meu orientador principalmente, que me ajudou em todas as etapas deste curso de pós graduação e me ensinou uma das coisas mais importantes e complicadas do mundo: escrever. Por último, agradeço à Coordenação de Aperfeiçoamento de Pessoa de Nível Superior (CAPES) pelo auxílio financeiro concedido para que eu me estabelecesse na cidade de Uberlândia durante esses dois anos.

"Você não precisa de um motivo para ajudar as pessoas"
(Zidane Tribal)

Resumo

Estruturas de indexação e algoritmos especializados de busca provêm consultas por similaridade. De acordo com a literatura atual, consultas por similaridade devem ser rápidas e utilizar o mínimo de espaço possível. Nesta dissertação foram estudadas abordagens para atender a esses requisitos no contexto de sequências numéricas. Na primeira abordagem foram propostas duas representações reduzidas das sequências para a criação de medidas *lower bounding* da distância euclidiana, sendo elas: *Error-Bounded Piecewise Linear Approximation* (EBPLA) e *Adaptive Indexable Piecewise Linear Approximation* (AIPLA). De modo inovador, essas duas propostas armazenaram um conjunto de coeficientes de tamanho adaptável às características das sequências. Em experimentos, a representação EBPLA, apesar de flexível, obteve erro de aproximação alto e, consequentemente, a eficiência de sua medida *lower bounding* foi inferior as outras representações. A outra proposta, AIPLA, proporcionou menores erros de aproximação e sua medida *lower bounding* foi comparável ás criadas a partir de representações tradicionais como *Piecewise Aggregate Approximation* (PAA) e *Indexable Piecewise Linear Approximation* (IPLA). A segunda abordagem teve como objetivo reduzir o tempo de consultas por meio do agrupamento de sequências de consulta enviadas em lote. Primeiramente formaram-se grupos de consultas para que, posteriormente, apenas uma varredura por grupo em *R-Trees* e *M-Trees* foi realizada. Ao todo foram avaliadas 5 estratégias para agrupar as consultas. Os resultados observados indicam que a estratégia que economiza mais acessos a memória secundária é aquela que cria um único grupo contendo todas as sequências de consulta. Entretanto, dependendo do tamanho do lote de consultas, a necessidade de espaço em memória principal pode aumentar consideravelmente ao utilizar essa estratégia. Por isso, em casos onde a quantidade de memória principal é limitada, sugere-se o uso da estratégia que cria N grupos a partir de N sequências de consultas escolhidas aleatoriamente.

Palavras-chave: agrupamento; busca em lote; consulta por similaridade; indexação; *lower bounding*; redução de dimensionalidade; sequência.

Abstract

Indexing structures and specialized search algorithms provide similarity queries. According to current literature, similarity queries should be fast and minimize the amount of space required. In this master's thesis, we studied two approaches in order to meet these requirements in the context of numeric sequences. In the first approach, we proposed two representations to approximate sequences and to create lower bounding measures to the euclidian distance: *Error-Bounded Piecewise Linear Approximation* (EBPLA) and *Adaptive Indexable Piecewise Linear Approximation* (AIPLA). In an innovative way, these two representations stored a set of coefficients such that its size was proportionally to the characteristics of the sequences. In experiments, the EBPLA, although flexible, obtained high approximation error and, consequently, the efficiency of its lower bounding was lower than the other representations. The other proposed representation, the AIPLA, provided the lowest approximation error and its lower bounding was similar to well known representations such as *Piecewise Aggregate Approximation* (PAA) and *Indexable Piecewise Linear Approximation* (IPLA). In the second approach we grouped query sequences, sent as batches, in order to reduce the time of similarity queries. Firstly we formed groups of queries and then we searched through indexing structures, such as R-Trees and M-Trees, only once. In our experiments, we evaluated 5 different strategies to group sequences. The results indicate the overall best strategy for grouping queries, the one which saved more access to secondary memory, is the one that unifies all queries in a single group. However, this grouping strategy can considerably increase the usage of primary memory for large batches. Therefore, in scenarios where primary memory is limited, we suggest the use of the strategy which creates N clusters from N initial sequences chosen randomly.

Keywords: batch mode; clustering; dimensionality reduction; indexing; lower bounding; sequence; similarity query.

Lista de ilustrações

Lista de tabelas

Lista de siglas

AIPLA *Adaptive Indexable Piecewise Linear Approximation*

AG *Adaptive Grouping*

APCA *Adaptive Piecewise Constant Approximation*

CAPES Coordenação de Aperfeiçoamento de Pessoa de Nível Superior

DFT *Discrete Fourier Transform*

DWT *Discrete Wavelet Transform*

EBPLA *Error-Bounded Piecewise Linear Approximation*

FCS *Feasible Coefficient Space*

FFT *Fast Fourier Transform*

FS *Feasible Space*

FSW *Feasible Space Window*

FCSW *Feasible Coefficient Space Window*

HWT *Haar Wavelet Transform*

IPLA *Indexable Piecewise Linear Approximation*

KMG *K-Medoids Grouping*

LSH *Locality Sensitive Hashing*

MCG *Maximum Capacity Grouping*

MBR *Minimum Bounding Rectangle*

NG *No Grouping*

NRG *N-Random Grouping*

PAA *Piecewise Aggregate Approximation*

PAM *Partition Around Medoids*

PIP *Perceptually Important Points*

PLA *Piecewise Linear Approximation*

SBBD Simpósio Brasileiro de Banco de Dados

SG *Single Grouping*

Lista de notações

Símbolo	Descrição
$S = \langle s_1, s_2, \ldots, s_n \rangle$	Sequência numérica de tamanho n
$\lvert S \rvert$	Cardinalidade de S
s_i	i-ésimo elemento da sequência S
$\lvert s_i \rvert$	Valor absoluto de s_i
$S_{i \ldots j}$	Segmento de S que começa em s_i e termina em s_j
$\bar{S}_{i \ldots j}$	Média dos valores do segmento $S_{i \ldots j}$
$dist(S, Q)$	Distância euclidiana entre sequências S e Q
$X = \langle x_1, x_2, \ldots, x_m \rangle$	Representação aproximada de sequência de tamanho m
x_j	j-ésimo elemento da representação X
$lb_R(X, Y)$	*Lower bounding* da distância euclidiana entre representações X e Y do tipo R
$p = (p_1, p_2)$	Ponto com abscissa p_1 e ordenada p_2
$D = \langle S_1, S_2, \ldots, S_n \rangle$	Conjunto com n sequências numéricas
D_{ij}	j-ésimo elemento da i-ésima sequência no conjunto D

Sumário

Capítulo 1

Introdução

Sequências numéricas são dados observados ao longo do tempo onde leituras podem ser realizadas entre intervalos constantes (HAMILTON, 1994). A cotação diária do *Bitcoin*, o preço mensal de ações e a temperatura anual média são exemplos de dados no formato de sequências numéricas. Em áreas como economia e sensoriamento remoto, a decomposição de sequências numéricas em componentes de sazonalidade e tendência permite analisar o comportamento das variáveis observadas para inferir informações relevantes sobre o problema (PATEL et al., 2015; HUTCHINSON et al., 2015). Na área de mineração de dados, essas sequências são utilizadas em tarefas como: classificação (XI et al., 2006), agrupamento (EISEN et al., 1998), previsão (JAIN; KUMAR, 2007), detecção de anomalias ou novidades (PIMENTEL et al., 2014) e detecção de *motifs* (MUEEN et al., 2009).

Devido à importância de sequências numéricas e à quantidade de dados produzidos nesse formato torna-se necessário o uso de algoritmos e estruturas de dados especializados. Existem duas etapas fundamentais na manipulação de grandes bases de sequências: a etapa de indexação e a etapa de recuperação. Na etapa de indexação, sequências são inseridas em estruturas de dados que permitam a implementação de métodos de acesso multidimensionais ou métricos. Na etapa de recuperação são realizadas consultas nas estruturas de dados baseadas na similaridade entre sequências (AGRAWAL; FALOUTSOS; SWAMI, 1993). Nesse tipo de consulta, usuários fornecem uma sequência de exemplo para que sejam recuperadas aquelas mais similares pertencentes à base de dados.

De acordo com (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994), consultas por similaridade podem ser classificadas de acordo com a maneira com que sequências de exemplo são comparadas com as sequências indexadas, podendo ser: por correspondência de sequência inteira, em inglês, *whole sequence matching* (AGRAWAL; FALOUTSOS; SWAMI, 1993), ou por correspondência de subsequência, em inglês, *subsequence matching* (MOON; WHANG; HAN, 2002). Na comparação por correspondência de sequência inteira, dado um conjunto de sequências previamente indexadas e uma sequência de consulta Q, todas de mesmo tamanho, o objetivo é encontrar, dentre as sequências indexadas, aquelas que forem mais similares à Q. Ao considerar similaridade por correspondência

de subsequência, deseja-se encontrar entre as sequências indexadas aquelas que possuam subsequências similares à Q. Nesse caso as sequências não precisam ter o mesmo tamanho.

Para a implementação desse tipo de consulta, a escolha da medida de similaridade adequada é uma parte importante para o desenvolvimento de algoritmos de indexação e recuperação de sequências. Em (BATISTA et al., 2014), os autores apontaram que a medida de similaridade deve ser definida de acordo com as invariâncias exigidas pelo domínio do problema. Por exemplo, o problema pode exigir invariância à diferentes escalas e deslocamentos da amplitude de sequências. Esse tipo de invariância pode ser obtido por meio da distância euclidiana antecedida da normalização *z-score* (RAKTHANMANON et al., 2012), que subtrai a média de cada sequência para alinhá-las verticalmente e divide pelo seu desvio padrão para normalizar as amplitudes conforme a distribuição normal.

De acordo com a literatura atual, consultas por similaridade devem satisfazer os seguintes requisitos: i) baixo tempo para execução de consultas; ii) pouco consumo de espaço para manutenção do índice; iii) pouco uso de memória principal durante a etapa recuperação; e iv) boa qualidade dos resultados recuperados. Os requisitos i), ii) e iii) estão relacionados a estruturas de dados, tais como *R-Trees* (GUTTMAN, 1984) e *M-Trees* (CIACCIA; PATELLA; ZEZULA, 1997), e aos algoritmos utilizados para manipulação dessas estruturas. O requisito iv) é influenciado pela escolha da medida de similaridade. Caso o método empregado seja aproximado (GIONIS; INDYK; MOTWANI, 1999), deve-se também considerar a taxa de falsos-negativos e, quando relevante, de falsos-positivos.

Uma maneira de melhorar o desempenho de consultas por similaridade é utilizar representações reduzidas para recuperação das sequências numéricas originais. Essas representações usam apenas uma fração do espaço das sequências originais e, além disso, a similaridade entre representações pode ser utilizada para descartar rapidamente sequências candidatas não promissoras, satisfazendo os requisitos i), ii) e iii). Em (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994), por exemplo, os autores mostraram que se $lb(X, Y) \leq dist(Q, S)$, onde X e Y são representações reduzidas das sequências Q e S, então $lb(X, Y)$ é *lower bounding* da medida de similaridade $dist(Q, S)$ e, consequentemente, o seu uso permite que consultas por similaridade que recuperam as sequências originais permaneçam exatas. Nesse trabalho, os autores utilizaram os primeiros k coeficientes produzidos pela *Discrete Fourier Transform* (DFT) para aproximar sequências de tamanho n, onde $k \ll n$, e criar uma medida *lower bounding* da distância euclidiana.

Sabe-se que representações que usam um número fixo de coeficientes para aproximar sequências numéricas são facilmente manipuladas pelos métodos de acesso tradicionais como vetores de características. Porém, essa predeterminação de coeficientes pode limitar a capacidade de representação das sequências e, consequentemente, faz com que medidas *lower bounding* criadas a partir dessas representações percam sua efetividade pois as distâncias estimadas são menos próximas das distâncias reais. Por isso, o estudo de

representações cuja quantidade de coeficientes seja proporcional à informação relevante contida em cada sequência pode melhorar o desempenho de consultas por similaridade.

Atualmente, poucos trabalhos utilizaram representações que aproximam sequências por meio de um conjunto de coeficientes de tamanho adaptável às suas características em consultas por similaridade. Um dos motivos é a dificuldade de criação de medidas *lower bounding*, uma vez que a comparação entre representações com quantidades diferentes de coeficientes pode ser complexa. Porém, de acordo com (KEOGH et al., 2001b), esse tipo de representação pode diminuir o erro de aproximação e, consequentemente, possibilita a criação de medidas *lower bounding* mais próximas das medidas de similaridade reais, permitindo que mais sequências candidatas sejam descartadas durante o processo de busca. Por exemplo, nesse mesmo trabalho, os autores propuseram a representação *Adaptive Piecewise Constant Approximation* (APCA) que aproxima cada sequência por médias associadas a segmentos de diferentes tamanhos. Desse modo, mais coeficientes foram usados para aproximar as partes com maior variância das sequências numéricas.

Outra maneira de acelerar consultas por similaridade é usar uma estratégia de agrupamento de sequências em ambientes onde múltiplas consultas são requisitadas em lote. Nessa técnica, agrupam-se consultas similares e realiza-se somente uma travessia na estrutura de indexação para cada grupo. Durante a etapa de recuperação, o algoritmo de busca mantém uma lista de resultados para cada consulta do grupo para que, posteriormente, sejam retornadas como resposta. Essa técnica pode diminuir o tempo total de processamento das consultas em um lote pois sequências de consultas similares percorrem caminhos semelhantes nas estruturas de indexação. Desse modo, os nós recuperados durante a busca são reutilizados para processar todas as consultas do grupo, diminuindo a quantidade de acessos à memória secundária e satisfazendo os requisitos i) e ii).

Para que consultas por similaridade em lote sejam eficientes, uma boa estratégia de agrupamento deve ser escolhida. A estratégia de consulta mais simples coloca todas as consultas em um mesmo grupo e realiza apenas uma varredura na estrutura de indexação (MOON; WHANG; LOH, 2001). Entretanto, se existirem sequências de consulta muito dissimilares, grande parte do índice pode ser recuperado. No pior caso, analisam-se todas as sequências da base para cada consulta recuperando-as por meio de acessos aleatórios. Outro problema é que uma lista de resultados para cada consulta precisa ser mantida, fazendo com que o uso de memória primária durante a busca seja alto. Por isso, a escolha da estratégia de agrupamento impacta diretamente no desempenho de consultas por similaridade em lote.

O problema de agrupamento de sequências é mencionado na literatura, porém é pouco discutido no contexto de consultas por similaridade. Por exemplo, em (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994) os autores compararam estratégias de agrupamento para encontrar hiper-retângulos a partir de um conjunto de sequências. Entretanto, esse processo é realizado durante a etapa de inserção de sequências em *R*-

Trees a fim de diminuir o espaço para armazenamento do índice. Em (FU et al., 2008a), motivados por sua simplicidade, os autores usaram a estratégia que cria um único grupo contendo todas as sequências do lote para realizar consultas por similaridade no estilo correspondência por subsequência. Porém, outras estratégias de agrupamento de consultas não foram comparadas ou avaliadas, uma vez que esse não foi o foco principal do trabalho. Desse modo, acredita-se que o estudo de estratégias de agrupamento de sequências de consulta pode contribuir para a melhor compreensão do método no contexto de consultas por similaridade.

Dessa maneira, nesta dissertação estudaram-se duas abordagens para acelerar buscas por similaridade utilizando métodos de acessos tradicionais tais como *R-Trees* e *M-Trees*. Na primeira abordagem, considerou-se que mais sequências candidatas podem ser descartadas durante o processo de recuperação caso sejam utilizadas representações que aproximem sequências de tamanhos iguais por meio de um conjunto de coeficientes de tamanho adaptável às suas características. Assim, formulou-se a seguinte hipótese:

Hipótese 1. *A utilização de representações que usam um conjunto de coeficientes de tamanho adaptável às características das sequências numéricas pode acelerar consultas por similaridade no estilo correspondência por sequência inteira.*

Para investigar a Hipótese 1, o objetivo foi propor e estudar técnicas eficientes baseadas em representações que aproximam sequências por meio de um conjunto retas, tais como a representação *Piecewise Linear Approximation* (PLA) (KEOGH et al., 2001c). Acreditou-se que essas representações podem obter melhores aproximações e, consequentemente, manter mais características das sequências originais. Dessa maneira, medidas *lower bounding* criadas a partir dessas representações podem ser mais eficientes no processamento de consultas por similaridade. Além disso, essas retas podem ser utilizadas em trabalhos futuros para estudo de tendências, atividade comum em áreas como economia e sensoriamento remoto.

Conforme os trabalhos correlatos, a representação PLA não é indexável. Por isso, neste trabalho foram estudadas duas novas representações: a representação *Error-Bounded Piecewise Linear Approximation* (EBPLA), apresentada no Capítulo 4, que modifica algoritmos tradicionais de extração de retas PLA de modo que os erros máximos em cada segmento também sejam armazenados para que, posteriormente, sejam usados no cálculo de uma nova medida *lower bounding*; e a representação *Adaptive Indexable Piecewise Linear Approximation* (AIPLA), descrita no Capítulo 5, que, por meio de um algoritmo adaptativo *top-down*, extrai retas, assim como a representação *Indexable Piecewise Linear Approximation* (IPLA) (CHEN et al., 2007), a sua versão não adaptativa, e, posteriormente, utiliza distâncias entre retas IPLA como parte do cálculo de uma nova medida *lower bounding*.

Os experimentos relacionados a essas representações, apresentados no Capítulo 7, avaliaram o erro médio de aproximação e o *pruning power* de suas medidas *lower bounding*. O erro médio de aproximação de uma representação descreve o quão próximo são as representações reduzidas das sequências originais. No caso onde sequências são aproximadas por retas, quanto menor o erro, melhor a qualidade das tendências. De maneira similar, o *pruning power* de uma representação descreve o quão próximo é a medida *lower bounding* da medida de similaridade entre as sequências originais. Quanto mais próximo, maior é a capacidade para descartar rapidamente candidatos não promissores durante o processamento de consultas por similaridade.

Nos experimentos, observou-se que a representação EBPLA, mesmo utilizando um algoritmo estado da arte para extração das retas, não conseguiu uma boa aproximação e, consequentemente, a medida *lower bounding* proposta obteve *pruning power* baixo. Isso aconteceu devido à necessidade de armazenar o dobro de coeficientes por reta. Entretanto, acredita-se que, caso seja possível reduzir ou eliminar a necessidade de coeficientes adicionais, uma variação da representação EBPLA pode ser promissora em trabalhos futuros.

Por outro lado, a representação AIPLA, proposta nesta dissertação, foi a representação com menor erro médio de aproximação dentre todas avaliadas. Observou-se também que a medida *lower bounding* proposta obteve *pruning power* similar às representações *Piecewise Aggregate Approximation* (PAA), DFT e IPLA quando o número de coeficientes considerados é maior que 16. Portanto, mostrou-se ser possível utilizar a representação AIPLA para acelerar consultas por similaridade no estilo correspondência por sequência inteira. Acredita-se que outras medidas *lower bounding* para a representação AIPLA podem ser estudadas em trabalhos futuros uma vez que, neste trabalho, utilizou-se uma estratégia pessimista apresentada no Capítulo 5.

Na segunda abordagem estudada nesta dissertação, considerou-se também que, no cenário onde múltiplas consultas por similaridade são requisitadas simultaneamente, ou em lote, o agrupamento dessas consultas pode diminuir o tempo total de processamento. Assim, formulou-se também a seguinte hipótese:

Hipótese 2. *Estratégias que agrupam sequências de consulta a fim de realizar consultas por similaridade em lote podem reduzir o tempo total de processamento.*

Para investigar a Hipótese 2, o objetivo foi implementar e estudar cinco estratégias de agrupamento de sequências, apresentadas no Capítulo 6, para realizar consultas por similaridade em lote. A estratégia que cria um único grupo contendo todas as sequências do lote é referenciada neste trabalho como estratégia *Single Grouping* (SG). Além dessa, estudou-se as estratégias: *N-Random Grouping* (NRG), que seleciona N sequências de consulta do lote aleatoriamente e cria N grupos a partir dessas sequências; *Maximum Capacity Grouping* (MCG), que insere as sequências de consulta no grupo atual até que

a sua capacidade máxima seja atingida; *Adaptive Grouping* (AG), que usa uma função de custo para verificar se uma sequência deve ser inserida em um dos grupos já criados ou se um novo deve ser criado; e *K-Medoids Grouping* (KMG), que utiliza o algoritmo apresentado em (PARK; JUN, 2009) para encontrar K grupos formados a partir de K *medoids*.

Os experimentos relacionados a essas estratégias de agrupamento, apresentados no Capítulo 7, compararam as cinco estratégias com a abordagem tradicional que realiza uma consulta por similaridade para cada sequência do lote de consultas. Esses experimentos mensuraram a economia de acessos à memória secundária e a economia de cálculos de distância no processamento dos lotes de consultas. Dessa maneira, é possível estimar o tempo total de processamento de consultas por similaridade em lote uma vez que essas duas medidas são responsáveis pelo maior tempo despendido durante o processamento dos algoritmos de busca. Além disso, dois tipos de consultas por similaridade foram testados: busca por abrangência, que dado um raio de abrangência ϵ e uma sequência de exemplo Q, retorna todas as sequências previamente indexadas que possuem para Q menor que ϵ; e busca pontual, referenciada nesta dissertação como busca do tipo contém, que verifica se uma dada sequência pertence ao índice.

De acordo com os experimentos, a melhor estratégia, na maioria dos cenários observados, foi a estratégia SG, que cria um único um único grupo e realiza apenas uma travessia na estrutura de indexação. Mesmo com sua simplicidade, essa estratégia obteve uma economia de acessos à memória secundária de até 2^{10} vezes para lotes com 1000 sequências mantendo-se uma quantidade de cálculos similar à abordagem tradicional. Esse comportamento foi observado tanto *R-Trees* quanto *M-Trees*. Nota-se que, dependendo do tamanho do lote, o uso de memória principal pode aumentar consideravelmente. Por isso, em sistemas onde a memória principal é limitada, a estratégia NRG, que seleciona N sequências de consulta aleatórias para iniciar grupos, é recomendada uma vez que ela também obteve boa economia de acessos à memória secundária e fornece um mecanismo para limitar o uso de memória principal.

Portanto, esta dissertação fez as seguintes contribuições:

1. propôs-se a representação EBPLA que aproximou sequências numéricas por meio de pontos pertencentes às sequências originais e erros associados às retas que passam por esses pontos;

2. desenvolveu-se uma medida *lower bounding* da distância euclidiana a partir de representações EBPLA;

3. propôs-se a representação AIPLA, que aproximou sequências numéricas por meio de um conjunto de retas IPLA de tamanhos adaptáveis às características das sequências originais; essa representação obteve menor erro de aproximação que técnicas tradicionais do estado da arte;

4. propôs-se um método eficiente de serialização e desserialização de representações AIPLA;

5. desenvolveu-se uma medida *lower bounding* da distância euclidiana a partir de representações AIPLA;

6. propôs-se adaptações de 5 estratégias de agrupamento de sequências para o contexto de consultas por similaridade em lote;

7. estudou-se 5 estratégias de agrupamento de consulta para otimizar buscas por similaridade em lote.

Os próximos capítulos desta dissertação estão dispostos da seguinte maneira: no Capítulo 2 são apresentados os conceitos essenciais para entendimento, tais como, similaridade entre sequências, estruturas de indexação, tipos de buscas por similaridade, representações reduzidas e medidas *lower bounding*. No Capítulo 3 são referenciados os trabalhos que motivaram a elaboração deste trabalho como, por exemplo, aquele que descreve a representação IPLA. Nos Capítulos 4 e 5 são descritas as novas representações EBPLA e AIPLA, respectivamente, juntamente com os algoritmos para sua extração e para computação de suas medidas *lower bounding*. No Capítulo 6 são descritos buscas por similaridade em lote e as estratégias para agrupamento de consultas. Finalmente, no Capítulos 7 são apresentados os experimentos para estudo das novas técnicas e, no Capítulo 8 são feitas as conclusões sobre esta dissertação.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica necessária para compreensão do restante desta dissertação. Na Seção 2.1 são descritos a similaridade entre sequências numéricas, o conceito de métrica e a distância *Minkowski*. Posteriormente, na Seção 2.2, são apresentadas as estruturas tradicionais de indexação e recuperação de sequências numéricas usadas para armazenamento em memória secundária. Na Seção 2.3 são apresentados dois tipos de consulta por similaridade: consulta por abrangência e consulta pelos k-vizinhos mais próximos. Na Seção 2.4 é definido o conceito de medidas *lower bounding* e como podem ser utilizadas para otimizar consultas por similaridade. Na Seção 2.5, são descritas técnicas de redução de dimensionalidade por meio de representações aproximadas das sequências numéricas. Por último, serão feitas considerações relevantes para esta dissertação na Seção 2.6.

2.1 Similaridade entre Sequências

A computação da similaridade entre sequências numéricas é fundamental nas etapas de indexação e recuperação. Na etapa de indexação, medidas de similaridades são utilizadas para preparar e armazenar em memória secundária grupos de sequências semelhantes em um índice. Na etapa de recuperação, esse índice é consultado para encontrar rapidamente um conjunto de sequências similares à consulta. Portanto, é importante que o custo da computação de medidas de similaridade seja computacionalmente baixo para que os tempos de indexação e de recuperação sejam razoáveis.

Existem dois tipos de medidas: aquelas que computam a similaridade e aquelas que computam a dissimilaridade entre sequências. Medidas de similaridade calculam a proximidade entre duas sequências, ou seja, um valor alto de similaridade significa que as sequências são muito próximas. De maneira contrária, medidas de dissimilaridade calculam a distância entre duas sequências. Nesse caso, a interpretação de um valor alto de dissimilaridade entre duas sequências é que elas são muito distantes. Tanto medidas de similaridade quanto de dissimilaridade podem ser utilizadas para indexação e recuperação

de sequências numéricas. Porém, para medidas de dissimilaridade, quanto menor for o valor retornado, maior é a similaridade entre as sequências.

Além disso, a similaridade, ou dissimilaridade, entre sequências numéricas também pode ser classificada de acordo com a maneira com que elas são comparadas, podendo ser: por correspondência inteira, em inglês, *whole matching* (AGRAWAL; FALOUTSOS; SWAMI, 1993); ou por correspondência de subsequência, em inglês, *subsequence matching* (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994; MOON; WHANG; HAN, 2002; HAN et al., 2011). Na comparação por correspondência inteira, as sequências possuem o mesmo tamanho e todos os seus valores são utilizados para computação da similaridade. Na comparação por correspondência de subsequência, as sequências podem ter tamanhos diferentes. Nesse caso, comparam-se sequências menores com segmentos de sequências maiores quando necessário.

Nas etapas de indexação e de recuperação as abordagens clássicas da literatura exploram medidas de similaridade ou de dissimilaridade computáveis em espaços métricos para acelerar e para garantir a exatidão das consultas. Para que uma medida seja considerada métrica ela deve satisfazer as seguintes propriedades:

(1) $dist(Q, S) = 0 \iff Q = S$ — identidade

(2) $dist(Q, S) = dist(S, Q)$ — simetria

(3) $dist(Q, S_2) \leq dist(Q, S_1) + dist(S_1, S_2)$ — desigualdade triangular

A propriedade de identidade (1) assegura que, dadas duas sequências diferentes, Q e S, não é possível que $dist(Q, S) > dist(Q, Q)$, do contrário, sequências exatamente iguais às consultas poderiam não ser retornadas como resultados de consultas por similaridade. A propriedade de simetria (2) garante a relação de ordem entre as sequências, ou seja, as sequências mais similares à consulta podem ser retornadas ordenadamente conforme suas distâncias à consulta. Além disso, a propriedade (2) permite reduzir a quantidade de computações de distância e de armazenamento pois é necessário calcular apenas $dist(Q, S)$ ou $dist(S, Q)$. A propriedade de desigualdade triangular (3) garante a relação de distância das sequências em espaços métricos. Dessa maneira é possível realizar inferências que possibilitam o descarte imediato de sequências não similares à consulta (ORCHARD, 1991).

Métricas baseadas na família de distâncias Minkowski são as mais utilizadas na literatura (AGRAWAL; FALOUTSOS; SWAMI, 1993; CHAN; FU, 1999; FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994; MOON; WHANG; LOH, 2001; MOON; WHANG; HAN, 2002). Essa formalização generaliza medidas de dissimilaridade no espaço L^p (DUNFORD; SCHWARTZ, 1958) e é definida na Equação 1, onde Q e S são sequências de tamanho n e p é a norma do espaço vetorial.

$$Minkowski(Q, S) = \left(\sum_{i=1}^{n} |Q_i - S_i|^p \right)^{1/p} \qquad \text{para } p \geq 1 \tag{1}$$

Dentro da família de distâncias Minkowski as mais conhecidas são a distância Manhattan, para $p = 1$, a distância euclidiana, para $p = 2$, e a distância Chebyshev, para $p = +\infty$. Essas medidas também são classificadas como medidas *lock-steps*, ou seja, a distância entre duas sequências é obtida comparando-se as posições i de cada sequência.

Métricas do espaço L^p são bastante utilizadas devido ao seu baixo custo computacional, sendo $O(n)$, onde n é o tamanho das sequências. Entretanto, as sequências devem ter o mesmo tamanho e, caso as sequências sejam ruidosas ou contenham desalinhamentos, esse tipo de medida pode aumentar consideravelmente a distância entre sequências similares (FU et al., 2008a).

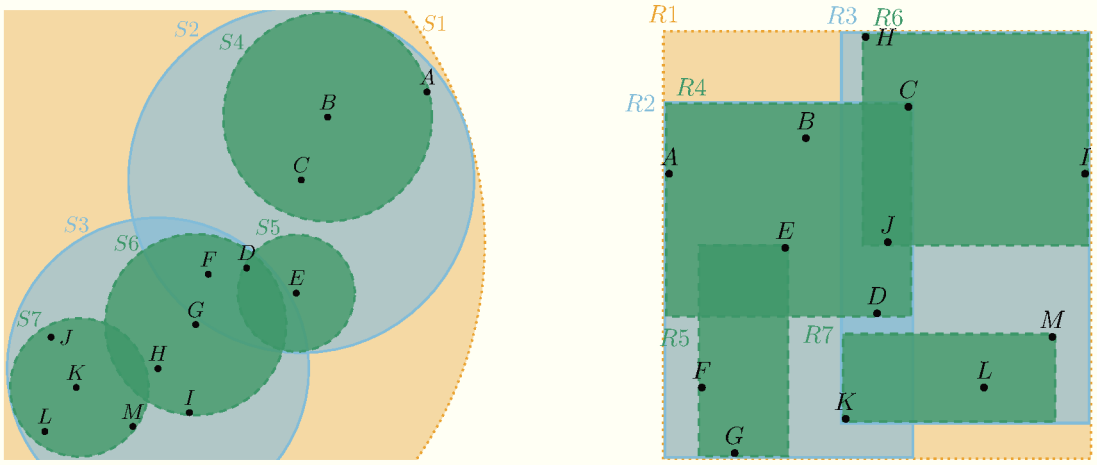
Uma outra classe de medidas, conhecidas como elásticas, também podem ser utilizadas para computar a similaridade entre sequências (KEOGH; RATANAMAHATANA, 2005a; YANKOV et al., 2007; FU et al., 2008a). Essas medidas aplicam transformações em sequências a fim de encontrar um conjunto de mapeamentos de índices que reduzam a distância entre sequências. Por isso, esse tipo de medida é mais tolerante a desalinhamentos e não requer que duas sequências tenham o mesmo comprimento, porém não são métricas pois não garantem todas as propriedades dos espaços métricos.

2.2 Estruturas de Indexação

Estruturas de indexação organizam sequências numéricas em memória secundária para facilitar e otimizar o processo de recuperação por meio de consultas por similaridade. Essas estruturas são parte fundamental de métodos de acessos multidimensionais e métricos e, geralmente, possuem formato de árvore ou de tabela *hash*. Exemplos de estruturas em árvores são: as *R-Trees* (GUTTMAN, 1984) e as *M-Trees* (CIACCIA; PATELLA; ZEZULA, 1997). Um exemplo de estrutura em formato de *hash* é a *Locality Sensitive Hashing* (LSH) (GIONIS; INDYK; MOTWANI, 1999).

R-Trees são estruturas multidimensionais que agrupam sequências similares em *containers* no formato de retângulos n-dimensionais ou *Minimum Bounding Rectangle* (MBR). Nesse caso, todas as sequências precisam ter o mesmo tamanho. *M-Trees* são estruturas métricas que utilizam bolas como *container* para delimitar grupos de sequências e, diferentemente das *R-Trees*, essas estruturas conseguem indexar qualquer tipo de dado contanto que representações nesse espaço sejam comparáveis por uma métrica. A Figura 1 ilustra espaços indexados por *R-Trees* e *M-Trees*.

Tanto *R-Trees* quanto *M-Trees* possuem dois tipos de nós: nós-diretórios, que contêm objetos do tipo diretório, e nós-folhas, que guardam objetos do tipo folha. Cada objeto-diretório possui um identificador, um *container* e uma referência para um nó-filho. No caso de *R-Trees*, os *containers* são retângulos n-dimensionais e, no caso de *M-Trees*, os *containers* são sequências indexadas juntamente com o raio das esferas n-dimensionais correspondentes. Adicionalmente, *M-Trees* também precomputam a distância para objetos pais a fim



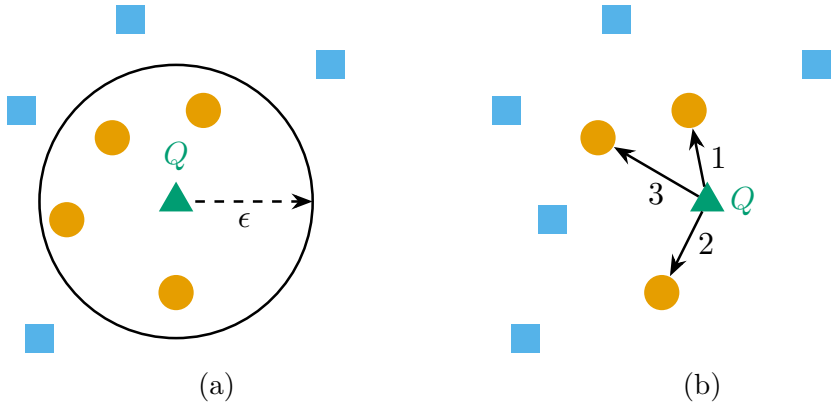
(a) (b)

Figura 1 – Agrupamento de sequências similares utilizando diferentes *containers*. Na Figura 1a é ilustrado o espaço indexado pela estrutura *M-Tree* e, na Figura 1b, o espaço indexado pela estrutura *R-Tree*. Nota-se que os *containers* mais abrangentes, amarelos, contêm todo o espaço indexado até o momento. Os *containers* menos abrangentes, azuis e verdes, são especificações do espaço indexado e são acomodados nos níveis mais baixos das estruturas de indexação.

de acelerar consultas por similaridade pela utilização da propriedade de desigualdade triangular. Cada objeto-folha contém um identificador; uma sequência, que funciona como chave adicionada por uma operação de inserção; e um valor associado à chave.

Dada uma sequência numérica, o procedimento para inserção em *R-Trees* ou *M-Trees* é descrito da seguinte maneira: caso o nó-raiz seja um nó-diretório, procure pelo objeto que contenha o melhor *container*, expanda o *container* se a sequência não estiver totalmente dentro e prossiga para o próximo nó-filho armazenado nesse objeto. Esta etapa é realizada recursivamente até que se encontre um nó-folha. Quando um nó-folha é encontrado, adiciona-se um novo objeto-folha contendo a nova sequência.

Como essas estruturas foram criadas para persistência em dispositivos de armazenamento secundário que utilizam páginas de disco de tamanho fixo, durante a operação de inserção, em caso de *overflow* de nó uma política de *split* é aplicada para redistribuir os seus objetos. Se esse nó for do tipo folha, cria-se um novo nó-folha e aplica-se a política de *split* para redistribuir os objetos entre esses dois nós. Além disso, criam-se dois novos *containers* para cada um dos nós, de modo que seus objetos estejam totalmente dentro desses *containers*. Caso o nó cheio for do tipo diretório, a redistribuição de seus objetos acontece da mesma maneira, porém, o novo nó é do tipo diretório e novos *containers* são criados a partir de outros *containers*. Adicionalmente, *M-Trees* também precisam atualizar a distância para os pais de objetos modificados. Caso o nó cheio for o nó-raiz, faz-se a redistribuição dos objetos e aumenta-se um nível da árvore a partir de um novo nó-raiz.



(a) (b)

Figura 2 – Na Figura 2a, dada a sequência de consulta Q, triângulo verde, realiza-se uma consulta por abrangência com raio ϵ recuperando-se as sequências com distância para Q menor ou igual que ϵ, bolas amarelas, e descartam-se aquelas com distância maior que ϵ, quadrados azuis. Na Figura 2b, dada a mesma consulta, realiza-se uma consulta pelos k vizinhos mais próximos, com $k = 3$, recuperando-se os k vizinhos mais próximos de Q, bolas amarelas, em ordem de proximidade e descartando-se as sequências restantes, quadrados azuis.

2.3 Consultas por Similaridade

A recuperação de sequências numéricas é realizada por meio de consultas por similaridade. Nesse tipo de consulta, uma sequência de exemplo é usada como parâmetro de consulta e as sequências mais próximas no índice são retornadas como resultado. Para isso, utilizam-se estruturas de indexação, como aquelas apresentadas na Seção 2.2, para otimizar o processo de busca.

Dois tipos de consulta por similaridade são comumente utilizados: consulta por abrangência, do inglês, *range query*, e consulta pelos k vizinhos mais próximos, do inglês k *nearest neighbours*. Em consultas por abrangência, dada uma sequência numérica Q e um raio de abrangência ϵ, encontram-se todas as sequências no índice cuja distância para Q seja no máximo ϵ. Em consultas pelos k vizinhos mais próximos, dada uma sequência numérica Q e um valor inteiro k, buscam-se as k sequências no índice mais próximas de Q. A Figura 2 ilustra esses dois tipos de consulta por similaridade.

O Algoritmo 1 descreve o processo para realizar consultas por abrangência. Primeiramente, cria-se um *container* que abrange a área dos possíveis resultados e que tem como centro o ponto representado pela sequência Q. No caso de *R-Trees*, cada lado do *container* tem largura igual a 2ϵ e, no caso de *M-Trees*, ϵ é usado diretamente como raio do *container*. Posteriormente, o algoritmo processa o nó-raiz e percorre todas as ramificações subjacentes cujos *containers* possuam sobreposição com o *container* de consulta.[1] Esse processo é realizado recursivamente até que todos os nós-folhas sejam alcançados. Quando

[1] De fato, não é necessário a criação de *containers* de consulta. Um otimização usa diretamente as sequências de consulta juntamente com seus raios de abrangência para determinar se há sobreposição com algum dos *containers* armazenados nas estruturas de indexação.

algum nó-folha é alcançado, as sequências candidatas C_i pertencentes ao seus objetos são avaliados e se $dist(Q, C_i) \leq \epsilon$, então C_i é adicionado ao resultado da consulta.

Algoritmo 1 Consulta por abrangência

Entrada: Sequência de consulta Q, raio de abrangência ϵ e nó $raiz$
Saída: Lista S contendo as sequências com distância para Q menor ou igual que ϵ

```
1: S ← {}
2: P ← {}                                              ▷ Inicializa pilha P
3: c_Q ← CRIA_CONTAINER(Q, ε, raiz)
4: INSERE_PILHA(P, raiz)
5: enquanto PILHA_VAZIA(P) é falso faça
6:     nó ← REMOVE_PILHA(P)
7:     se nó é diretório então
8:         para todo obj ∈ OBJETOS(nó) tal que CONTAINER(obj) sobrepõe c_Q faça
9:             INSERE_PILHA(P, FILHO(obj))
10:    senão se nó é folha então
11:        para todo obj ∈ OBJETOS(nó) tal que dist(Q, SEQUÊNCIA(obj)) ≤ ε faça
12:            insira SEQUÊNCIA(obj) em S
13: retorna S
```

O Algoritmo 2 descreve o processo, descrito em (ROUSSOPOULOS; KELLEY; VINCENT, 1995), para realizar buscas pelos k vizinhos mais próximos. Nesse algoritmo são necessárias uma fila de prioridade F, cujo elemento de menor prioridade é removido primeiro, e uma medida de dissimilaridade $mindist(Q, container)$, que calcula a menor distância entre a sequência Q e um *container*. Primeiramente, inicializa-se uma fila F e insere-se o nó-raiz da estrutura de indexação. Enquanto existirem elementos em F e a quantidade de vizinhos mais próximos encontrados for menor que k, o algoritmo remove um elemento de F e o processa de acordo com o seu tipo. Se o elemento é um nó-diretório, então para cada objeto desse nó, uma tupla $\langle \text{FILHO}(objeto), mindist(Q, \text{CONTAINER}(objeto)) \rangle$ é inserida na fila, onde FILHO($objeto$) é o nó-filho e CONTAINER($objeto$) é o *container* armazenado naquele objeto. Se o elemento é um nó-folha, então para cada objeto desse nó, uma tupla $\langle \text{SEQUÊNCIA}(objeto), mindist(Q, \text{SEQUÊNCIA}(objeto)) \rangle$ é inserida na fila, onde SEQUÊNCIA($objeto$) é a sequência armazenada naquele objeto. Se o elemento é uma sequência, então ela é adicionada à lista de resultados. Esse algoritmo é correto porque a distância entre uma sequência e um *container* é sempre menor ou igual que a distância entre duas sequências. Por isso, quando uma sequência é inserida na fila de prioridade, outros nós podem ser recuperados nas próximas iterações.

Na literatura correlata, observa-se que consultas pelos k vizinhos mais próximos são mais utilizadas que buscas por abrangência. Um fator determinante é que escolher uma quantidade k de resultados parece mais natural como parâmetro de busca. Outro motivo é que o parâmetro ϵ, usado em consultas por abrangência, pode ser difícil de ser inferido dependendo de sua aplicação. Em (AGRAWAL; FALOUTSOS; SWAMI, 1993), os autores sugerem o uso da média de energia entre as sequências $\bar{E}(D)$ como parte da formulação

Algoritmo 2 Busca dos k vizinhos mais próximos

Entrada: Sequência de consulta Q, quantidade de vizinhos k e nó $raiz$
Saída: Lista S contendo, ordenadamente, as k sequências mais próximas a Q

```
1: S ← {}
2: F ← {}                                          ▷ Inicializa fila de prioridade F
3: INSERE_PRIORIDADE(F, ⟨raiz, 0⟩)
4: enquanto FILA_VAZIA(F) é falso e |S| = k faça
5:     elemento ← REMOVE_PRIORIDADE(F)
6:     se elemento é um nó-diretório então
7:         para todo obj ∈ OBJETOS(elemento) faça
8:             INSERE_PRIORIDADE(F, ⟨FILHO(obj), mindist(Q, CONTAINER(obj))⟩)
9:     senão se elemento é um nó-folha então
10:        para todo obj ∈ OBJETOS(elemento) faça
11:            INSERE_PRIORIDADE(F, ⟨SEQUÊNCIA(obj), dist(Q, SEQUÊNCIA(obj))⟩)
12:    senão se elemento é uma sequência então
13:        insira elemento em S
14: retorna S
```

de ϵ, onde D é um conjunto de sequências. $\bar{E}(D)$ é descrito na Equação 2 onde n é a quantidade de sequências numéricas, d o tamanho de cada uma delas e D_{ij} o j-ésimo elemento da sequência D_i. Essa medida foi usada no Capítulo 7 desta dissertação.

$$\bar{E}(D) = \sum_{i=1}^{n} \sum_{j=1}^{d} D_{ij}^2 / n \tag{2}$$

2.4 Medidas *Lower Bounding*

Medidas *lower bounding* são computacionalmente mais baratas que medidas de similaridade e servem como seus limitantes inferiores. Em (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994), os autores demonstraram que se $lb(Q, S) \leq dist(Q, S)$, onde Q e S são sequências numéricas, então as consultas por similaridade permanecem exatas. Ou seja, não existe a ocorrência de falsos-negativos nos resultados de consultas por similaridade que utilizam a medida *lower bounding*. Posteriormente, os falsos-positivos podem ser descartados numa etapa de pós-processamento.

Muitos trabalhos utilizam medidas *lower bounding* criadas a partir de representações aproximadas das sequências originais para otimizar consultas por similaridade (AGRAWAL; FALOUTSOS; SWAMI, 1993; KEOGH et al., 2001a; KEOGH et al., 2001b). Na etapa de indexação, as aproximações das sequências são usadas como chave de busca e o endereço para as sequências originais são juntamente armazenadas como valor. Durante a etapa de recuperação, no caso de consultas por abrangência, como descrito no Algoritmo 3, utiliza-se a representação aproximada da sequência de consulta para recuperar um conjunto de sequências candidatas e, posteriormente, realiza-se uma etapa de pós-processamento.

Nessa próxima etapa, calculam-se as distâncias reais entre a consulta e as sequências candidatas para descartar aquelas que não fazem parte do resultado.

Algoritmo 3 Consulta por abrangência com *lower bounding*

Entrada: Sequência de consulta Q, raio de abrangência ϵ e nó $raiz$
Saída: Lista S contendo as sequências com distância para Q menor ou igual que ϵ

```
1: S ← {}
2: aproximação ← EXTRAI_APROXIMAÇÃO(Q)
3: candidatos ← BUSCA_POR_ABRANGÊNCIA(aproximação, ε, raiz)
4: para todo candidato ∈ candidatos faça
5:     Recupere a sequência C associada à candidato
6:     se dist(Q, C) ≤ ε então                    ▷ Remove falsos-positivos
7:         insira C em S
8: retorna S
```

Entretanto, o uso de medidas *lower bounding* para realizar consultas pelos k vizinhos mais próximos não é tão direto como em consultas por abrangência. Nesse caso, como descrito no Algoritmo 4, quando um nó-folha é retirado da fila de prioridade, então para cada objeto desse nó, insere-se uma tupla $\langle \text{APROX}(objeto), dist_{lb}(a_Q, \text{APROX}(objeto))\rangle$ na fila de prioridade, onde a_Q é a aproximação da consulta Q e $\text{APROX}(objeto)$ é a aproximação armazenada naquele objeto. Se o elemento retirado da fila é uma aproximação, então recupera-se a sequência original acessando a memória secundária e insere-se uma tupla $\langle C, dist_{real}(Q, C)\rangle$ na fila. O resto do algoritmo que realiza a busca pelos k vizinhos mais próximos permanece inalterado.

A desvantagem do Algoritmo 4 é que as sequências originais devem ser recuperadas sempre que uma aproximação é dada como um candidato promissor. Desse modo, realizam-se muitos acessos à memória secundária de maneira intermitente para recuperar uma única sequência por vez. Em (HAN et al., 2011), os autores postergam a recuperação das sequências candidatas para que sejam recuperados grupos inteiros de sequências similares. Além disso, eles utilizaram uma medida adicional para descartar grupos de sequências postergadas, evitando acessos desnecessários à memória secundária.

2.5 Redução de Dimensionalidade de Sequências

Cada elemento de uma sequência numérica é considerado uma dimensão. Por isso, desafios relacionados à alta dimensionalidade surgem nos processos de indexação e recuperação de sequências. Por exemplo, estruturas de indexação, tais como *R-trees* (GUTTMAN, 1984; BECKMANN et al., 1990; BERCHTOLD; KEIM; KRIEGEL, 2001), deterioram-se rapidamente quando registros possuem alta dimensionalidade (OTTERMAN, 1992).

Nesse cenário, o uso de algoritmos de redução de dimensionalidade torna-se necessário para obter representações reduzidas e possibilitar indexação de sequências em estruturas tradicionais. O uso de representações reduzidas proporciona a diminuição do espaço para

Algoritmo 4 Busca dos k vizinhos mais próximos com *lower bounding*

Entrada: Sequência de consulta Q, quantidade de vizinhos k e nó $raiz$
Saída: Lista S contendo, ordenadamente, as k sequências mais próximas a Q

1: $S \leftarrow \{\}$
2: $F \leftarrow \{\}$ ▷ Inicializa fila de prioridade F
3: $a_Q \leftarrow$ EXTRAI_APROXIMAÇÃO(Q)
4: INSERE_PRIORIDADE$(F, \langle raiz, 0\rangle)$
5: **enquanto** FILA_VAZIA(F) é falso **e** $|S| = k$ **faça**
6: $elemento \leftarrow$ REMOVE_PRIORIDADE(F)
7: **se** $elemento$ é um nó-diretório **então**
8: **para todo** $obj \in$ OBJETOS$(elemento)$ **faça**
9: INSERE_PRIORIDADE$(F, \langle$FILHO$(obj), mindist(Q,$ CONTAINER$(obj))\rangle)$
10: **senão se** $elemento$ é um nó-folha **então**
11: **para todo** $obj \in$ OBJETOS$(elemento)$ **faça**
12: INSERE_PRIORIDADE$(F, \langle$APROX$(obj), dist_{lb}(a_Q,$ APROX$(obj))\rangle)$
13: **senão se** $elemento$ é uma aproximação **então**
14: Recupere da memória secundária a sequência candidata C associada à $elemento$
15: INSERE_PRIORIDADE$(F, \langle C, dist_{real}(Q, C)\rangle)$
16: **senão se** $elemento$ é uma sequência **então**
17: insira $elemento$ em S
18: **retorna** S

armazenamento do índice. Além disso, essas representações são utilizadas na criação de medidas *lower bounding* para acelerar a computação de medidas de similaridade.

Uma opção para redução de dimensionalidade é o emprego de algoritmos de segmentação (KEOGH et al., 2001c). Esses algoritmos encontram particionamentos representativos das sequências que, posteriormente, são aproximados de alguma maneira. Dada uma sequência S, primeiramente, o algoritmo de segmentação encontra um conjunto de segmentos X, onde cada elemento X_i é uma partição contida em S. Posteriormente, aproxima-se cada segmento extraindo-se suas características principais. Por exemplo, representações *Piecewise Linear Approximation* (PLA) aproximam cada segmento X_i por uma reta, dessa maneira é possível manter a estrutura da sequência de modo que possibilite analisar as tendências em cada segmento.

Existem duas categorias de algoritmos de redução de dimensionalidade: algoritmos não adaptativos (KEOGH et al., 2001a; COOLEY; LEWIS; WELCH, 1969; MEYER, 1995) e algoritmos adaptativos (KEOGH et al., 2001b; KEOGH et al., 2001c; CHUNG et al., 2001). Algoritmos não adaptativos são aqueles que aproximam sequências utilizando segmentos de mesmo tamanho. De maneira oposta, algoritmos adaptativos utilizam segmentos de tamanhos variáveis, possibilitando maior granularidade na aproximação das partes mais importantes das sequências.

A seguir, são detalhados algoritmos que encontram representações com dimensionalidade reduzida para aproximar sequências. Primeiramente são descritos os algoritmos referentes às representações *Piecewise Aggregate Approximation* (PAA), *Discrete Fourier Transform*

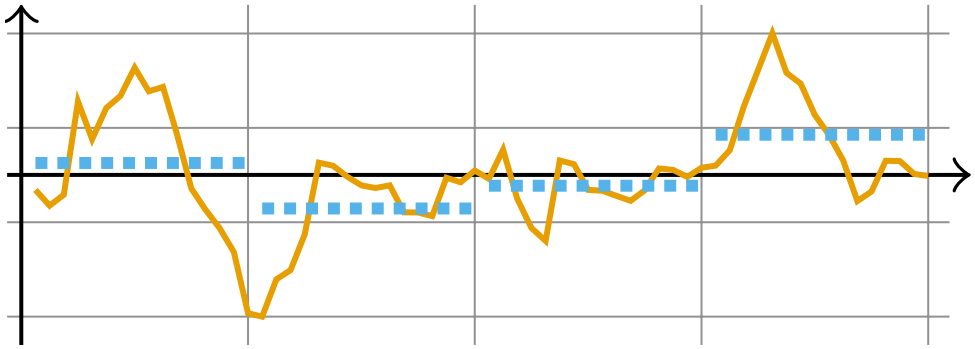


Figura 3 – Aproximação da sequência numérica, linha contínua laranjada, por meio da representação PAA, linha tracejada azul. Nesse exemplo uma sequência de tamanho 64 foi representada por apenas 4 médias de segmentos de tamanhos iguais.

(DFT) e *Discrete Wavelet Transform* (DWT), que pertencem à classe de algoritmos não adaptativos. Posteriormente, são descritos os algoritmos adaptativos para extração da representações *Adaptive Piecewise Constant Approximation* (APCA), PLA e *Perceptually Important Points* (PIP).

2.5.1 *Piecewise Aggregate Approximation*

A representação PAA (KEOGH et al., 2001a) representa cada sequência pelas médias de segmentos de mesmo tamanho, como ilustrado na Figura 3, e, por isso, o algoritmo que extrai as representações é não adaptativo. Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$, o algoritmo encontra $X = \langle x_1, x_2, \ldots, x_m \rangle$, onde x_i representa a média dos valores de cada segmento x_i. A representação PAA de uma sequência S é definida pela Equação 3.

$$x_i = \frac{m}{n} \sum_{j=\frac{n}{m}(i-1)+1}^{\frac{n}{m}i} S_j, \qquad \text{para } 1 \leq i \leq m \tag{3}$$

Um algoritmo para encontrar a representação PAA de uma sequência S utiliza uma janela disjunta (sem sobreposição) de tamanho $\frac{n}{m}$, tal que n seja múltiplo de m, e, a cada iteração, calcula a média dos elementos dentro da janela. Devido à sua complexidade computacional ser $O(n)$, onde n é o tamanho da sequência S, a representação PAA tem sido utilizada na literatura (ZHANG; GLASS, 2011; DING et al., 2008; GUO; LI; PAN, 2010).

Em (KEOGH et al., 2001a), os autores utilizaram a distância euclidiana entre as médias de duas representações PAA como medida *lower bounding*. Para garantir que $lb_{PAA}(X, Y) \leq dist(S, Q)$, onde X e Y são representações PAA contendo m coeficientes das sequências S e Q de tamanho n, introduziu-se o fator multiplicativo $\sqrt{\frac{n}{m}}$. A medida *lower bounding* lb_{PAA} é apresentada na Equação 4, onde x_i e y_i são as médias contidas nas representações PAA X e Y, respectivamente.

$$lb_{PAA}(X,Y) = \sqrt{\frac{n}{m}}\left(\sum_{i=1}^{m}(x_i - y_i)^2\right)^{1/2} \quad (4)$$

2.5.2 *Discrete Fourier Transform*

A DFT (COOLEY; LEWIS; WELCH, 1969) extrai informações referentes à energia de sequências no domínio do tempo, mapeando-as para o domínio das frequências. Para isso, analisa-se cada sequência como uma combinação de sinais com deslocamento, periodicidade e intensidade diferentes. Assim, a DFT agrupa essas três informações organizando-as entre intervalos das frequências.

Dada a sequência $S = \langle s_1, s_2, ..., s_n \rangle$ a DFT encontra os coeficientes não redundantes $Z = \langle z_1, z_2, ...z_{n/2+1} \rangle$, onde s_i e z_k pertencem ao conjunto dos números complexos no forma $a + b\boldsymbol{j}$ e z_k representam informações referentes aos sinais ordenados por suas frequências. A Figura 4 ilustra a representação de uma sequência utilizando apenas os m primeiros coeficientes da DFT.

As Equações 5 and 6 definem as transformações direta e inversa, usadas para encontrar os coeficientes da DFT e para reconstrução da sequência original. Nota-se que, para a transformação direta, a parte imaginária de cada sequência pode ser considerada igual a 0 e, para a transformação inversa, a parte imaginária pode ser descartada.

$$z_k = \sum_{i=1}^{n} s_i e^{-\boldsymbol{j}2\pi k\frac{i}{n}}, \qquad \text{para } 1 \leq k \leq n \quad (5)$$

$$s_i = \sqrt{n}\sum_{k=1}^{n} z_k e^{\boldsymbol{j}2\pi k\frac{i}{n}}, \qquad \text{para } 1 \leq i \leq n \quad (6)$$

Nota-se que, caso fosse empregado o método ingênuo, o custo computacional das Equações 5 e 6 seria $O(n^2)$. O algoritmo *Fast Fourier Transform* (FFT) (WELCH, 1967) otimiza essa computação utilizando a simetria da DFT para empregar uma abordagem divisão-e-conquista (CORMEN, 2009). Na etapa de divisão, as posições pares e ímpares são separadas para serem usadas em chamadas recursivas. Na etapa de conquista os resultados são agregados considerando-se os ajustes necessários. Dessa maneira é possível computar a DFT, nas duas direções, com custo computacional $O(n \log n)$.

Em (AGRAWAL; FALOUTSOS; SWAMI, 1993), os autores utilizaram a distância entre os primeiros m coeficientes DFT, tal que $m \ll n$, para criar uma medida *lower bounding* para a distância euclidiana. Sabe-se que, ao utilizar a DFT, a transformação de uma sequência para o domínio da frequência preserva a sua energia e, por isso, não é possível que a distância entre os m primeiros coeficientes seja maior que a distância real entre as sequências originais inteiras.

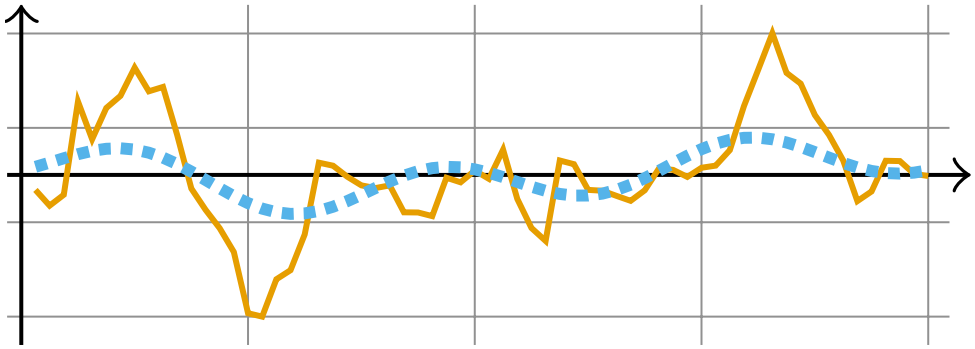


Figura 4 – Aproximação da sequência numérica, linha contínua laranjada, por meio da coeficientes DFT, linha tracejada azul. Nesse exemplo uma sequência de tamanho 64 foi representada pelos 4 primeiros coeficientes da DFT. Esses coeficientes referem-se às frequências mais baixas das sequências originais e, por isso, essa técnica possui um efeito de suavização.

2.5.3 *Discrete Wavelet Transform*

A DWT (MEYER, 1995), de maneira parecida com a DFT, extrai informações de sequências no domínio do tempo e as mapeia para o domínio das frequências. Porém, enquanto a DFT obtém uma visão global sobre as frequências da sequência, a DWT mantém uma resolução temporal e pode-se analisar as combinações de sinais por janelas de tempo. Para isso, usam-se funções *kernel* com propriedades ortogonais chamadas *wavelets*. Exemplos de *wavelets* para representar sequências numéricas são: a *Haar Wavelet* (HAAR, 1910) e a *Daubechies Wavelet* (DAUBECHIES, 1988).

Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$ a *Haar Wavelet Transform* (HWT) encontra os coeficientes $X = \langle c, x_1, x_2, \ldots, x_n \rangle$ com custo $O(n)$, onde n é múltiplo de 2, c é a média global de S e x_i são os coeficientes de detalhe de cada resolução no intervalo $[1, \log n]$. A média c é a representação mais grosseira da sequência. Entretanto, por meio das operações de soma e diferença dos coeficientes de detalhe x_i é possível encontrar médias localizadas em diferentes resoluções. Nota-se que as representações PAA e HWT possuem a mesma informação quando o tamanho das sequências é múltiplo de 2. A Figura 5 ilustra a representação de uma sequência de tamanho 64 representada pelos 4 primeiros coeficientes HWT.

Um algoritmo para computar a HWT com custo $O(n)$ recebe a sequência numérica $S = \langle s_1, s_2, \ldots, s_n \rangle$ e realiza o seguinte procedimento: para cada par de valores $\langle s_1, s_2 \rangle, \langle s_3, s_4 \rangle, \ldots, \langle s_{n-1}, s_n \rangle$, computam-se as suas médias, definido como $M_k = (s_i + s_{i+1})/2$ e as suas diferenças para M_k, definido como $D_k = (s_i - s_{i+1})/2$, onde k é a resolução das diferenças D_k, ou coeficientes de detalhe, inicialmente 1. Esse procedimento continua recursivamente utilizando as últimas médias computadas, M_{k-1}, até que $k = \log n$.

Assim como a DFT, em (CHAN; FU, 1999), os autores mostraram que a HWT também preserva a energia das sequências no domínio das frequências. Entretanto, mostrou-se que a

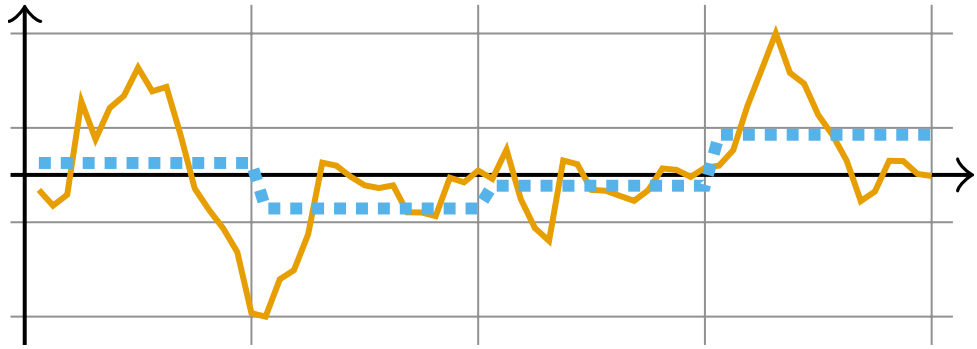


Figura 5 – Aproximação da sequência numérica, linha contínua laranjada, por meio da coeficientes HWT, linha tracejada azul. Nesse exemplo uma sequência de tamanho 64 foi representada pelos 4 primeiros coeficientes da HWT. Esses coeficientes são as médias da sequência original na resolução $r = 4$ ($m = n/2^r$), que são exatamente os mesmos coeficientes encontrados pela representação PAA na Seção 2.5.1.

energia total no domínio do tempo é $\sqrt{2}$ vezes a energia no domínio das frequências. Dessa maneira, criou-se uma medida *lower bounding* utilizando os m primeiros coeficientes HWT, como definido na Equação 7, para indexação e recuperação das sequências numéricas. Os autores também utilizaram a *Daubechies Wavelet Transform*, porém a HWT obteve melhores resultados.

$$lb_{HWT}(X, Y) = \sqrt{2}\left(\sum_{i=1}^{m}(x_i - y_i)^2\right)^{1/2} \tag{7}$$

2.5.4 *Adaptive Piecewise Constant Approximation*

A representação APCA é a versão adaptativa da representação PAA. O algoritmo para extração de representações APCA também representa a sequência S pelas médias de seus segmentos, porém, os segmentos de uma mesma sequência podem ter tamanhos diferentes, como ilustrado na Figura 6. Dada uma sequência S, a sua representação APCA tem o formato $X = \angle\langle x_1, t_1\rangle, \langle x_2, t_2\rangle, \ldots, \langle x_m, t_m\rangle\rangle$, onde x_i é a média do segmento de tamanho t_i. Dessa maneira é possível utilizar mais segmentos para representar partes complexas (muitos picos e vales) e poucos segmentos para representar partes mais simples. Devido à melhor distribuição de coeficientes APCA, é possível obter uma melhor aproximação e, consequentemente, manter um número maior de características das sequências originais.

Em (KEOGH et al., 2001b), os autores propuseram o Algoritmo 5 para encontrar a representação APCA de uma sequência S com custo $O(n \log n)$, onde n é o tamanho de S. Nesse algoritmo, primeiramente obtém-se o conjunto de coeficientes C de tamanho n produzidos pela HWT e, posteriormente, ordena-se C em ordem decrescente de acordo com seus valores absolutos. Em seguida, os m primeiros coeficientes de C são utilizados para construir a aproximação S_{aprox} de S. Por último, obtém-se a representação APCA observando-se os segmentos encontrados em S_{aprox}. Porém, nessa última etapa, deve-se

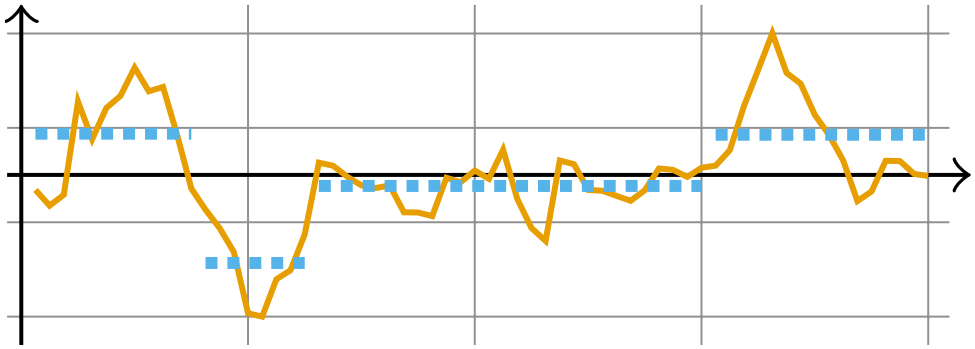


Figura 6 – Aproximação da sequência numérica, linha contínua laranjada, por meio da representação APCA, linha tracejada azul. Nesse exemplo uma sequência de tamanho 64 foi representada por 4 médias de segmentos de tamanhos diferentes. Esses coeficientes APCA são distribuídos de acordo com a variação das sequências para melhorar a sua aproximação.

corrigir erros provocados pelo truncamento do conjunto de coeficientes C. Como os valores x_i são aproximados, os seus valores exatos devem ser recalculados. Além disso, como a quantidade de segmentos encontrados M pode ser maior do que m, uma etapa de agrupamento de segmentos deve ser feita para obter apenas m segmentos.

Algoritmo 5 *Adaptive Piecewise Constant Approximation*

Entrada: Sequência S e número de segmentos m
Saída: Lista de segmentos X

1: $C \leftarrow$ HAAR_TRANSFORM(S)
2: Mantenha os m maiores valores de C, atribua 0 para o restante
3: Reconstrua um APCA X utilizando a inversa da HWT em C
4: Substitua as médias aproximadas de X por médias exatas
5: **enquanto** $|X| > m$ **faça**
6: Combine dois segmentos adjacentes que produzam o menor erro
7: **retorna** X

Para realizar consultas por similaridade a partir de representações APCA, primeiro extraem-se as representações APCA, conforme o Algoritmo 5, de cada sequência para inserção na estrutura de indexação. Entretanto, durante a etapa de recuperação, deve-se utilizar um outro algoritmo para extração da representação APCA da sequência de consulta. Esse algoritmo de extração utiliza os tempos dos segmentos de uma segunda representação para computar as suas médias. Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \langle$ e uma outra representação APCA $Y = \langle \langle y_1, t_1 \rangle, \langle y_2, t_2 \rangle, \ldots, \langle y_m, t_m \rangle \rangle$, esse novo algoritmo retorna uma representação $X' = \langle \langle \bar{s}_{1 \ldots t_1}, t_1 \rangle, \langle \bar{s}_{t_1+1 \ldots t_2}, t_2 \rangle, \ldots, \langle \bar{s}_{t_{m-1}+1 \ldots t_m}, t_m \rangle \rangle$, onde $\bar{s}_{início \ldots fim}$ é a média do segmento que vai da posição *início* até a posição fim. Ou seja, para cada comparação com uma representação APCA candidata, uma nova representação é extraída da consulta conforme os alinhamentos dessa representação com custo $O(n)$. A medida

lower bounding é apresentada na Equação 8, onde $t_0 = 0$ e m é o número de coeficientes.

$$lb_{APCA}(X', \bar{Y}) = \left(\sum_{i=1}^{m} (t_i - t_{i-1})(x'_i - y_i)^2 \right)^{1/2} \quad (8)$$

2.5.5 *Piecewise Linear Approximation*

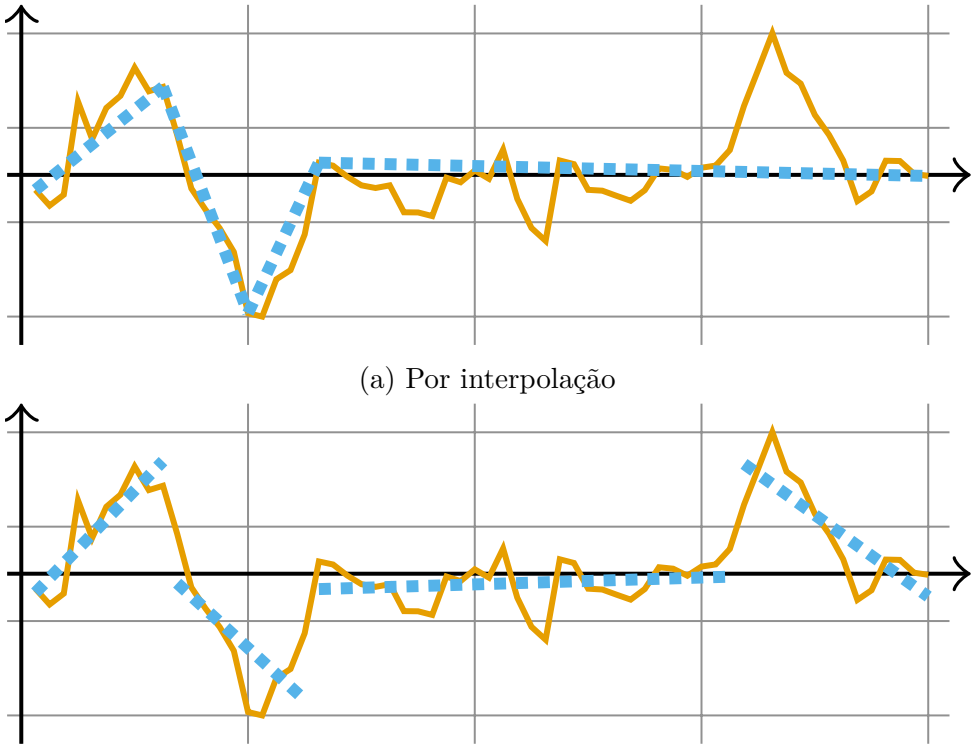
PLA é uma classe de algoritmos que representa sequências utilizando conjuntos de retas. Dada uma sequência, o objetivo desses algoritmos é escolher um conjunto de retas que minimize o erro de sua aproximação. Diferente das representações apresentadas anteriormente, onde se determina a quantidade de coeficientes a serem extraídos, algoritmos PLA também podem receber parâmetros como: o erro máximo total de aproximação e o erro máximo por segmento. Consequentemente, os algoritmos que encontram representações PLA podem retornar quantidades variáveis de coeficientes (retas) para cada sequência.

Existem duas etapas distintas para encontrar representações PLA: a etapa de segmentação da sequência e etapa de aproximação de cada segmento. Na etapa de segmentação, divide-se a sequência em segmentos para que na etapa de aproximação sejam computadas as retas mais adequadas. A maneira mais simples de aproximação interpola retas utilizando os pontos iniciais e finais de cada segmento e tem custo $O(1)$. Uma outra técnica mais sofisticada realiza regressão linear para melhor ajustar as retas. Nesse caso, o custo dessa etapa, é $O(n)$, onde n é o tamanho da sequência original. A Figura 7 ilustra esses dois tipos de aproximação de segmentos.

Várias abordagens para encontrar, adaptativamente, representações PLA de maneira subótima foram propostas (KEOGH et al., 2001c). As três abordagens mais conhecidas são: a abordagem por janela, a abordagem *top-down* e a abordagem *bottom-up*.

Na abordagem por janela, dada uma sequência S e o erro máximo por segmento ϵ, o procedimento para extração de PLA é descrito no Algoritmo 6. Nesse algoritmo, uma janela de tamanho 2 é posicionada no início de S para representar um possível segmento. Posteriormente, o erro de aproximação naquele segmento é calculado e, caso seja menor que ϵ, a janela é ampliada. Esse processo repete até que o erro de aproximação ultrapasse ϵ. Quando isso acontece, o segmento representado pela janela anterior é adicionado ao resultado e uma nova janela de tamanho 2 é posicionada no término do segmento encontrado. O algoritmo termina quando todos os segmentos de S foram extraídos. Esse procedimento é possível pois, à medida que o tamanho da janela cresce, o erro de aproximação aumenta monotonicamente. Essa abordagem tem custo $O(nl)$, onde n é o tamanho da sequência S e l é o comprimento médio dos segmentos encontrados. Além disso, esse é um algoritmo *online*, por isso, essa abordagem pode ser utilizada em contextos em que sequências são coletadas dinamicamente.

Os algoritmos *top-down* e *bottom-up* são semelhantes aos algoritmos de agrupamento hierárquicos. A abordagem *top-down* começa aproximando S com um segmento único.



(a) Por interpolação

(b) Por regressão

Figura 7 – Aproximação da sequência numérica, linha contínua laranjada, por meio de representações PLA, linha tracejada azul. Nesse exemplo, uma sequência de tamanho 64 foi representada por 4 segmentos de retas. Na Figura (a) os segmentos de reta são computados a partir do método de interpolação. Na Figura (b) utilizou-se o método de regressão. Nota-se que, na Figura (a), o início e fim de cada segmento de reta é um ponto pertencente a sequência original, diferentemente da Figura (b).

Algoritmo 6 *Piecewise Linear Approximation* — Abordagem por janela

Entrada: Sequência original S e erro máximo por segmento ϵ
Saída: Lista de segmentos X

1: $início \leftarrow 1$
2: $X \leftarrow \{\}$
3: **enquanto** segmentação não terminou **faça**
4: $\quad i \leftarrow 1$
5: $\quad$ **enquanto** erro do segmento $S_{início...início+i} < \epsilon$ **faça**
6: $\quad\quad i \leftarrow i + 1$
7: $\quad$ Adicione o segmento $S_{início...início+(i-1)}$ em X
8: $\quad início \leftarrow início + (i - 1)$
9: **retorna** X

Posteriormente, divide-se a sequência em dois segmentos de modo a proporcionar o menor erro de aproximação. Os dois segmentos são recursivamente aperfeiçoados até que todos os segmentos tenham erro de aproximação menor que ϵ. O Algoritmo 7 descreve o procedimento completo. A abordagem *top-down* tem custo computacional $O(kn^2)$, onde n é o tamanho da sequência S e k a quantidade de segmentos encontrados.

Algoritmo 7 *Piecewise Linear Approximation* — Abordagem *top-down*

Entrada: Sequência original S e erro máximo por segmento ϵ
Saída: Lista de segmentos X

Encontre posição i que separa S em dois segmentos com menor soma de erros
$X \leftarrow \{\}$
se erro de segmento $S_{1...i} > \epsilon$ **então**
    Chame recursivamente o algoritmo passando $S_{1...i}$ e adicione o resultado em X
senão
    Adicione o segmento $S_{1...i}$ em X
se erro de segmento $S_{i...|S|} > \epsilon$ **então**
    Chame recursivamente o algoritmo passando $S_{i...|S|}$ e adicione o resultado em X
senão
    Adicione o segmento $S_{i...|S|}$ em X
retorna X

A abordagem *bottom-up*, ao contrário da abordagem *top-down*, começa com todos os segmentos possíveis de comprimento igual a dois. A cada iteração o algoritmo agrupa dois segmentos adjacentes de maneira que o erro de aproximação seja o menor possível. Esse procedimento repete-se até que todos os segmentos tenham erro menor que o máximo permitido. O Algoritmo 8 descreve a abordagem *bottom-up* para encontrar a representação PLA. A abordagem *bottom-up* possui custo $O(nl)$, onde n é o tamanho da sequência S e l é o comprimento médio dos segmentos encontrados.

Algoritmo 8 *Piecewise Linear Approximation* — Abordagem *bottom-up*

Entrada: Sequência original S e erro máximo por segmento ϵ
Saída: Lista de segmentos X

$X \leftarrow$ segmentos em S de tamanho 2
$E \leftarrow$ custos para combinar dois segmentos adjacentes em X
enquanto $\min(E) < \epsilon$ **faça**
    Encontre segmentos X_i, X_{i+1} cuja combinação resulte em $\min(E)$
    Remova X_i e X_{i+1} e insira no lugar a sua combinação
    Recalcule os custos para combinar o novo segmento com seus adjacentes
retorna X

2.5.6 Identificação de *Perceptually Important Points*

Uma outra classe de algoritmos utiliza os pontos mais importantes de uma sequência, chamados de PIP, para representá-la. Os algoritmos de identificação de PIP exploram

características geométricas, como distância aos PIP previamente selecionados, para encontrar os pontos mais importantes de uma sequência. De modo similar à representação PLA, pontos importantes também podem ser utilizados para segmentar sequências e aproximá-las por meio de retas.

Dada uma sequência S de tamanho n, o número de PIP k e a medida de similaridade f, o algoritmo mais conhecido, descrito em (CHUNG et al., 2001), inicializa uma lista de PIP P contendo o primeiro e o último elemento de S. Posteriormente, $\forall p \in S$, tal que $p = (p_1, p_2)$ e $p \notin P$, onde (p_1, p_2) é um ponto com valor p_2 no tempo p_1, calcula-se a sua distância, utilizando-se f, para pontos $q = (q_1, q_2)$ e $r = (r_1, r_2)$, adjacentes em P, sendo que $q_1 < p_1 < r_1$. O elemento mais distante aos PIP correspondentes é inserido ordenadamente na lista P. Esse processo continua até que $|P| = k$, onde $|P|$ é a cardinalidade da lista P. O Algoritmo 9 tem custo $O(kn/2)$ e descreve os passos para identificação de k PIP em uma sequência S. Esse algoritmo pode ser facilmente modificado para retornar um número variável de PIP baseando-se no erro de aproximação das representações.

Algoritmo 9 Identificação de *Perceptually Important Points*

Entrada: Sequência S de tamanho n, número de PIP k e medida de similaridade f
Saída: Lista de PIP P de tamanho k ordenados por sua importância

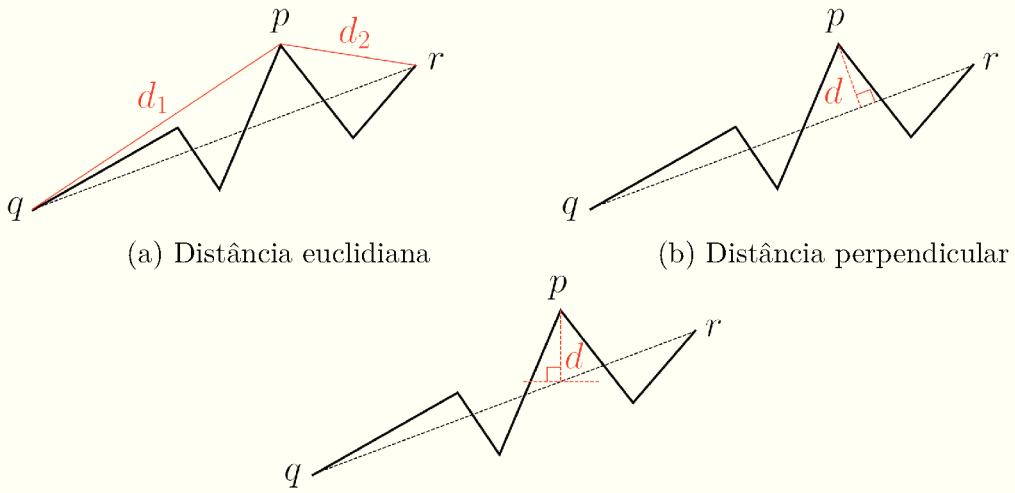
```
1: P ← {S_1, S_n}
2: enquanto |P| < k faça
3:     para todo pontos p que não estão em P faça
4:         Compute a distância entre p e pontos adjacentes em P usando f
5:     Insira p com maior distância em P
6: retorna P
```

As medidas de distância comumente utilizadas na literatura para obter a distância entre um ponto $p = (p_1, p_2)$ na sequência S e dois elementos na lista de PIP P são: a distância euclidiana de PIP, distância perpendicular e distância vertical (FU et al., 2008b). A Figura 8 ilustra as três medidas de distância na qual $q = (q_1, q_2)$ e $r = (r_1, r_2)$ são dois PIP adjacentes.

A distância euclidiana (DE) de PIP é definida como a soma das distâncias euclidianas de p para os outros dois pontos q e r, descrita na Equação 9.

$$DE(p, q, r) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} + \sqrt{(p_1 - r_1)^2 + (p_2 - r_2)^2} \tag{9}$$

A distância perpendicular (DP) é a distância de p para a reta que passa pelos pontos q



(a) Distância euclidiana

(b) Distância perpendicular

(c) Distância vertical

Figura 8 – Medidas de dissimilaridade entre o ponto p pertencente à sequência e dois pontos adjacentes q e r pertencentes à lista de PIP. Figura extraída de (FU et al., 2008b).

e r, definida pela Equação 10.

$$\begin{aligned} &inclinação(q,r) = a = \frac{r_2 - q_2}{r_1 - q_1} \\ &x_c = \frac{p_1 + ap_2 + ar_2 - a^2 r_1}{1 + a^2} \\ &y_c = ax_c - ar_1 + r_2 \\ &DP(p,q,r) = \sqrt{(p_1 - x_c)^2 + (p_2 - y_c)^2} \end{aligned} \tag{10}$$

A distância vertical (DV) também é a distância de p para a reta que passa por q e r porém, o ponto escolhido na reta é vertical a p. DV é definida pela Equação 11.

$$\begin{aligned} &inclinação(q,r) = a = \frac{r_2 - q_2}{r_1 - q_1} \\ &x_c = \frac{p_1 + ap_2 + ar_2 - a^2 r_1}{1 + a^2} \\ &y_c = q_2 + (r_2 - q_2)\frac{x_c - q_1}{r_1 - q_1} \\ &DV(p,q,r) = |y_c - p_2| \end{aligned} \tag{11}$$

2.6 Considerações

Neste capítulo apresentaram-se as principais técnicas para realizar consultas por similaridades no contexto de sequências. Uma dessas técnicas é baseada em medidas *lower bounding* computadas a partir de representações reduzidas para recuperar rapidamente sequências numéricas das estruturas de indexação. Observa-se que essas medidas são computacionalmente mais baratas e permitem filtrar um conjunto pequeno de candidatos, sem a ocorrência de falsos-negativos, para, posteriormente, computar medidas de similaridade

tais como aquelas da família de distâncias *Minkowski*. Além disso, gasta-se pouco espaço para armazenar representações reduzidas em memória secundária. Desse modo é possível aumentar a taxa de transferência de candidatos e manter mais candidatos em memória primária para processamento das consultas por similaridade.

A maioria dos algoritmos para extração de representações reduzidas que possuem medidas *lower bounding* utilizam uma quantidade fixa de coeficientes. Outros algoritmos que produzem representações com quantidades variáveis de coeficientes, tais como PLA e PIP, não possuem medidas *lower bounding*. Porém, percebe-se que algoritmos adaptativos geram representações mais próximas das sequências originais e, por isso, medidas *lower bounding* criadas a partir delas podem ser mais próximas das medidas de similaridade. Além disso, representações como PLA aproximam os segmentos das sequências por meio de retas. Dessa maneira é possível recuperar as tendências desses segmentos juntamente às sequências originais.

Neste trabalho, empregou-se algoritmos de redução de dimensionalidade adaptativos, baseados em representações PLA que produzam uma quantidade variável de coeficientes. Deseja-se encontrar representações que consigam uma boa aproximação para as sequências originais e que possam ser usadas para acelerar consultas por similaridade. Por isso, essas representações devem proporcionar medidas *lower bounding* que sejam computacionalmente baratas e que possuam alto *pruning power*. Desse modo, muitas sequências candidatas podem ser descartadas rapidamente durante o processamento das buscas.

Capítulo 3

Trabalhos Correlatos

Neste capítulo são apresentados os trabalhos que trataram o problema de segmentação de sequências a partir de representações *Piecewise Linear Approximation* (PLA) para indexação e recuperação de sequências. Primeiramente, na Seção 3.1, é apresentado o trabalho que deu origem à representação *Indexable Piecewise Linear Approximation* (IPLA) (CHEN et al., 2007), que usa um algoritmo não adaptativo para criação de uma medida *lower bounding* da distância euclidiana. Nas seções seguintes são apresentados trabalhos que segmentaram sequências porém o foco foi a indexação e recuperação das representações, descartando-se totalmente as sequências originais. Por fim, considerações sobre esses trabalhos são feitas na Seção 3.4.

3.1 *Indexable Piecewise Linear Approximation*

Todos os algoritmos para extração de representações PLA de sequências numéricas apresentados no Capítulo 2 são adaptativos e usam um erro máximo de aproximação no processo de extração das retas. Portanto, a aplicação desses algoritmos em sequências distintas podem produzir quantidades diferentes de retas. Além disso, não existe um alinhamento bem definido entre retas de representações distintas. Essas características dificultam a criação de medidas *lower bounding* e de algoritmos que realizam as operações básicas das estruturas de indexação.

Em (CHEN et al., 2007), os autores apresentaram a representação IPLA, a primeira tentativa de indexação e recuperação de sequências numéricas a partir de representações PLA. Como ilustrado na Figura 9, essa representação aproxima sequências numéricas utilizando m segmentos de reta de mesmo comprimento obtidos pelo método de regressão.

Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$ o Algoritmo 10, não adaptativo, encontra uma representação $X = \langle \langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \ldots, \langle a_m, b_m \rangle \rangle$, onde $\langle a_i, b_i \rangle$ são os coeficientes lineares da reta i. Primeiramente, divide-se cada sequência de tamanho n em m segmentos de tamanho l, onde $l = {}^{n}/_{m}$ e n é múltiplo de m. Depois, a partir desses segmentos extraem-se retas pelo método de regressão. Desse modo, todas as representações IPLA possuem a

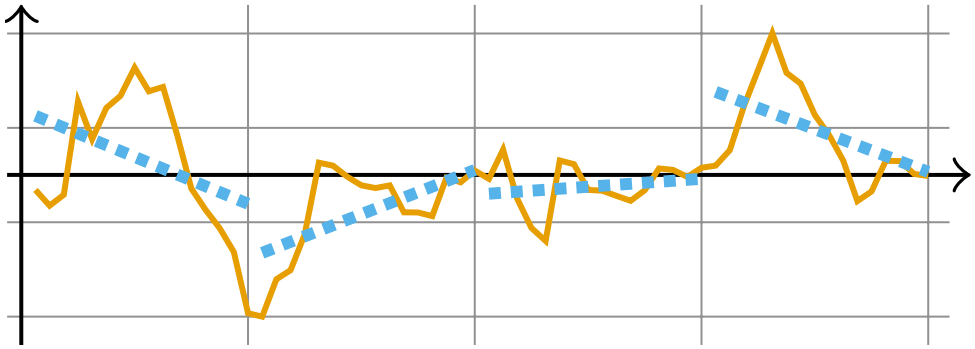


Figura 9 – Aproximação da sequência numérica, linha contínua laranjada, por meio da representação IPLA, linha tracejada azul. Nesse exemplo uma sequência de tamanho 64 foi representada por 4 segmentos de reta de tamanhos iguais encontrados a partir do método de regressão.

mesma quantidade de retas com mesmos alinhamentos, o que possibilita a criação de medidas *lower bounding*.

Algoritmo 10 *Indexable Piecewise Linear Approximation*

Entrada: Sequência S de tamanho n e quantidade de retas m, tal que n é múltiplo de m
Saída: Lista de retas X

1: $l \leftarrow n/m$
2: **para** $i = 1$ **até** n **em incrementos** l **faça**
3: $\quad início \leftarrow i; \quad fim \leftarrow i + l - 1$
4: $\quad$ insira o resultado de EXTRAIR_RETA($S_{início...fim}$) em X
5: **retorna** X

Cada reta é descrita pela função $f(t) = at + b$, onde a é o coeficiente de inclinação e b o coeficiente de interceptação no eixo das ordenadas. Para computar os coeficientes a e b, dado um segmento qualquer, os autores apresentaram as Equações 12 e 13, cada uma com custo assintótico $O(l)$, onde l é o comprimento do segmento. Sendo assim o custo total para extração da representação IPLA é $O(n)$.

$$a = \frac{12 \sum_{t=1}^{n} \left(t - \frac{(n+1)}{2}\right) S_t}{n(n+1)(n-1)} \tag{12}$$

$$b = \frac{6 \sum_{t=1}^{n} \left(t - \frac{(2n+1)}{3}\right) S_t}{n(1-n)} \tag{13}$$

Essas equações foram obtidas da seguinte maneira: sabendo-se que o erro de aproximação de um segmento S para uma reta $f(t) = at + b$ é dado pela Equação 14 e obedecendo-se as restrições das Equações 15 e 16, tem-se um sistema linear. Dessa maneira, resolvendo-se esse sistema linear é possível encontrar as equações para o cálculo dos coeficientes a e b.

$$SegError(S, a, b) = \left(\sum_{t=1}^{m} |S_t - (at + b)|^2 \right)^{1/2} \tag{14}$$

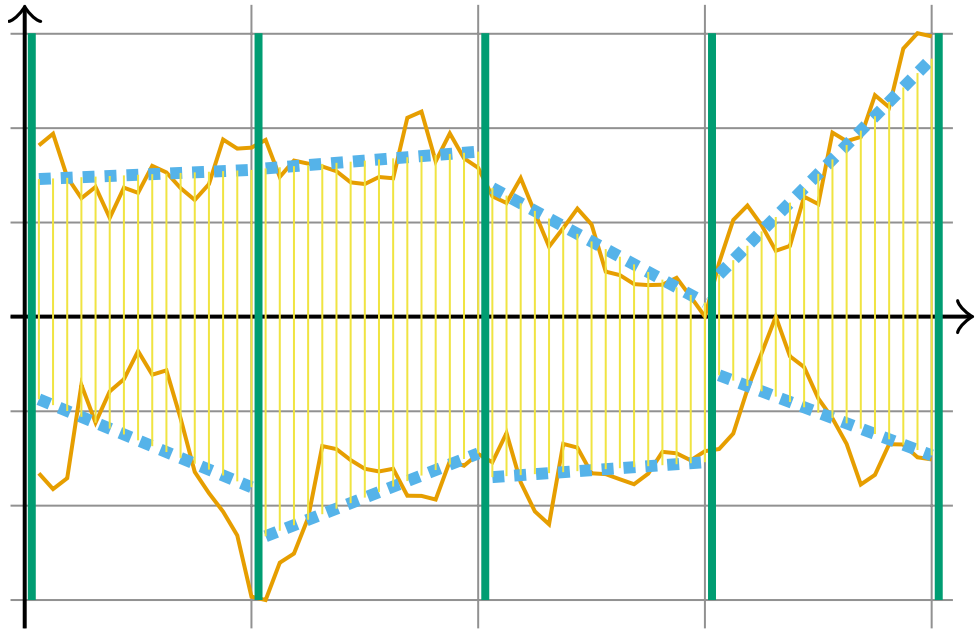


Figura 10 – Medida *lower bounding* entre representações IPLA. Primeiramente, extrai-se representações IPLA, linhas tracejadas azuis, de sequências numéricas, linhas contínuas laranjadas, usando segmentos de tamanhos iguais, delimitados pelas linhas verticais verdes. Depois computa-se a soma de todas as distâncias entre os pares de retas, representado pelas linhas verticais amarelas entre retas.

$$\frac{\partial SegError(S,a,b)}{\partial a} = 0 \tag{15}$$

$$\frac{\partial SegError(S,a,b)}{\partial b} = 0 \tag{16}$$

Nesse trabalho, como ilustrado na Figura 10, a soma dos quadrados das distâncias entre pares de retas foi utilizada como medida *lower bounding*. Dadas duas sequências S e Q, primeiramente, extraem-se as representações IPLA $X = \langle\langle a_{11}, b_{11}\rangle, \langle a_{12}, b_{12}\rangle, \ldots, \langle a_{1m}, b_{1m}\rangle\rangle$ e $Y = \langle\langle a_{21}, b_{21}\rangle, \langle a_{22}, b_{22}\rangle, \ldots, \langle a_{2m}, b_{2m}\rangle\rangle$. Posteriormente, computa-se a medida *lower bounding* $dist^2_{IPLA}$ por meio da Equação 18. O custo dessa medida é $O(m)$, onde m é a quantidade de segmentos de reta das representações. Dessa maneira, pôde-se indexar e recuperar sequências numéricas a partir de representações IPLA.

$$dist^2_{IPLA}(X,Y) = \sum_{i=1}^{m}\sum_{j=1}^{l}((a_{1i}j + b_{1i}) - (a_{2i}j + b_{2i})))^2 \tag{17}$$

$$= \sum_{i=1}^{m}\left(\begin{array}{c}\frac{l(l+1)(2l+1)}{6}(a_{1i}-a_{2i})^2 \\ +l(l+1)(a_{1i}-a_{2i})(b_{1i}-b_{2i}) + l(b_{1i}-b_{2i})^2\end{array}\right) \tag{18}$$

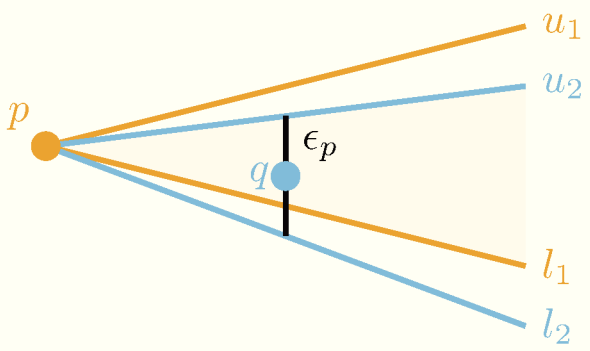


Figura 11 – *Feasible Space*. Observando-se apenas os pontos considerados até o ponto p, todas as retas entre u_1 e l_1 permitem que esses pontos tenham erro menor que ϵ_p. Quando o ponto q é incluído nesse segmento, a área para possíveis retas torna-se o espaço entre u_2 e l_1. Figura inspirada em (QI et al., 2015)

3.2 Método *Feasible Space Window*

Até o momento, os algoritmos para extração de representações PLA apresentados neste trabalho usaram como parâmetros para segmentação das sequências o erro máximo por segmento, o erro máximo total e o número total de retas.

Em (LIU; LIN; WANG, 2008), os autores apresentaram uma nova estratégia *online* chamada *Feasible Space Window* (FSW). Por ser *online*, a computação dessa representação é realizada a medida que a sequência é coletada. Essa estratégia utilizou como parâmetro o erro máximo por ponto ϵ_p para segmentar as sequências de modo que os comprimento dos segmentos de reta fossem os maiores possíveis. Para isso, apresentou-se um novo conceito chamado *Feasible Space* (FS). Esse espaço, ilustrado na Figura 11, representa todas as retas possíveis cuja distância para qualquer um dos pontos originais seja menor que ϵ_p. Dessa maneira, pôde-se aproximar as sequências utilizando uma quantidade reduzida de retas.

Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$ e o erro máximo por ponto ϵ_p, o Algoritmo 11 retorna uma representação PLA cujas retas são representadas por uma sequência de pontos. Primeiramente, o algoritmo inicializa um segmento *cur* contendo apenas s_1 e o FS com limites $[-\infty, +\infty]$. À medida que um novo ponto s_i é adicionado em *cur*, atualiza-se o FS. Para isso, utilizam-se as retas u, que passa por cur_1 e pelo ponto $(i, s_i + \epsilon_p)$, e l, que passa por cur_1 e por $(i, s_i - \epsilon_p)$. Se u possui coeficiente angular menor que o limite esquerdo então esse será o seu novo valor. No algoritmo, a função EXTRAI_A(p, q) extrai o coeficiente angular da reta que passa pelos pontos p e q. De maneira semelhante, se l possui coeficiente angular maior que o limite direito então será seu novo valor. Assim que o FS torna-se vazio, ou seja, o limite esquerdo se torna maior que o direito, o último ponto válido é adicionado ao resultado. O processo reinicia a partir do último ponto válido e continua até que toda a sequência seja processada.

Algoritmo 11 Método *Feasible Space Window*

Entrada: Sequência S e erro máximo por ponto ϵ_p
Saída: Lista de retas X

$l \leftarrow -\infty$
$u \leftarrow +\infty$
$i \leftarrow melhor \leftarrow 1$
$X \leftarrow \langle (1, S_1) \rangle$
enquanto segmentação não terminou **faça**
    $i \leftarrow i + 1$
    $u \leftarrow$ **se** $i \leq |S|$ **então** $min(u, \text{EXTRAI\_A}(X_{-1}, (i, S_i + \epsilon_p)))$ **senão** $-\infty$
    $l \leftarrow$ **se** $i \leq |S|$ **então** $max(l, \text{EXTRAI\_A}(X_{-1}, (i, S_i - \epsilon_p)))$ **senão** $+\infty$
    se $u < l$ **então**
        Adicione $(melhor, S_{melhor})$ em X
        $l \leftarrow -\infty, \quad u \leftarrow +\infty$
        $i \leftarrow melhor$
        se $melhor = |S|$ **então**
            Saia do loop
    senão se $l \leq \text{EXTRAI\_A}(X_{-1}, (i, S_i)) \leq u$ **então**
        $melhor \leftarrow i$
retorna X

3.3 Método *Feasible Coefficient Space Window*

Representações PLA aproximam segmentos de sequências utilizando funções lineares. Em outros trabalhos, encontram-se métodos que utilizam outros tipos de funções como por exemplo, funções polinomiais com grau maior que um (LEMIRE, 2007; FUCHS et al., 2010). De modo geral, qualquer tipo de função pode ser utilizada para representar segmentos de sequências numéricas.

Em (QI et al., 2015), os autores utilizaram um conjunto de funções candidatas para aproximar os segmentos de uma sequência com erro máximo por ponto ϵ_p. Essas funções são escolhidas de modo a minimizar o erro de aproximação e a quantidade total de coeficientes. Por exemplo, na Figura 12a, uma sequência numérica foi aproximada por 3 funções lineares, $f(t) = at + b$, totalizando 6 coeficientes. Em contrapartida, na Figura 12b, a mesma sequência foi melhor aproximada utilizando apenas uma função linear e uma função quadrática no formato $f(t) = at^2 + bt + c$. Para isso, expandiu-se o conceito de FS e criou-se o *Feasible Coefficient Space* (FCS). Cada tipo de função possui o seu próprio FCS, por exemplo, o FCS de um tipo de função f qualquer representa um conjunto F contendo todas as funções de tipo f que aproximam um conjunto de pontos cujas distâncias verticais sejam menores que ϵ_p.

Na Seção 3.2, mostrou-se que o FS possui uma única dimensão, o coeficiente angular das retas. No caso de uma função do tipo f expressa por m coeficientes, o seu FCS possui $m - 1$ dimensões. Por exemplo, funções quadráticas são expressas usando 3 coeficientes, $f(t) = at^2 + bt + c$, por isso o FCS para esse tipo de função possui duas dimensões, sendo

(a) 3 funções lineares (b) 1 função linear e 1 função quadrática

Figura 12 – Aproximação de sequência utilizando múltiplas funções. Na Figura (a) a sequência representada por 3 funções lineares. Na Figura (b), representa-se o segmento chamado *copo* por uma função quadrática e o segmento chamado *alça* por uma função linear. Nota-se que o erro de aproximação de (b) é menor que de (a). Figura inspirada em (QI et al., 2015)

elas representadas pelos coeficientes a e b uma vez que são dependentes de t.

O algoritmo *Feasible Coefficient Space Window* (FCSW) aproxima sequências por meio da utilização de FCS. A ideia é parecida com a do algoritmo FSW, apresentado na Seção 3.2, entretanto, cada tipo de função candidata deve manter os seus próprios segmentos e FCS, separadamente. No momento em que pelo menos um segmento é aproximado por cada uma das funções candidatas, a subsequência até o último ponto válido é aproximada pela função que gastou a menor quantidade de coeficientes. Depois, reinicia-se o processo de segmentação a partir do último ponto válido e continua-se até que toda a sequência seja processada.

3.4 Considerações

Neste capítulo foram apresentadas as técnicas de segmentação de sequências que utilizaram representações do tipo PLA. O primeiro trabalho (CHEN et al., 2007), que introduziu a representação IPLA, usou uma estratégia não adaptativa para criação de uma medida *lower bounding*. Desse modo, foi possível indexar e recuperar sequências numéricas utilizando os algoritmos apresentados no Capítulo 2.

Os trabalhos restantes (LIU; LIN; WANG, 2008; QI et al., 2015) não apresentaram medidas *lower bounding* para computar a distância entre as novas representações. O objetivo desses trabalhos foi indexar e recuperar as representações diretamente, descartando-se as sequências originais. Por isso não houve a necessidade do uso de medidas *lower bounding*.

Como apresentado nesta dissertação, não existe na literatura correlata nenhuma medida *lower bounding* para representações PLA, extraída a partir de algoritmos adaptativos. Assim sendo, neste trabalho, duas novas representações chamadas *Error-Bounded Piecewise Linear Approximation* (EBPLA) e *Adaptive Indexable Piecewise Linear Approximation* (AIPLA) são apresentadas nos Capítulos 4 e 5, respectivamente. A representação EBPLA

utiliza algoritmos adaptativos para extração de retas juntamente com erros máximos por segmento para criação de medidas *lower boundings*. Entretanto, essa representação necessita do armazenamento de dois coeficientes adicionais por segmento. A representação AIPLA flexibiliza a extração de retas IPLA de modo a aproximar sequências numéricas por uma quantidade variável de retas. Diferentemente, a representação AIPLA não necessita do armazenamento de coeficientes adicionais. Desse maneira, novos algoritmos podem utilizar essas representações para indexação e recuperação de sequências numéricas.

Capítulo 4

Error-Bounded Piecewise Linear Approximation

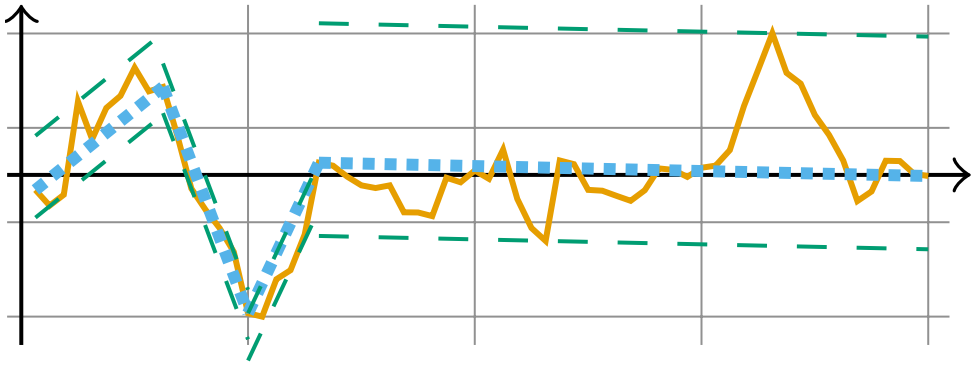
Neste capítulo é proposta a representação *Error-Bounded Piecewise Linear Approximation* (EBPLA) para indexação e recuperação de sequências numéricas. Essa representação utiliza algoritmos adaptativos, para extração de representações *Piecewise Linear Approximation* (PLA) e armazena, junto às retas, os erros verticais máximos de cada segmento aproximado. Posteriormente, é apresentado um algoritmo para extração de representações EBPLA que utiliza uma abordagem *bottom-up*. Dessa maneira é possível construir envelopes utilizando os erros de cada segmento para criação de uma nova medida *lower bounding*. Na Seção 4.2 é proposta uma medida *lower bounding* para computar a distância euclidiana entre duas representações EBPLA. Finalmente, na Seção 4.3 são destacados as qualidades e as desvantagens da abordagem proposta.

4.1 Representação EBPLA

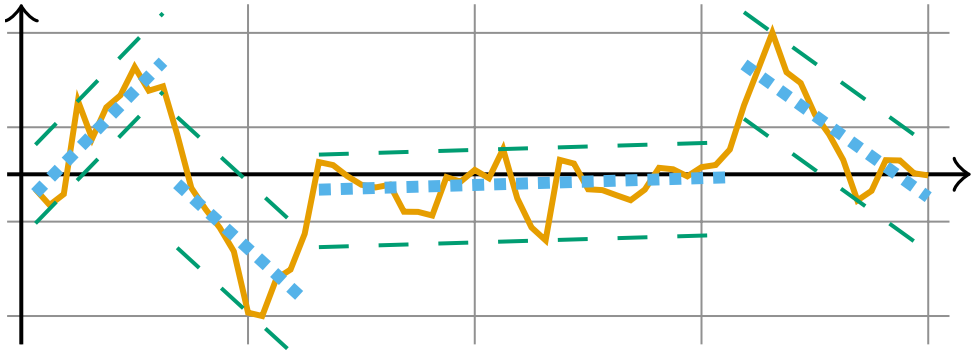
A representação EBPLA é uma extensão da representação PLA, apresentada na Seção 2.5.5, que possibilita a indexação e recuperação de sequências numéricas. Essa representação utiliza os erros verticais máximos positivos δ^+ e negativos δ^- de cada segmento para criação de envelopes a fim de estabelecer medidas *lower bounding*.

Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$, a sua representação EBPLA tem o formato $X = \langle \langle (t_1, v_1), 0, 0 \rangle, \langle (t_2, v_2), \delta_2^+, \delta_2^- \rangle, \ldots, \langle (t_m, v_m), \delta_m^+, \delta_m^- \rangle \rangle$ onde (t_i, v_i) são pontos da sequência original com amplitude v_i e tempo t_i. Os coeficientes δ_i^+ e δ_i^- representam, respectivamente, o maior valor absoluto dentre os erros verticais positivos e negativos encontrados a partir de uma reta f que aproxima o segmento $S_{t_{(i-1)} \ldots t_i}$ e passa pelos pontos (t_{i-1}, v_{i-1}) e (t_i, v_i), sendo que o erro vertical em um único ponto é definido pela diferença $s_k - f(k)$.

Nesta abordagem, os pontos que delimitam cada segmento de reta pertencem a sequência original. Dessa maneira, obtém-se as retas que aproximam cada segmento utilizando-se o



(a) Por interpolação



(b) Por regressão

Figura 13 – Aproximação da sequência numérica, linha contínua laranjada, por meio de representações EBPLA, linha tracejada azul. Nesse exemplo, uma sequência de tamanho 64 foi representada por 4 segmentos de retas. Na Figura (a) os segmentos de reta são computados a partir do método de interpolação e na Figura (b) utilizou-se o método de regressão. Nota-se que, todos os segmentos de reta são delimitados por envelopes, linhas tracejadas verdes, construídos a partir dos erros de aproximação verticais máximos positivos e negativos.

método de interpolação. Esse método foi escolhido devido o seu baixo custo computacional, $O(1)$, e pela estratégia adotada na computação da medida *lower bounding* descrita na Seção 4.2. Observa-se que uma outra estratégia poderia utilizar o método de regressão e usar o mesmo conceito de envelopes, como ilustrado na Figura 13.

Qualquer algoritmo adaptativo que extrai representações PLA pode ser facilmente modificado para extração de representações EBPLA. Nesta dissertação, modificou-se a abordagem *bottom-up*, apresentado na Seção 2.5.5, uma vez que essa abordagem produz melhores aproximações (KEOGH et al., 2001c) e a computação dos erros δ^+ e δ^- não possui custo adicional. Dada uma sequência S e um erro máximo por segmento ϵ, o Algoritmo 12 obtém a representação EBPLA de S cujos erros de aproximação de cada segmento não ultrapassam ϵ. Primeiramente, como no algoritmo original, aproxima-se S por meio de todos os segmentos possíveis de comprimento igual a dois. Adicionalmente, atribui-se o valor 0 para os erros δ^+ e δ^- de cada segmento visto que, inicialmente, a combinação de todos os segmentos é a própria sequência S. A cada iteração, o algoritmo agrupa dois segmentos adjacentes cujo erro de aproximação seja o menor. Porém, diferentemente

do algoritmo original, durante a computação do erro de aproximação de cada segmento, os erros δ^+ e δ^- também são obtidos e armazenados junto aos novos segmentos. Esse procedimento itera até que os erros de todos os segmentos sejam menores ou iguais a ϵ.

Algoritmo 12 *Error-Bounded Piecewise Linear Approximation* — Abordagem *bottom-up*

Entrada: Sequência S e erro máximo por segmento ϵ
Saída: Lista de segmentos X

1: $X \leftarrow$ segmentos em S de tamanho 2
2: $E \leftarrow$ custos para combinar dois segmentos adjacentes em X
3: $\delta^+, \delta^- \leftarrow$ erros verticais máximos positivos e negativos encontrados durante o cálculo de custos
4: **enquanto** $\min(E) < \epsilon$ **faça**
5: Encontre segmentos x_i, x_{i+1} em X e a posição j cuja combinação resulte em $\min(E)$
6: Remova x_i e x_{i+1} de X e insira no lugar a sua combinação, armazenando δ_j^+ e δ_j^- em sua tupla
7: Recalcule os custos para combinar o novo segmento com seus adjacentes, juntamente com os erros máximos e mínimos encontrados
8: **retorna** X

4.2 Distância entre representações EBPLA

Nesta abordagem, pontos que delimitam segmentos de reta são considerados importantes, caso contrário, durante o processo de segmentação outros pontos seriam escolhidos. Porém, duas representações EBPLA distintas não compartilham os mesmos alinhamentos de pontos, uma vez que o processo de segmentação é adaptativo. Por isso, quando necessário, interpolam-se novos pontos a partir das retas que passam pelos pontos originais de ambas as representações EBPLA para a computação de uma medida *lower bounding*. Além disso, envelopes são construídos a partir dos erros δ^+ e δ^- de cada segmento a fim de garantir a corretude dessa medida.

Dadas duas representações EBPLA X e Y, obtidas a partir das sequências Q e S, a distância $lb_{EBPLA}(X, Y)$ é obtida da seguinte maneira. Para cada ponto $p = (p_1, p_2)$ em X interpola-se um ponto $q' = (q'_1, q'_2)$ em Y, pela Equação 19, utilizando-se a reta no tempo correspondente. Como o ponto p pertence à Q, os erros δ^+ e δ^- associados à p são desconsiderados, entretanto, o ponto py' não existe em S, por isso esse ponto herda os erros δ^+ e δ^- correspondentes à posição p_2 da representação Y. Posteriormente, calcula-se a distância pontual $lb_{EBPLA}^{(ponto)}(p, y')$, definida na Equação 20, entre cada par de pontos $\langle x, y' \rangle$. De maneira similar, para cada ponto y em Y interpola-se um ponto q' em X no tempo correspondente e, posteriormente, calcula-se $lb_{EBPLA}^{(ponto)}(q', y)$. Observa-se que os primeiros e últimos pontos de X e Y pertencem as sequências Q e S, por isso não precisam ser

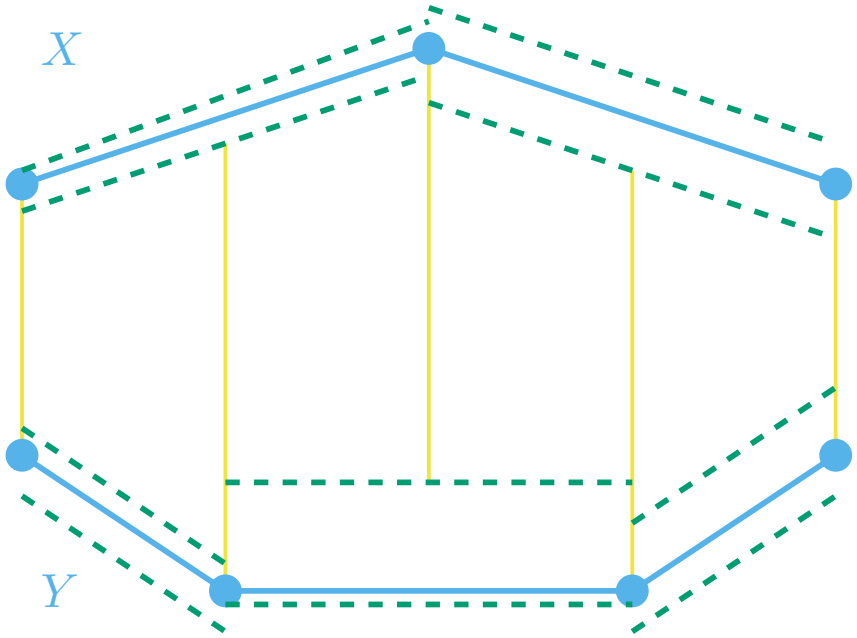


Figura 14 – Alinhamento das representações EBPLA X e Y, linhas contínuas azuis, com quantidades de segmentos 2 e 3, respectivamente. Para cada ponto pertencente à alguma das representações, indicado por círculo azul, calcula-se a distância para o ponto correspondente na outra representação que equivale ao comprimento das linhas contínuas amarelas. Caso essa outra representação não possua um ponto no mesmo alinhamento, interpola-se um novo ponto a partir da reta correspondente e os erros verticais máximos δ^+ e δ^- para construir os envelopes delimitados por linhas tracejadas verdes.

interpolados.

$$interpola(t, p, q) = \frac{q_2 - p_2}{q_1 - p_1}(t - p_1) + p_2 \tag{19}$$

$$lb^{(ponto)}_{EBPLA}(\langle (p_1, p_2), x\delta^+, x\delta^- \rangle, \langle (q_1, q_2), y\delta^+, y\delta^- \rangle) = \begin{cases} ((p_2 - x\delta^-) - (q_2 + y\delta^-))^2, & \text{se } (p_2 - x\delta^-) > (q_2 + y\delta^+) \\ ((p_2 + x\delta^+) - (q_2 - y\delta^-))^2, & \text{se } (p_2 + x\delta^+) < (q_2 - y\delta^-) \\ 0, & \text{caso contrário} \end{cases} \tag{20}$$

Essa estratégia usa apenas um pequeno subconjunto de pontos das sequências originais para calcular a similaridade entre representações EBPLA. Além disso, nos casos em que é necessário interpolar um novo ponto, compara-se esses pontos de modo que nunca seja superior à distância real. Por isso, a medida lb_{EBPLA} é *lower bounding* da distância euclidiana. A Figura 14 ilustra o alinhamento entre duas representações EBPLA e os seus envelopes construídos a partir dos erros δ^+ e δ^- de cada segmento.

Dadas as representações EBPLA X de tamanho n e Y de tamanho m, o Algoritmo 13 apresenta o procedimento para computar iterativamente $lb_{EBPLA}(X, Y)$ com custo $O(m+n)$.

Primeiramente, calcula-se a distância pontual $lb^{(ponto)}_{EBPLA}$ entre os pontos iniciais. Depois, inicializam-se dois ponteiros para marcar os primeiros pontos de cada representação. Esses pontos permitirão a interpolação de novos pontos p' e q' quando não existir um alinhamento entre as representações. Posteriormente, itera-se avançando sempre o ponteiro cujo sucessor é o mais próximo com relação ao tempo. Por exemplo, se o ponteiro da representação X avançar para o ponto X_i então o ponteiro da representação Y marca um ponto Y_j cujo tempo é menor que o tempo de X_i. Dessa maneira, utilizando-se os pontos Y_j e Y_{j+1}, interpola-se um novo ponto q' alinhado à X_i que herda os erros δ^+ e δ^- de Y_{j+1}, descarta-se os erros δ^+ e δ^- de X_i e contabiliza-se a distância pontual $lb^{(ponto)}_{EBPLA}(X_i, q')$. De modo similar, quando o ponteiro da representação Y avança, contabiliza-se a distância pontual $lb^{(ponto)}_{EBPLA}(q', Y_j)$. O algoritmo itera até que as distâncias entre todos os pontos intermediários sejam contabilizadas. Por fim, contabiliza-se a distância entre os pontos finais de cada representação, ignorando-se os erros δ^+ e δ^-.

Algoritmo 13 Distância entre representações EBPLA

Entrada: Representações EBPLA X e Y com n e m pontos, respectivamente
Saída: Distância d entre X e Y

1: $i \leftarrow j \leftarrow 1$
2: $d \leftarrow lb^{(ponto)}_{EBPLA}(X_1, Y_1)$
3: **enquanto** $(i+1) < n$ **e** $(j+1) < m$ **faça**
4: **se** TEMPO(X_{i+1}) < TEMPO(Y_{j+1}) **então**
5: $i \leftarrow i + 1$
6: $q' \leftarrow$ INTERPOLA(TEMPO$(X_i), Y_j, Y_{j+1})$ ▷ definido na Equação 19
7: Troque os valores dos erros δ^+ e δ^- de X_i por 0
8: $d \leftarrow d + lb^{(ponto)}_{EBPLA}(X_i, q')$ ▷ definido na Equação 20
9: **senão**
10: $j \leftarrow j + 1$
11: $p' \leftarrow$ INTERPOLA(TEMPO$(Y_j), X_i, X_{i+1})$
12: Troque os valores dos erros δ^+ e δ^- de Y_i por 0
13: $d \leftarrow d + lb^{(ponto)}_{EBPLA}(p', Y_j)$
14: Troque os valores dos erros δ^+ e δ^- de X_n e Y_m por 0
15: $d \leftarrow d + lb^{(ponto)}_{EBPLA}(X_n, Y_m)$
16: **retorna** $\sqrt{d}$

4.3 Considerações

Neste capítulo, apresentou-se a representação EBPLA, que aproxima sequências numéricas por meio de retas extraídas pelo método de interpolação. Qualquer algoritmo adaptativo existente que, durante o processo de segmentação, compute o erro de aproximação das retas para os segmentos correspondentes, pode ser facilmente modificado para extração de representações EBPLA. Neste trabalho, modificou-se o algoritmo *bottom-up* (KEOGH et al., 2001c), o qual gera retas com menor erro de aproximação. Dessa maneira, melho-

res tendências podem ser recuperadas em buscas por similaridade, juntamente com as sequências originais.

Para criação de uma medida *lower bounding*, armazenou-se também, os erros máximos positivos δ^+ e negativos δ^- de cada reta. Esses erros foram utilizados para construção de envelopes de sequências a fim de garantir que a distância entre pontos, pertencentes a diferentes representações EBPLA, não ultrapasse a distância real entre as sequências originais. Dessa maneira, usa-se dois números reais positivos adicionais por reta. Por isso, deve-se verificar a viabilidade do uso dessa representação em aplicações onde a quantidade de espaço para armazenamento é limitada.

Nesta proposta, um algoritmo foi proposto para computar a distância entre representações EBPLA. Esse algoritmo utiliza todos os pontos que delimitam as extremidades de cada reta e, quando necessário, interpola novos pontos utilizando o método de interpolação. Como pontos interpolados não pertencem às sequências originais, os erros máximos positivos e negativos associados aos segmentos são usados para criação de envelopes. Essa estratégia garante que a distância entre pontos interpolados nunca ultrapasse a distância real entre os pontos das sequências originais.

Capítulo 5

Adaptive Indexable Piecewise Linear Approximation

Neste capítulo, uma nova representação chamada *Adaptive Indexable Piecewise Linear Approximation* (AIPLA) é proposta para indexação e recuperação de sequências numéricas. Essa representação utiliza um algoritmo adaptativo *top-down* para extração de retas *Indexable Piecewise Linear Approximation* (IPLA), apresentado na Seção 3.1, de modo a diminuir o erro de aproximação para as sequências originais. Apesar de ser uma estratégia adaptativa, a representação AIPLA foi projetada para não necessitar de armazenamento do comprimento de cada reta e nem de coeficientes adicionais. Para isso, ela usa uma representação da árvore associada ao processo de extração de retas para obter a mesma informação gastando-se menos de $2(m-1)$ bits por representação, onde m é a quantidade de retas extraídas. Na Seção 5.3 é proposta uma medida *lower bounding* para computar a distância euclidiana entre duas representações AIPLA. Na Seção 5.4 são apresentados experimentos que sugerem empiricamente que essa medida *lower bounding* é correta nos casos avaliados. Na Seção 5.5 serão considerados os pontos positivos desta abordagem.

5.1 Representação AIPLA

A representação AIPLA flexibiliza o processo de extração de retas IPLA a fim de diminuir o erro de aproximação para as sequências originais. Para isso, um algoritmo adaptativo *top-down* subdivide recursivamente os segmentos de uma sequência em duas partes iguais até que todas as retas correspondentes tenham erro de aproximação menor ou igual a um erro máximo por segmento ϵ_{seg}. Dessa maneira, melhores tendências podem ser recuperadas, juntamente às sequências originais, em buscas por similaridade.

Dada uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$, a sua representação AIPLA tem o formato $X = \langle R, T \rangle$, onde $R = \langle \langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \ldots, \langle a_m, b_m \rangle \rangle$ é uma lista contendo m retas IPLA. Cada reta $\langle a_i, b_i \rangle$ possui comprimento múltiplo de 2 com relação ao eixo das abscissas e é formada pelos coeficientes de inclinação a_i e de interceptação no eixo das ordenadas b_i. O

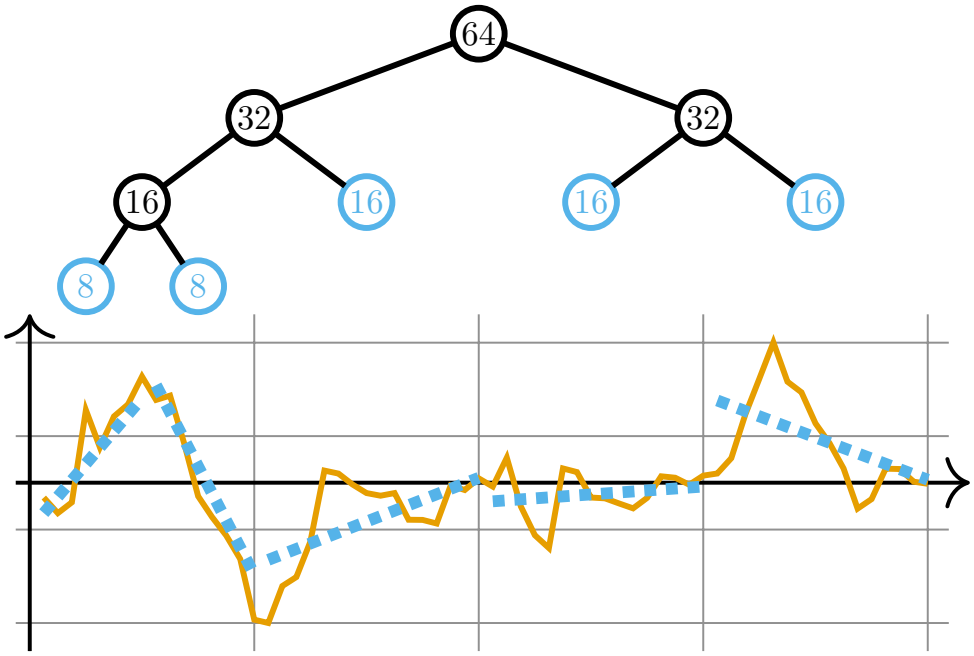


Figura 15 – Aproximação de uma sequência numérica de tamanho 64 (linha contínua laranjada) por meio da representação AIPLA (linha tracejada azul). A representação AIPLA também guarda a árvore associada ao processo de extração de segmentos para diminuir o espaço de armazenamento em memória secundária. Nesse exemplo, primeiro a sequência foi particionada em dois segmentos de tamanho 32. Depois, esses dois segmentos foram subdivididos em 4 segmentos de tamanho 16. Por último, o primeiro segmento de tamanho 16 foi novamente subdividido em duas partes de tamanho 8.

elemento T é a árvore binária associada ao processo de segmentação e pode ser serializada com menos de $2(m-1)$ bits, como descrito na Seção 5.2. Os nós-diretórios de T indicam as subdivisões sucessivas dos segmentos de S e os nós-folhas marcam os níveis em que foram encontradas as retas IPLA. Assim, obtém-se o comprimento l de cada segmento de reta observando-se o tamanho das sequências originais n e a profundidade d de cada nó-folha, tal que $l = n/2^d$. A Figura 15 ilustra a representação AIPLA de uma sequência numérica.

O Algoritmo 14 recebe uma sequência S, um erro máximo por segmento ϵ_{seg} e os índices *início* e fim que delimitam o segmento a ser processado e retorna a representação AIPLA correspondente contendo uma lista de retas IPLA R e a árvore de chamadas recursivas T. Inicialmente, o algoritmo aproxima a sequência S com uma única reta IPLA. Nesse caso, os índices *início* e fim têm valores 1 e n, e $S_{1...n}$ é o segmento processado. Se o erro de aproximação dessa reta para o segmento $S_{1...n}$ for maior que ϵ_{seg}, particiona-se $S_{1...n}$ em dois segmentos de tamanhos iguais $S_{1...k}$ e $S_{k+1...n}$. Depois, invoca-se recursivamente o algoritmo para processar cada uma das partições. Por fim, a combinação dos resultados das duas chamadas recursivas é retornada. Essa combinação é feita da seguinte maneira: as retas provenientes dos particionamentos esquerdo e direito são adicionadas em uma única lista R e, com relação às subárvores correspondentes às duas chamadas recursivas,

cria-se um novo nó-diretório T e adicionam-se as respectivas subárvores como nós-filhos da esquerda e da direita. Esse processamento recursivo é realizado até que cada uma das retas ou tenha erro de aproximação menor ou igual a ϵ_{seg}, ou tenha tamanho igual a 2. Nesse caso, retorna-se a reta juntamente com um nó-folha, que representa o término desta chamada recursiva.

Algoritmo 14 *Adaptive Indexable Piecewise Linear Approximation*

Entrada: Sequência S, erro máximo por segmento ϵ_{seg}, *início* e fim do segmento
Saída: Lista de retas R e representação de árvore T

1: $r \leftarrow$ Extraia reta IPLA do segmento $S_{início...fim}$
2: **se** $|S_{início...fim}| \leq 2$ **ou** ERRO$(r) \leq \epsilon_{seg}$ **então**
3: Adicione r em R
4: **retorna** $\langle R, null \rangle$
5: $R \leftarrow T \leftarrow \{\}$
6: $k \leftarrow (fim - início)/2$
7: $\langle R', T' \rangle \leftarrow$ AIPLA$(S, \epsilon_{seg}, início, k)$
8: $\langle R'', T'' \rangle \leftarrow$ AIPLA$(S, \epsilon_{seg}, k, fim)$
9: Extraia todas as retas de R' e R'', nessa ordem, e adicione em R
10: Adicione T' e T'' em T como sendo suas subárvores esquerda e direita
11: **retorna** $\langle R, T \rangle$

5.2 Serialização e Desserialização de Árvores

Para serializar e armazenar em memória secundária as árvores associadas ao processo de segmentação das sequências, utilizou-se uma estratégia de enumeração de árvores binárias de modo a mapeá-las para o domínio dos números inteiros positivos. Sabe-se que a quantidade de árvores binárias únicas com k nós é dada pelo número de Catalão C_k, definido recursivamente pelas funções de recorrência $C_{k+1} = \sum_{i=0}^{k} C_i C_{k-i}$ e $C_0 = 1$ (HILTON; PEDERSEN, 1991). A partir dessas funções é possível definir uma algoritmo que, dada uma árvore binária com k nós, computa um identificador único no intervalo $(0, C_k]$. Inversamente, pode-se também definir um outro algoritmo que constrói uma árvore binária com k nós a partir de seu identificador. Dessa maneira, armazena-se a árvore binária de uma representação AIPLA que possui m retas gastando-se somente um inteiro positivo de $2(m-1) - {}^3\!/_2 \log m - 1$ bits. Observa-se que o uso de $2(m-1)$ bits é possível usando o valor 1 para nós internos e 0 para nós externos, ordenados por uma travessia pré ou em-ordem da árvore.

O Algoritmo 15 computa o identificador único de uma árvore T com k nós. Para isso, usou-se a função bijetora $g(\langle n_1, id_1 \rangle, \langle n_2, id_2 \rangle) = \sum_{i=0}^{n_1-1} C_i C_{n_1+n_2-i} + (id_1 \cdot C_{n_2}) + id_2$ definida a partir das quantidades de nós n_i e dos identificadores id_i das subárvores da esquerda e da direita de T. Desse modo, utilizando-se a função g, primeiramente computam-se os identificadores das subárvores menores e, recursivamente, os agrega para obter os

identificadores das subárvores maiores. Quando os identificadores das maiores subárvores de T são agregados, obtém-se o identificador de T. O caso base desse algoritmo é aquele onde a subárvore é composta por um único nó-folha. Nesse caso retorna-se o identificador com valor 0, uma vez que existe apenas uma maneira de representar essa árvore.

Algoritmo 15 Serializa árvore

Entrada: Representação de árvore T
Saída: Quantidade de nós n e identificador id

1: **se** T é *null* **então**
2: **retorna** $\langle 0, 0 \rangle$
3: $\langle n_1, id_1 \rangle \leftarrow$ SERIALIZA_ÁRVORE(SUBÁRVORE_ESQUERDA(T))
4: $\langle n_2, id_2 \rangle \leftarrow$ SERIALIZA_ÁRVORE(SUBÁRVORE_DIREITA(T))
5: $n \leftarrow n_1 + n_2 + 1$
6: $id \leftarrow \sum_{i=0}^{n_1-1} C_i C_{n_1+n_2-i} + (id_1 \cdot C_{n_2}) + id_2$ ▷ C são números de Catalão
7: **retorna** $\langle n, id \rangle$

O Algoritmo 16 realiza a operação inversa do Algoritmo 15 para construir, recursivamente, uma árvore binária T com k nós a partir do seu identificador único. Para isso, dados a quantidade de nós k e o identificador id de uma subárvore, inicialmente T, computam-se os identificadores correspondentes às suas subárvores da esquerda, pela função $g_1^{-1}(k, id) = (id - \sum_{i=0}^{j} C_i C_{k-i-1}) \text{ div } (C_i C_{k-j})$, e da direita, pela função $g_2^{-1}(k, id) = (id - \sum_{i=0}^{j} C_i C_{k-i-1}) \bmod (C_i C_{k-j})$, onde div e mod são as operações de divisão inteira e de resto da divisão inteira. O índice j equivale à quantidade de nós da subárvore da esquerda e seu valor é escolhido de modo a minimizar a diferença $id - \sum_{i=0}^{j} C_i C_{k-i-1}$, tal que o resultado seja positivo. Posteriormente, utilizam-se os identificadores de cada subárvore com número de nós j e $k - j - 1$ para invocar novamente o algoritmo a fim de continuar o processo de construção dos níveis inferiores de T. Finalmente, retorna-se um nó-diretório T cujos filhos da esquerda e da direita são os resultados das chamadas referentes às subárvores da esquerda e da direita. Esse processo é realizado recursivamente até que $k = 0$ e $id = 0$, nesse caso, chegou-se a um nó-folha e por isso retorna-se o valor *null*.

Nota-se que não é necessário o armazenamento explícito do número de nós k para desserialização das árvores binárias. Esse valor pode ser obtido indiretamente utilizando a quantidade de retas m contidas nas representações AIPLA. Como cada árvore binária de uma representação AIPLA possui m nós-folhas e sabendo-se que cada nó é representado pelo valor *null*, então, tem-se que $k = m - 1$, a quantidade de nós internos dessa árvore.

5.3 Distância entre representações AIPLA

Na abordagem AIPLA, as distâncias entre retas de duas representações foram utilizadas para computar uma medida *lower bounding* da distância euclidiana. Embora

Algoritmo 16 Desserializa árvore

Entrada: Quantidade de nós n e identificador id
Saída: Representação de árvore T

1: **se** $n < 1$ **então**
2: **retorna** $null$
3: $T \leftarrow \{\}$
4: $i \leftarrow 0$
5: **enquanto** $id \geq C_i C_{n-i-1}$ **faça**
6: $id \leftarrow id - C_i C_{n-i-1}$
7: $i \leftarrow i + 1$
8: $T' \leftarrow$ DESERIALIZA_ÁRVORE$(i, id \text{ div } C_{n-i-1})$ ▷ div = divisão inteira
9: $T'' \leftarrow$ DESERIALIZA_ÁRVORE$(n - i - 1, id \bmod C_{n-i-1})$
10: Adicione T' e T'' em T como sendo suas subárvores esquerda e direita
11: **retorna** T

diferentes representações AIPLA possuam quantidades diferentes de retas, nota-se que elas representam sequências de mesmo tamanho e as suas retas têm comprimentos múltiplos de dois. Sendo assim, numa comparação *whole matching*, existem dois casos possíveis de alinhamentos.

No primeiro caso, alinham-se duas retas, pertencentes a representações diferentes, as quais iniciam no mesmo índice e possuem o mesmo comprimento. Nesse caso, a distância entre retas é dada pela Equação 18, apresentada na Seção 3.1, que define a distância entre retas IPLA. No segundo caso, alinham-se as retas $\langle\langle a_{1i}, b_{1i}\rangle, \langle a_{1i+1}, b_{1i+1}\rangle, \ldots, \langle a_{1i+k}, b_{1i+k}\rangle\rangle$, pertencente a uma das representações, com uma única reta $\langle a_{2j}, b_{2j}\rangle$ de outra, de modo que as retas $\langle a_{1i}, b_{1i}\rangle$ e $\langle a_{2j}, b_{2j}\rangle$ começam no mesmo índice e a soma dos comprimentos das retas $\langle\langle a_{1i}, b_{1i}\rangle, \langle a_{1i+1}, b_{1i+1}\rangle, \ldots, \langle a_{1i+k}, b_{1i+k}\rangle\rangle$ seja igual ao comprimento da reta $\langle a_{2j}, b_{2j}\rangle$. Nesse caso, divide-se a reta $\langle a_{2j}, b_{2j}\rangle$ em k partes, onde k é a quantidade de retas em $\langle\langle a_{1i}, b_{1i}\rangle, \langle a_{1i+1}, b_{1i+1}\rangle, \ldots, \langle a_{1i+k}, b_{1i+k}\rangle\rangle$ e, posteriormente, calcula-se a distância entre cada uma das partes e as retas correspondentes da outra representação. Porém, caso fossem computadas k retas IPLA, provavelmente não seriam as mesmas k partes da reta $\langle a_{2j}, b_{2j}\rangle$. Por isso, um valor múltiplo do erro máximo por segmento ϵ_{seg} é subtraído da distância desse alinhamento para garantir que a medida seja *lower bounding* da distância euclidiana, neste trabalho utilizou-se o valor $3\epsilon_{seg}$, encontrado empiricamente conforme mostrado na seção seguinte. A Figura 16 mostra o alinhamento completo entre duas representações AIPLA.

Quando ocorre o alinhamento de k retas de uma representação com apenas uma reta de outra, a comparação é feita individualmente para cada uma das suas k partes. Cada reta IPLA de comprimento l é extraída de modo que os valores de seu contradomínio sejam obtidos por domínios no intervalo $[1, l]$, como apresentado na Seção 3.1. Por isso, as $k - 1$ partes subsequentes à primeira são deslocadas para a esquerda por meio da Equação 21 para que comecem no índice de valor 1. Dessa maneira, calcula-se a distância quadrática

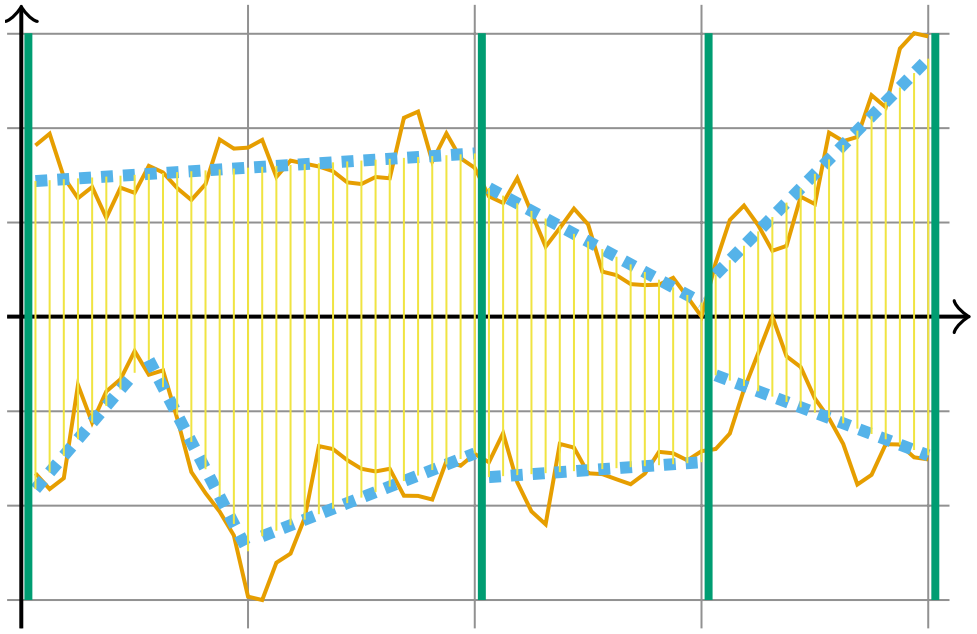


Figura 16 – Distância entre duas representações AIPLA, que são indicadas pelas linhas pontilhadas azuis, extraídas a partir de duas sequências, indicadas pelas linhas contínuas laranjadas. A distância entre as duas representações equivale ao comprimento das linhas contínuas amarelas. Nota-se que, em cada alinhamento entre as barras verdes, mais de uma reta pertencente a uma representação podem estar alinhadas com uma única reta de outra. Nesse caso, é necessário subtrair um valor múltiplo de ϵ_{seg} para que a medida seja *lower bounding*.

lb^2_{AIPLA} entre k retas de uma representação e uma única reta de outra, representadas por k partes independentes com deslocamento d_r, utilizando-se a Equação 22, onde o deslocamento da r-ésima reta é $d_r = (\sum_{j=i}^{r} l_j) - l_r$ tal que a primeira das k retas começa no índice i e possui comprimento l_j.

$$\begin{aligned} f(x+d) &= a(x+d)+b \\ &= ax + (ad+b) \end{aligned} \tag{21}$$

$$\begin{aligned} &lb^2_{AIPLA}(\langle\langle a_{1i}, b_{1i}\rangle, \langle a_{1i+1}, b_{1i+1}\rangle, \ldots, \langle a_{1i+k}, b_{1i+k}\rangle\rangle, \langle a_{2j}, b_{2j}\rangle\rangle) = \\ &= \max\left(0, \sum_{r=i}^{i+k}\sum_{t=1}^{l_r}\left[\left(a_{1r}t + b_{1r}\right) - \left(a_{2j}t + \left(a_{2j}d_r + b_{1j}\right)\right)\right]^2 - 3\epsilon_{seg}\right) \\ &= \max\left(0, \sum_{i\in I}\left(\begin{array}{c} \frac{l_r(l_r+1)(2l_r+1)}{6}(a_{1j}-a_{2r})^2 \\ + l_r(l_r+1)(a_{1j}-a_{2r})(b_{1r}-(a_{2j}d_r+b_{2j})) \\ + l_r(b_{1r}-(a_{2j}d_r+b_{2j}))^2 \end{array}\right) - 3\epsilon_{seg}\right) \end{aligned} \tag{22}$$

Dadas duas representações AIPLA $X = \langle\langle a_{11}, b_{11}\rangle, \langle a_{12}, b_{12}\rangle, \ldots, \langle a_{1m}, b_{1m}\rangle, T_1\rangle$ e $Y = \langle\langle a_{21}, b_{21}\rangle, \langle a_{22}, b_{22}\rangle, \ldots, \langle a_{2n}, b_{2n}\rangle, T_2\rangle$ dois algoritmos equivalentes foram propostos para computar uma medida *lower bounding* da distância euclidiana, sendo o primeiro *offline* e

segundo *online*. O primeiro, descrito no Algoritmo 17, recupera o comprimento l de cada segmento de reta observando a profundidade d dos nós-folhas das árvores T_1 e T_2 de cada representação, tal que $l = s/2^d$ e s é o tamanho das sequências originais. Posteriormente, as retas de X são alinhadas com as retas de Y de modo que, em cada alinhamento, a posição inicial das primeiras retas e a soma de cada conjunto de retas sejam iguais. Finalmente, a raiz quadrada da soma das distâncias quadráticas entre as retas de cada alinhamento é calculada da seguinte maneira: se o alinhamento possui uma reta para cada representação, utilize a medida *lower bounding* entre retas IPLA lb^2_{IPLA}, descrita na Seção 3.1. Caso o alinhamento possua mais de uma reta, pertencentes a uma das representações, e uma única reta, pertencente à outra, utilize a medida *lower bounding* lb^2_{AIPLA}.

Algoritmo 17 Distância entre representações AIPLA — Abordagem *offline*

Entrada: Representações AIPLA $X = \langle R_1, T_1 \rangle$ e $Y = \langle R_2, T_2 \rangle$
Saída: Distância d entre X e Y

1: $dist \leftarrow 0$
2: Recupere o comprimento de cada reta utilizando as árvores T_1 e T_2
3: $P \leftarrow$ Alinhe R_1 e R_2 de modo que cada alinhamento tenha retas R'_1 e R'_2
4: **para todo** alinhamento $p = \langle R'_1, R'_2 \rangle \in P$ **faça**
5: **se** $|R'_1| = 1$ **e** $|R'_2| = 1$ **então**
6: $dist \leftarrow dist + lb^2_{IPLA}(R'_1, R'_2)$ ▷ descrito no Capítulo 3
7: **senão se** $|R'_1| > 1$ **então**
8: $dist \leftarrow dist + lb^2_{AIPLA}(R'_1, R'_2)$ ▷ descrito na Equação 22
9: **senão**
10: $dist \leftarrow dist + lb^2_{AIPLA}(R'_2, R'_1)$
11: **retorna** $\sqrt{dist}$

O Algoritmo 18, diferentemente, não realiza um pré-processamento das listas de retas R_1 e R_2, pertencentes a cada uma das representações AIPLA, a fim de compor previamente todos os alinhamentos. Em vez disso, comparam-se pares de retas alinhadas de maneira iterativa considerando-se divisões e deslocamentos quando necessário. Primeiramente, como no algoritmo anterior, recuperam-se os comprimentos das retas utilizando-se as árvores das respectivas representações AIPLA. Adicionalmente, inicializam-se os índices $i = 1$ e $j = 1$, para marcar as próximas retas a serem comparadas em R_1 e R_2. Na próxima etapa, itera-se nas listas de retas R_1 e R_2 para calcular as distâncias entre pares de retas alinhadas da seguinte maneira: se R_{1i} e R_{2j} têm o mesmo comprimento, então adicione $lb^2_{IPLA}(R_{1i}, R_{2j})$ a distância total e incremente os índices i e j. Caso contrário, a reta com comprimento maior, pertencente a uma das representações, é comparada com as próximas retas da outra representação. A cada comparação, adiciona-se $lb^2_{IPLA}(R_{1i}, R_{2j})$ a uma variável temporária tmp, considerando-se os menores comprimentos entre retas e deslocamentos necessários, e incrementa-se o índice correspondente as retas que possuem comprimento menor. Quando não é mais possível deslocar a reta maior, adiciona-se $\max(0, tmp - 3\epsilon)$ a distância total e incrementa-se o índice correspondente a reta que

possui comprimento maior. Esse procedimento é realizado até que as todas as distâncias quadráticas entre retas alinhadas sejam contabilizadas em *dist* e, finalmente, retorna-se a distância final $\sqrt{dist}$.

Algoritmo 18 Distância entre representações AIPLA — Abordagem *online*

Entrada: Representações AIPLA $X = \langle R_1, T_1 \rangle$ e $Y = \langle R_2, T_2 \rangle$, com m e n retas
Saída: Distância *dist* entre X e Y

1: $dist \leftarrow 0$
2: $i \leftarrow j \leftarrow 1$
3: Recupere os comprimentos de cada reta utilizando as árvores T_1 e T_2
4: **enquanto** $i \leq m$ **e** $j \leq n$ **faça**
5: **se** COMPRI$(R_{1i}) =$ COMPRI(R_{2j}) **então** ▷ 1 reta com 1 reta
6: $dist \leftarrow dist + lb^2_{IPLA}$ entre R_{1i} e R_{2j}
7: $i \leftarrow i + 1$
8: $j \leftarrow j + 1$
9: **senão se** COMPRI$(R_{1i}) >$ COMPRI(R_{2j}) **então** ▷ 1 reta com muitas retas
10: $tmp \leftarrow d \leftarrow 0$
11: **enquanto** $d <$ COMPRI(R_{1i}) **faça**
12: $r_1 \leftarrow$ Desloca R_{1i} para esquerda d posições
13: $tmp \leftarrow tmp + lb^2_{IPLA}$ entre r_1 e R_{2j} considerando o comprimento de R_{2j}
14: $j \leftarrow j + 1$
15: $dist \leftarrow dist + \max(0, tmp - 3\epsilon)$
16: $i \leftarrow i + 1$
17: **senão** ▷ muitas retas com 1 reta
18: $tmp \leftarrow d \leftarrow 0$
19: **enquanto** $d <$ COMPRI(R_{2j}) **faça**
20: $r_2 \leftarrow$ Desloca R_{2j} para esquerda d posições
21: $tmp \leftarrow tmp + lb^2_{IPLA}$ entre R_{1i} e r_2 considerando o comprimento de R_{1i}
22: $i \leftarrow i + 1$
23: $dist \leftarrow dist + \max(0, tmp - 3\epsilon)$
24: $j \leftarrow j + 1$
25: **retorna** $\sqrt{dist}$

5.4 Corretude da medida *lower bounding* lb_{AIPLA}

Como apresentado na seção anterior, a medida lb_{AIPLA} utiliza as distâncias entre retas IPLA como *lower bounding* da distância euclidiana. Quando se forma um alinhamento que compara apenas uma reta IPLA de cada uma das representações AIPLA, usa-se a medida lb_{IPLA} para computar a sua distância. Quando mais de uma reta, pertencentes a uma das representações, são alinhadas com apenas uma reta, de outra, divide-se esta em partes individuais para uso da medida lb_{IPLA}. Entretanto, nesse segundo caso, subtrai-se um valor múltiplo do erro máximo por segmento ϵ_{seg} para que a medida lb_{AIPLA} não ultrapasse a distância euclidiana entre os segmentos das sequências originais correspondentes.

Nesta proposta, subtraiu-se, no máximo, o valor $3\epsilon_{seg}$ de cada alinhamento do segundo tipo. Para verificar empiricamente a corretude desta proposta, foi conduzido um experimento utilizando sequências numéricas geradas a partir de um modelo de caminhadas aleatórias. Esse modelo foi utilizado principalmente devido sua capacidade de geração de sequências com variações semelhantes àquelas comumente encontradas em diversas áreas do conhecimento. Além disso, técnicas, tais como o método das diferenças (HAMILTON, 1994), podem ser utilizadas a fim de converter sequências desse domínio para o de outros modelos.

Para realização desse experimento, 80000 sequências numéricas de tamanho 512 foram aproximadas por meio de representações AIPLA utilizando um mesmo erro máximo por segmento $\epsilon_{seg} = 8.14$. No processo de segmentação utilizou-se a distância euclidiana quadrática como medida de erro de aproximação entre uma reta e o segmento correspondente da sequência original. Dentre as 80000 sequências, um quarto foram aproximadas por representações AIPLA contendo apenas 1 reta e as sequências restantes, igualmente distribuídas, foram aproximadas por representações com 2, 3 ou 4 retas. Dessa maneira, observou-se as distâncias máximas entre representações AIPLA, $dist_{AIPLA}$, obtidas a partir de comparações entre as representações contendo apenas uma reta e representações com k retas, tal que $1 \leq k \leq 4$, totalizando 4×20000^2 cálculos de distância computados. A distância $dist_{AIPLA}$ é similar à medida *lower bounding* lb_{AIPLA}, porém, a primeira não subtrai um valor múltiplo de ϵ_{seg}. Como apresentado na Tabela 1, quando duas representações AIPLA, que contêm apenas uma reta, são comparadas, a distância aproximada nunca ultrapassa a distância real. Porém, nos casos em que uma das representações possui mais do que uma reta, a distância aproximada pode ser maior que distância real. Nesses casos, percebeu-se que quanto mais retas essa representação possui, maior é a diferença mínima entre as distâncias reais e a distâncias aproximadas. Além disso, a menor diferença mínima absoluta foi 21.3384, que, por sua vez, é menor que $3\epsilon_{seg}$.

5.5 Considerações

Neste capítulo, apresentou-se a representação AIPLA que aproxima sequências numéricas por meio de retas IPLA. Diferentemente à estratégia IPLA, utilizou-se um processo de segmentação *top-down* e adaptativo para extração de representações AIPLA. A quantidade de retas a serem utilizadas para aproximar uma sequência foi dependente da variação de seus valores e do erro máximo por segmento ϵ_{seg}, escolhido como parâmetro. Dessa maneira, pôde-se extrair melhores tendências que aquelas da representação IPLA.

Representações adaptativas, como a representação *Error-Bounded Piecewise Linear Approximation* (EBPLA), apresentada no Capítulo 4, precisam armazenar o tempo de cada ponto ou, de outra forma, os comprimentos de cada segmento aproximado. Nesta abordagem, propôs-se guardar a árvore associada ao processo de segmentação para que,

Tabela 1 – Máximas distâncias aproximadas e reais em grupos contendo representações AIPLA com k retas e suas sequências correspondentes. A primeira coluna indica o grupo, a segunda coluna apresenta as maiores distâncias, $dist_{AIPLA}$, entre as representações de cada grupo e aquelas contendo apenas uma reta, $k = 1$; a terceira mostra as maiores distâncias entre suas sequências correspondentes; e a quarta, a diferença absoluta entre as duas colunas anteriores. Percebe-se que quanto mais retas de uma representação são comparadas com uma única reta de outra, menor é a diferença absoluta, exceto no cenário onde as duas representações possuem uma única reta. Como o valor ϵ_{seg} utilizado para extração das representações foi 8.14 e a maior diferença absoluta foi 21.3384, a subtração de $3\epsilon_{seg}$ de $dist_{AIPLA}$ é suficiente para assegurar a propriedade de *lower bounding* nesse conjunto de dados.

k	$aprox$	$real$	$\lvert aprox - real \rvert$
1	0	0	0
2	98.5014	77.1630	21.3384
3	54.1265	34.6620	19.4645
4	57.2665	38.0935	19.1729

posteriormente, durante a comparação entre representações AIPLA, fosse possível recuperar o comprimento de cada segmento. A forma serializada dessa árvore ocupa apenas um inteiro positivo com menos de $2(m-1)$ bits, onde m é a quantidade de retas pertencentes a cada representação. Como, geralmente, poucas retas são utilizadas na aproximação de sequências, devido ao problema da alta dimensionalidade (OTTERMAN, 1992), essa estratégia torna-se viável.

Também foi proposta uma nova medida *lower bounding* para distância euclidiana. Para isso, utilizou-se a medida lb_{IPLA}, apresentada na Seção 3.1, entre retas ou partes de retas, e, no caso em que partes de retas são comparadas, subtraiu-se um valor múltiplo do erro máximo por segmento ϵ_{seg} de cada alinhamento. Demostrou-se empiricamente que, nos cenários considerados, $lb_{AIPLA} \leq dist$. Dessa maneira, é possível indexar representações AIPLA para acelerar o processo recuperação de sequências numéricas. Nota-se que essa medida *lower bounding* pode ser melhorada estudando-se a natureza do problema.

Capítulo 6

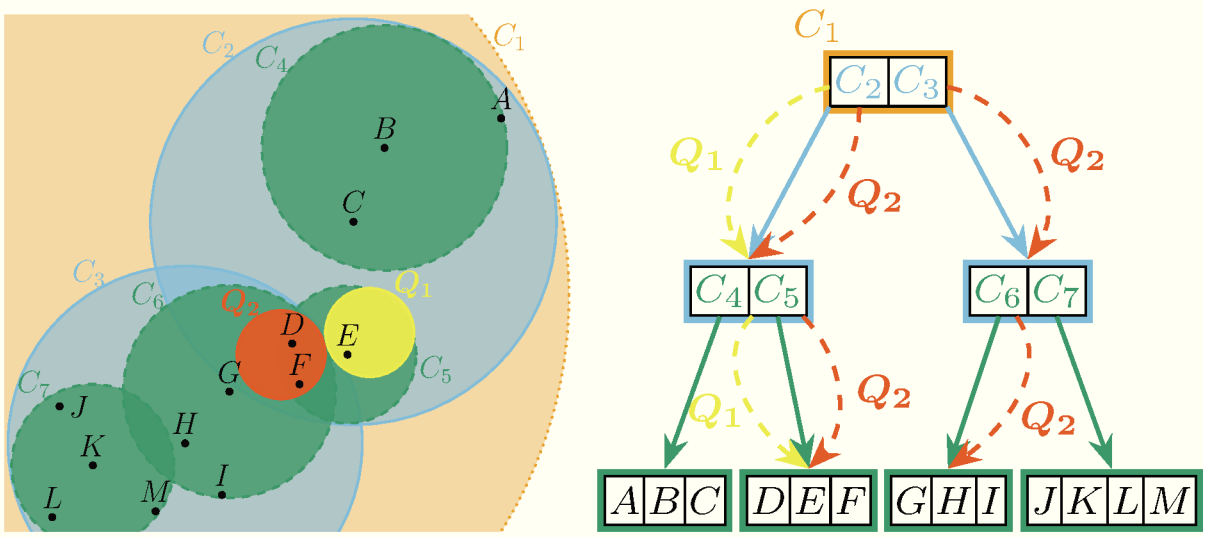
Agrupamento de Sequências Numéricas para Buscas por Similaridade em Lote

Neste capítulo foram adaptados algoritmos para acelerar buscas por similaridade realizadas em lote. Nesse tipo de busca, múltiplas sequências são consultadas simultaneamente e, ao final do processo, retorna-se separadamente os seus resultados. Essa técnica pode ser útil pois quando múltiplas consultas são processadas separadamente existe o problema de mal-uso dos dados recuperados. Por isso, agruparam-se sequências de consultas a fim de reduzir a quantidade de acesso à memória secundária. Dessa maneira, todas as sequências de consulta pertencentes a um mesmo grupo compartilham informações recuperadas pelas estruturas de indexação. Na Seção 6.1 são apresentados os algoritmos propostos para realização de buscas por similaridade em lote. Na Seção 6.2 são apresentadas cinco estratégias para agrupar sequências de consulta que foram utilizadas em trabalhos correlatos para outras finalidades. Por último, na Seção 6.3 são feitas as considerações sobre o uso dessa técnica.

6.1 Buscas por Similaridade em Lote

Como apresentado no Capítulo 2, existem duas etapas distintas para a realização de buscas por abrangência: a etapa de indexação e a etapa de recuperação. Na etapa de indexação, sequências são organizadas em estruturas multidimensionais ou métricas, tais como *R-Trees* e *M-Trees*, de modo que grupos de sequências similares pertençam a um mesmo nó-folha. Na etapa de recuperação, buscam-se sequências indexadas similares a uma sequência de consulta utilizando-se essa organização de nós. Dessa maneira, é possível realizar a poda de ramos não promissores, reduzindo a quantidade de acessos à memória secundária e o número de cálculos de similaridade entre a sequência de consulta e sequências candidatas.

Quando múltiplas consultas são processadas separadamente existe o problema de mal-uso dos dados recuperados durante a execução de cada busca. Consultas similares



(a) (b)

Figura 17 – Múltiplas consultas por similaridade em uma estrutura de indexação no formato de árvore. A Figura 17a ilustra o espaço indexado pela estrutura em 17b. Dadas duas consultas Q_1 e Q_2, representadas por círculos e flechas amarelos e laranjas, respectivamente, percebe-se que, se fossem realizadas duas buscas de maneira isolada, alguns nós seriam recuperados duas vezes da memória secundária, como é o caso dos nós apontados pelos *containers* C_2 e C_5, este último contendo as sequências D, E e F. Um método eficiente agruparia as sequências de consulta e recuperaria os nós candidatos apenas uma vez de modo que as buscas fossem processadas simultaneamente.

percorrem caminhos similares nas estruturas de indexação e, por isso, acessam a memória secundária repetidas vezes para recuperar nós recorrentes. Nesse caso, buscas por similaridade em lote são utilizadas para recuperação eficiente dos nós das estruturas de indexação. Primeiramente, agrupam-se consultas similares, as quais possivelmente percorreriam caminhos parecidos nas estruturas de indexação e, posteriormente, processa-se simultaneamente as consultas de cada grupo para evitar acessos repetidos à memória secundária. A Figura 17 ilustra o cenário onde múltiplas sequências similares são consultadas nas estruturas de indexação.

Uma estratégia simples de agrupamento de sequências de consulta cria um único grupo contendo todas as sequências e realiza apenas uma varredura no índice (MOON; WHANG; LOH, 2001). Essa estratégia pode melhorar a busca quando as sequências são similares entre si. Porém, quando as sequências são muito dissimilares, todo o índice pode ser recuperado e armazenado em memória primária. Além disso, a quantidade de computações de similaridade entre sequências aumenta uma vez que, durante a etapa de processamento dos nós-folhas, calcula-se a distância entre cada sequência de consulta e cada sequência candidata. Portanto, a escolha da estratégia de agrupamento das consultas impacta diretamente no desempenho de buscas por similaridade em lote.

O Algoritmo 19 apresenta o procedimento geral para realização de buscas por similari-

dade em lote. Dados um conjunto com N sequências de consulta $Q = \langle Q_1, Q_2, \ldots, Q_N \rangle$, uma função de agrupamento f e outros parâmetros a depender do tipo de consulta, tais como raio de abrangência ϵ ou quantidade de vizinhos k, o algoritmo retorna uma lista de resultados para cada consulta Q_i. Primeiramente, aplica-se a função f em Q para obter um conjunto $G = \langle G_1, G_2, \ldots, G_M \rangle$ contendo M grupos de sequências similares. Posteriormente, invoca-se um algoritmo de busca específico em lote para processar cada grupo G_j. Esse algoritmo pode ser uma busca em abrangência ou uma busca pelos k vizinhos mais próximas, por exemplo. Finalmente, as listas de resultados provenientes do processamento de cada grupo G_j são ordenadas conforme a disposição das consultas Q_i.

O Algoritmo 20 implementa a busca por abrangência em lote. Dados um grupo $Q = \langle Q_1, Q_2, \ldots, Q_N \rangle$ contendo N sequências de consulta, um raio de abrangência ϵ e o nó-raiz do índice, o algoritmo retorna uma lista de resultados para cada sequência Q_i. Primeiramente, cria-se um *container* que abrange a área de todas as sequências de consulta. Para isso, pode-se começar com um *container* que abrange uma única consulta, como apresentado no Capítulo 2, e, gradualmente, aumentá-lo a medida que uma nova sequência é considerada. Nota-se que, no caso de *R-Trees*, a ordem das sequências não influencia no retângulo final pois essa é uma estrutura espacial. Porém, no caso de *M-Trees*, a escolha da sequência inicial, ou centro da esfera, é importante uma vez que essa é uma estrutura métrica. Nesse caso, escolheu-se aquela cuja soma das distâncias para todas as outras sequências fosse a menor. Posteriormente, partindo-se do nó-raiz, o algoritmo percorre todas as ramificações subjacentes cujos *containers* possuam sobreposição com o *container* do grupo de consulta. Esse processo é realizado recursivamente até que todos os nós-folhas sejam alcançados. Quando algum nó-folha é alcançado, as sequências candidatas C_j pertencentes ao seus objetos são avaliadas para cada sequência de consulta Q_i e, se $dist(Q_i, C_j) \leq \epsilon$, então C_j é adicionada ao resultado da consulta Q_i.

Algoritmo 19 Busca genérica em lote

Entrada: Lista de sequências de consulta Q, estratégia de agrupamento f, nó-*raiz* e outros parâmetros como raio de abrangência ϵ ou quantidade de vizinhos k
Saída: Lista S contendo listas de resultados para cada consulta em Q

```
1: S ← {}
2: G ← f(Q)                          ▷ Aplica estratégia f para formar grupos
3: para todo grupo ∈ G faça
4:     S' ← Invoca algoritmo de busca com parâmetros adequados para processar grupo
5:     para resultado ∈ S' faça
6:         Encontre índice i que mapeia resultado para S
7:         Insira resultado em S_i
8: retorna S
```

Buscas por abrangência em lote também podem ser usadas juntamente com representações reduzidas de sequências numéricas tais como *Error-Bounded Piecewise Linear Approximation* (EBPLA) e *Adaptive Indexable Piecewise Linear Approximation* (AIPLA).

Algoritmo 20 Busca por abrangência em lote

Entrada: Grupo de consultas G, raio de abrangência ϵ e nó $raiz$
Saída: Resultado S contendo listas de sequências com distância menor ou igual que ϵ para as respectivas consultas

```
1: S ← P ← {}                                          ▷ Inicializa pilha P
2: C_G ← CRIA_CONTAINER(G, ε)          ▷ Cria container para um grupo de consultas
3: INSERE_PILHA(P, raiz)
4: enquanto PILHA_VAZIA(P) é falso faça
5:     nó ← REMOVE_PILHA(P)
6:     se nó é diretório então
7:         para todo obj ∈ OBJETOS(nó) tal que CONTAINER(obj) sobrepõe C_G faça
8:             INSERE_PILHA(P, FILHO(obj))
9:     senão se nó é folha então
10:        para todo obj ∈ OBJETOS(nó) faça
11:            para 1 ≤ i ≤ |G| faça
12:                se dist(G_i, SEQUÊNCIA(obj)) ≤ ε então
13:                    Insira SEQUÊNCIA(obj) em S_i
14: retorna S                                          ▷ Retorna listas de sequências
```

Para isso é necessário que a representação forneça uma medida *lower bounding*, como apresentado no Capítulo 2. Dessa maneira, primeiramente, indexam-se referências para as sequências originais utilizando-se as representações como chaves. Posteriormente, em uma busca por abrangência, buscam-se referências de sequências candidatas por meio da representação reduzida da sequência de consulta e, posteriormente, aplica-se uma etapa de pós-processamento para recuperar e descartar aquelas que não fazem parte de algum resultado.

6.2 Estratégias de Agrupamento de Consultas

Nesta seção são apresentadas cinco estratégias de agrupamento de consulta para realização de buscas por similaridade em lote. Essas estratégias foram adaptadas a partir de trabalhos relacionados que agruparam sequências em outros contextos, tais como, na etapa de indexação de subsequências (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994). A primeira estratégia é aquela que cria somente um grupo contendo todas as sequências de consulta. Neste trabalho, essa estratégia é referenciada como estratégia *Single Grouping* (SG). Além dessa estratégia, avaliaram-se outras quatro, descritas nas seções seguintes, para verificar a importância do método de agrupamento de consultas para o processamento de consultas por similaridade em lote, sendo elas: *N-Random Grouping* (NRG), *Maximum Capacity Grouping* (MCG), *Adaptive Grouping* (AG), *K-Medoids Grouping* (KMG).

6.2.1 Estratégia *N-Random Grouping*

Dados um conjunto de sequências de consulta e uma quantidade de grupos N, a estratégia *N-Random Grouping* (NRG) escolhe aleatoriamente N sequências para iniciar grupos e depois insere as sequências restantes nos grupos cujas sequências sejam as mais similares. A escolha do grupo de uma sequência é feita da seguinte maneira: para cada um dos N grupos, mantém-se um *container* que delimita suas sequências, inicialmente apenas uma sequência escolhida aleatoriamente. Posteriormente, cada consulta restante é inserida no grupo cujo *container* associado seja o melhor, expandindo-o se necessário. O melhor *container* é aquele que abrange totalmente a nova sequência ou, caso não exista, é aquele que ofereça o menor aumento de seu volume. Em caso de empate, escolhe-se aquele que tenha o menor volume.

6.2.2 Estratégia *Maximum Capacity Grouping*

A estratégia *Maximum Capacity Grouping* (MCG) limita a quantidade de sequências pertencentes a um grupo em vez de escolher uma quantidade fixa de grupos. Nessa estratégia não se utilizam heurísticas para formar grupos mais compactos. Dado um conjunto de sequências de consulta e a quantidade máxima M de sequências, primeiramente cria-se um novo grupo, e, posteriormente, tenta-se inserir as próximas M sequências. Caso ainda existam sequências sem grupo, cria-se um novo grupo e repete-se o processo. Esse processo é realizado até que todas as sequências de consulta pertençam a algum grupo.

6.2.3 Estratégia *Adaptive Grouping*

Em (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994), os autores empregaram uma técnica para agrupar pontos n-dimensionais em *Minimum Bounding Rectangle* (MBR) durante a operação de inserção em *R-Trees*. Dado um conjunto de pontos, primeiramente, essa técnica cria um novo MBR. Depois, é verificado se os pontos restantes devem ser incluídos no último MBR criado ou em ou novo MBR. Para isso, calcula-se o custo marginal $cm_{R\text{-}tree} = \prod_{i=1}^{d}(L_i + 0.5)/k$, onde L_i é a largura da i-ésima dimensão de um MBR e k é a quantidade de pontos no MBR. Dessa maneira, inclui-se um próximo ponto no MBR atual caso o custo marginal após a sua inserção seja inferior ou igual ao custo marginal anterior e, caso contrário, insere-se em um novo MBR.

A estratégia *Adaptive Grouping* (AG) é uma adaptação da técnica utilizada em (FALOUTSOS; RANGANATHAN; MANOLOPOULOS, 1994) para o contexto de agrupamento de sequências. Diferente da técnica anterior, na estratégia AG, verificam-se todos os *containers* criados até o momento para a inserção de uma nova sequência de consulta e não somente o último criado. Além disso, no caso de *M-Trees*, o custo marginal é dado por $cm_{M\text{-}tree} = (r + 0.5)^d/k$, onde r é o raio de uma esfera, d é a dimensão suposta do

espaço e k a quantidade de sequências nesse grupo. Esse custo é justificável pois seu valor cresce proporcionalmente ao volume da esfera, assim como o custo marginal $cm_{R\text{-}trees}$.

O Algoritmo 21 implementa a estratégia AG. Dado um conjunto de sequências de consulta $Q = \langle Q_1, Q_2, \ldots, Q_N \rangle$, retornam-se grupos de sequências similares de modo que não seja possível adicionar mais sequências a algum grupo sem que seu custo marginal aumente. Primeiramente, inicializam-se um conjunto de grupos G e um conjunto de *containers* C. Posteriormente, cria-se o primeiro grupo contendo Q_1 e adiciona-se a G. Depois, a partir de Q_1 cria-se um novo *container*, associado a Q_1, e insere-se em C. Para cada sequência restante Q_i, avalia-se o custo marginal de cada *container* ao adicioná-la a um dos grupos previamente criados. Se o custo marginal de um dos *containers*, após a sua expansão, for menor ou igual que seu custo marginal anterior, adiciona-se Q_i no grupo correspondente e atualiza-se o seu *container*. Caso contrário, cria-se um novo grupo contendo Q_i e associa-se a um novo *container* criado a partir de Q_i. Esse procedimento é realizado até que todas as sequências pertençam a algum grupo.

Algoritmo 21 Estratégia *Adaptive Grouping*

Entrada: Conjunto de N sequências de consulta Q
Saída: Lista de grupos G contendo listas de sequências similares entre si

1: $G \leftarrow C \leftarrow \{\}$ ▷ Inicializa conjuntos de grupos G e de *containers* C
2: Adicione $\{Q_1\}$ a G
3: Adicione $\{\text{CRIA\_CONTAINER}(Q_1)\}$ a C
4: **para** $2 \leq i \leq N$ **faça**
5: *novo_grupo* $\leftarrow$ **verdadeiro**
6: **para** $1 \leq j \leq |G|$ **faça**
7: $c \leftarrow \text{EXPANDE\_CONTAINER}(C_j, Q_i)$
8: **se** $\text{CUSTO\_MARGINAL}(c, |G|+1) \leq \text{CUSTO\_MARGINAL}(C_j, |G_j|)$ **então**
9: Adicione Q_i a G_j
10: $C_j \leftarrow c$
11: *novo_grupo* $\leftarrow$ **falso**
12: **saia do laço**
13: **se** *novo_grupo* **então**
14: Adicione $\{Q_i\}$ a G
15: Adicione $\{\text{CRIA\_CONTAINER}(Q_i)\}$ à C
16: **retorna** G

6.2.4 Estratégia *K-Medoids Grouping*

A estratégia *K-Medoids Grouping* (KMG) encontra K grupos ao redor de sequências representativas, chamadas *medoids*, de modo que a soma de todas as sequências para seus *medoids* seja minimizada. Existem vários algoritmos na literatura para encontrar os *K-medoids* em um conjunto de sequências numéricas. O mais conhecido é o algoritmo *Partition Around Medoids* (PAM) (ROUSSEEUW; KAUFMAN, 1990). Porém, nesta

dissertação, utilizou-se o algoritmo descrito em (PARK; JUN, 2009) pois, apesar de sua simplicidade, ele consegue melhor desempenho que o algoritmo PAM em termos de tempo.

Dado um conjunto de sequências numéricas, o Algoritmo 22 descreve o processo para encontrar os K grupos de sequências de consulta a partir dos *medoids*. Primeiramente, computa-se a matriz de distância entre todas as sequências para utilização nas etapas seguintes. Essa matriz é utilizada em todas as etapas subsequentes para evitar a recomputação de distâncias entre sequências.

Na etapa seguinte, formam-se K grupos iniciais $G = \langle G_1, G_2, \ldots, G_K \rangle$ utilizando-se uma heurística. Essa etapa é importante pois, quanto melhor forem os *medoids* iniciais, mais rápida será a convergência do algoritmo. Primeiramente, computa-se a soma das distâncias normalizadas $v_j = \sum_{i=1}^{n} (M_{ij} / \sum_{l=1}^{n} M_{il})$ para cada sequência j, onde M_{ij} são as distâncias precomputadas entre a sequência j e as sequências i, e $\sum_{l=1}^{n} M_{il}$ é o fator de normalização referente à sequência i. Posteriormente, escolhem-se as K sequências que obtiveram os menores valores v_j para serem os *medoids* iniciais e atribuem-se as sequências restantes para o grupo cujo *medoid* seja o mais próximo.

Na próxima etapa, iterativamente, realizam-se modificações locais em G de modo a minimizar custo dessa configuração, sendo o custo definido pela soma das distâncias entre todas as sequências e os *medoids* de seus respectivos grupos. Para isso, escolhem-se novas sequências como *medoids* e reatribuem-se as sequências restantes para grupos cujo novo *medoid* seja o mais próximo. A atualização dos *medoids* é feita de modo que a soma das distâncias intragrupo das sequências escolhidas para as outras seja a menor naquele grupo. Dessa maneira, se o custo da nova configuração for menor que o custo da configuração anterior, mantém-se a nova configuração e repete-se o processo. Caso contrário, finaliza-se o algoritmo e a configuração atual é retornada.

Algoritmo 22 Estratégia *K-Medoids Grouping*

Entrada: Conjunto de sequências de consulta Q e quantidade de grupos K
Saída: Lista de grupos G contendo listas de sequências similares entre si

1: $M \leftarrow$ COMPUTA_MATRIZ(Q) ▷ Computa matriz de distância entre sequências
2: Para cada sequência j em Q, compute $v_j = \sum_{i=1}^{n} (M_{ij} / \sum_{l=1}^{n} M_{il})$
3: Escolha as K sequências com menor valor v_j para serem os *medoids* iniciais
4: Forme K grupos $G = \langle G_1, G_2, \ldots, G_K \rangle$ a partir das sequências escolhidas
5: Insira as sequências restantes no grupo G_i cujo *medoid* seja o mais próximo
6: *custo_anterior* $\leftarrow \infty$
7: *custo_atual* $\leftarrow$ CUSTO(G) ▷ Computa custo da configuração atual G
8: **enquanto** *custo_atual* $<$ *custo_anterior* **faça**
9: Atualize os *medoids* de cada grupo G_i
10: Reinsira as outras sequências no grupo G_i cujo *medoid* seja o mais próximo
11: *custo_anterior* $\leftarrow$ *custo_atual*
12: *custo_atual* $\leftarrow$ CUSTO(G)
13: **retorna** G

6.3 Considerações

Neste capítulo, apresentou-se uma proposta de abordagem para acelerar consultas por similaridade em lote. O método tradicional realiza uma busca para cada sequência de consulta. Porém, quando sequências no mesmo lote são similares, caminhos parecidos nas estruturas de indexação são percorridos. Por isso, alguns nós são recuperados repetidas vezes durante as múltiplas buscas. Diferentemente, a abordagem proposta na Seção 6.1 agrupa sequências similares e realiza apenas uma varredura no índice por grupo. Dessa maneira, economiza-se acessos à memória secundária.

Para que a abordagem proposta seja eficiente, deve-se escolher uma estratégia de agrupamento adequada e uma quantidade suficiente de sequências por grupo. Quando consultas muito dissimilares são colocadas em um mesmo grupo, muitos nós-folhas podem ser recuperados e, para cada nó-folha, todas as sequências candidatas são comparadas com todas as sequências de consulta no grupo, aumentando o número de cálculos de distância. Além disso, se um grupo possui muitas consultas, deve-se manter uma lista de resultados para cada consulta, aumentando-se o espaço necessário em memória primária. Dessa maneira, a escolha da estratégia de agrupamento e a quantidade máxima de sequências por grupo impactam diretamente no desempenho de consultas por similaridade em lote.

Na Seção 6.2, foram apresentadas 5 estratégias de agrupamento para estudar o problema de buscas por similaridade em lote. As estratégias apresentadas foram: a estratégia SG, que cria um grupo somente contendo todas as sequências do lote; a estratégia NRG, que cria N grupos a partir de N sequências escolhidas aleatoriamente; a estratégia MCG, que limita a quantidade máxima de sequências por grupo; a estratégia AG, que utiliza uma função de custo para verificar se um novo grupo deve ser criado; e a estratégia KMG, que faz uso de um algoritmo *K-Medoids* para criar K grupos a partir de K sequências centrais. Dessa maneira, espera-se que o uso dessas estratégias tragam informações relevantes no estudo desse tipo de consulta.

Nota-se que as representações reduzidas de sequências numéricas propostas nos Capítulos 4 e 5 também podem ser utilizadas juntamente com as estratégias de agrupamento para acelerar e diminuir o espaço necessário de buscas por similaridade em lote. Dessa maneira, todas os benefícios da utilização de representações reduzidas são adquiridos nesse tipo de consulta. Além disso, uma das dificuldades, que é o uso de memória principal proporcional a quantidade de sequências de consulta em um grupo, pode ser minimizada, uma vez que essas representações gastam uma fração do espaço utilizado pelas sequências originais.

Capítulo 7

Experimentos e Análise dos Resultados

Neste capítulo são apresentados experimentos para verificar a viabilidade de uso das representações *Error-Bounded Piecewise Linear Approximation* (EBPLA) e *Adaptive Indexable Piecewise Linear Approximation* (AIPLA), descritas nos Capítulos 4 e 5, e do método de agrupamento de consultas, apresentado no Capítulo 6, para realizar consultas por similaridade em lote. Na Seção 7.1 são descritos os experimentos para avaliação das representações reduzidas propostas. Nesses experimentos, as novas representações foram comparadas com as representações *Piecewise Aggregate Approximation* (PAA), *Discrete Fourier Transform* (DFT) e *Indexable Piecewise Linear Approximation* (IPLA) avaliando-se o erro médio de aproximação para as sequências originais e o seu *pruning power*, uma medida comumente empregada na literatura (KEOGH; RATANAMAHATANA, 2005b). Na Seção 7.2 é apresentado o experimento que comparou as 5 estratégias de agrupamento, apresentadas na Capítulo 6, para realização de consultas por similaridade em lote utilizando-se *R-Trees* e *M-Trees*. A estratégia de referência para a comparação dessas estratégias foi a abordagem tradicional que realiza uma busca na estrutura de indexação para cada consulta no lote. Finalmente, na Seção 7.3 são feitas considerações sobre as técnicas desenvolvidas nesta dissertação.

7.1 Representações de Sequências Numéricas

Nesta seção, dois experimentos foram realizados para avaliação das representações EBPLA e AIPLA, propostas nos Capítulos 4 e 5, respectivamente. As novas representações foram comparadas com as representações PAA, DFT e IPLA. No primeiro experimento mensurou-se o erro médio de aproximação das representações para as sequências numéricas originais. No segundo, avaliou-se o *pruning power* de cada representação para verificar se sua medida *lower bounding* consegue descartar rapidamente sequências candidatas não promissoras durante o processamento de consultas por similaridade.

O erro de aproximação de uma representação para a sua sequência original é obtido reconstruindo-se uma nova sequência a partir da representação reduzida e calculando-

se a distância para a sequência original. Dessa maneira, dadas uma sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$ e sua representação reduzida X, primeiramente reconstrói-se uma nova sequência $S' = \langle s'_1, s'_2, \ldots, s'_n \rangle$ a partir de X e, posteriormente, calcula-se $dist(S, S')$. Nesta dissertação utilizou-se o quadrado da distância euclidiana $dist^2(S, S')$, descrito na Equação 23.

$$dist^2(S, S') = \sum_{i=1}^{n} (s_i - s'_i)^2 \tag{23}$$

Para reconstrução de uma sequência S' a partir de cada uma das representações foram realizados os seguintes procedimentos. No caso da representação PAA, mapeou-se cada média aos índices correspondentes da sequência original. Como, nos experimentos realizados, a quantidade de coeficientes PAA m é sempre múltiplo do tamanho das sequências n, o mapeamento foi feito repetindo-se cada média, ordenadamente, n/m vezes. No caso da representação DFT, preencheu-se os coeficientes faltantes com o valor 0 e aplicou-se a transformação inversa da DFT. No caso das representações IPLA, EBPLA e AIPLA, utilizou-se a função de cada reta para recuperar os respectivos segmentos.

A medida *pruning power*, apresentado em (KEOGH; RATANAMAHATANA, 2005b), mensura o quão próximo é a medida *lower bounding* fornecida por uma representação de um medida de similaridade real. Se a medida *lower bounding* é próxima da medida real, então, muitas sequências candidatas não promissoras podem ser descartadas no processo de busca utilizando-se apenas a medida *lower bounding*. Dessa maneira, com o *pruning power*, é possível estimar, independentemente de estruturas de indexação, o desempenho de buscas por similaridade que utilizam representações reduzidas.

O *pruning power* é obtido calculando-se *pruning power* $= M/N$, onde, em uma busca 1-NN, M é a quantidade de vezes em que se descartou uma sequência candidata usando somente a medida lb_{rep} e N é o total de sequências candidatas. Dado um conjunto com N sequências $S = \langle S_1, S_2, \ldots, S_N \rangle$ e uma sequência de consulta Q, o Algoritmo 23 computa o *pruning power* de uma de representação rep de maneira ingênua. Primeiramente, encontra-se o vizinho mais próximo calculando-se as distâncias entre Q e todas as sequências em S. Depois, aproxima-se a sequência Q para a representação X e as sequências em S para as representações $\langle Y_1, Y_2, \ldots, Y_N \rangle$ Posteriormente, calcula-se a medida *lower bounding* entre X e todas as representações Y_i. Por último, retorna-se M/N, onde M é a quantidade de vezes em que $lb_{rep}(X, Y_i)$ foi maior que a distância real do vizinho mais próximo.

Para realização dos experimentos foram gerados conjuntos de sequências numéricas de tamanhos 2^p, onde $p \in [6, 10]$ a partir de um modelo de caminhada aleatória. Todas as sequências foram aproximadas utilizando-se 2^q coeficientes, onde $q \in [2, 5]$. No primeiro experimento, calculou-se o erro médio de aproximação das representações analisando-se conjuntos com 10^5 sequências. No segundo, para cada conjunto, escolheu-se uma sequência Q aleatoriamente e calculou-se o *pruning power* de cada representação com

Algoritmo 23 Estratégia ingênua para calcular o *pruning power* de uma representação

Entrada: Conjunto de sequências $S = \langle S_1, S_2, \ldots, S_N \rangle$, uma sequência de consulta Q e uma função f que aproxima uma sequência como uma representação rep
Saída: *pruning power* pp de uma representação rep

```
1: 1nndist ← ∞
2: para todo sequência ∈ S faça
3:     1nndist ← min(1nndist, dist(Q, sequência))
4: M ← 0
5: X ← f(Q)                    ▷ Aplica f para aproximar Q como uma representação rep
6: para todo sequência ∈ S faça
7:     se lb_rep(X, f(sequência)) > 1nndist então
8:         M ← M + 1
9: retorna M/N
```

as sequências restantes. A média de 10^2 repetições desse processo foi retornado como resultado avaliando-se conjuntos contendo 10^3, 10^4 e 10^5 sequências.

As representações EBPLA e AIPLA usam algoritmos adaptativos e, por isso, encontram um número variável de coeficientes. Para realizar uma comparação justa foram escolhidos valores para o parâmetro ϵ_{seg}, o erro máximo por segmento, tal que, dado um conjunto de sequências, a quantidade média de coeficientes para aproximar cada sequência fosse aproximadamente m. No caso da representação AIPLA, assim como IPLA, são necessários 2 coeficientes por reta e, no caso da representação EBPLA, são necessários 4 coeficientes por ponto. Dessa maneira, comparou-se m coeficientes das representações PAA e DFT com $m/2$ retas das representações IPLA e AIPLA com $m/4$ pontos da representação EBPLA.

A Tabela 2 mostra os resultados do primeiro experimento, que computou os erros médios da aproximação de cada representação para as sequências originais. Percebe-se que a aproximação de sequências por retas resulta em um erro médio de aproximação menor, sendo que a representação AIPLA, proposta nesta dissertação, oferece a melhor alternativa nesse quesito. Nota-se que, na representação EBPLA, para aproximar a sequência com uma única reta usam-se 8 coeficientes. Por isso, devido à alta quantidade de coeficientes necessários para adicionar mais retas, a representação EBPLA obteve erros médios altos.

A Figura 18 apresenta o complemento do *pruning power*, sendo $1 - pruning\ power$, de cada representação variando a dimensionalidade das sequências, a quantidade de coeficientes utilizados pelas representações e a quantidade de sequências no conjunto de candidatos. O complemento foi utilizado para melhor visualização dos resultados. Percebe-se que todas as representações possuem *pruning power* alto com exceção da representação EBPLA. Desse modo, para essas representações, muitas sequências candidatas podem ser rapidamente descartadas durante o processo de buscas nas estruturas de indexação utilizando-se somente a medida *lower bounding*. Quando são utilizados 16 coeficientes, por exemplo, a diferença no *pruning power* é mínima dentre essas representações. Observa-se também que o *pruning power* das representações lentamente aumenta proporcionalmente

Tabela 2 – Erro médio de aproximação entre as representações PAA, DFT, IPLA, EBPLA e AIPLA, com m coeficientes, para as sequências originais correspondentes de tamanho n. A medida utilizada para calcular o erro foi a distância euclidiana quadrática, apresentada na Equação 23.

		erro médio				
m	n	PAA	DFT	IPLA	EBPLA	AIPLA
4	64	1.44	1.92	1.25	5.25	1.07
8	64	0.77	0.88	0.65	5.25	0.53
16	64	0.38	0.40	0.32	1.33	0.24
32	64	0.16	0.16	0.13	0.49	0.07
4	128	2.68	3.58	2.32	9.82	2.02
8	128	1.44	1.65	1.22	9.83	1.01
16	128	0.74	0.78	0.62	2.61	0.49
32	128	0.36	0.36	0.30	1.00	0.22
4	256	5.11	6.83	4.42	19.05	3.80
8	256	2.73	3.14	2.32	19.07	1.91
16	256	1.42	1.49	1.18	5.13	0.95
32	256	0.72	0.72	0.59	2.02	0.46
4	512	9.87	13.23	8.53	37.00	7.42
8	512	5.28	6.08	4.47	37.00	3.71
16	512	2.75	2.89	2.29	9.91	1.85
32	512	1.41	1.40	1.15	4.05	0.92
4	1024	19.27	25.90	16.64	72.01	14.32
8	1024	10.33	11.89	8.74	72.05	7.12
16	1024	5.37	5.65	4.46	19.82	3.61
32	1024	2.75	2.75	2.25	8.06	1.80

ao número de sequências no conjunto considerado. Nota-se que a representação EBPLA obteve baixo *pruning power* pois ela usa muitos coeficientes para armazenar poucas retas. Por exemplo, 32 coeficientes representam uma sequência com apenas 8 pontos, ou 7 retas. Em contrapartida, 32 coeficientes nas representações IPLA e AIPLA equivalem a 16 retas.

7.2 Estratégias de Agrupamento para Consultas por Similaridade em Lote

Nesta seção, são apresentados os experimentos para avaliação das 5 estratégias de agrupamento de consultas, apresentadas no Capítulo 6, em consultas por similaridade em lote. Nesses experimentos, mensuraram-se a economia em acessos à memória secundária e em cálculos de distância comparando-se com a abordagem tradicional *No Grouping* (NG), que realiza uma busca independente para cada consulta no lote. Para isso, construiu-se índices contendo sequências previamente indexadas para o processamento de consultas por similaridade em lote.

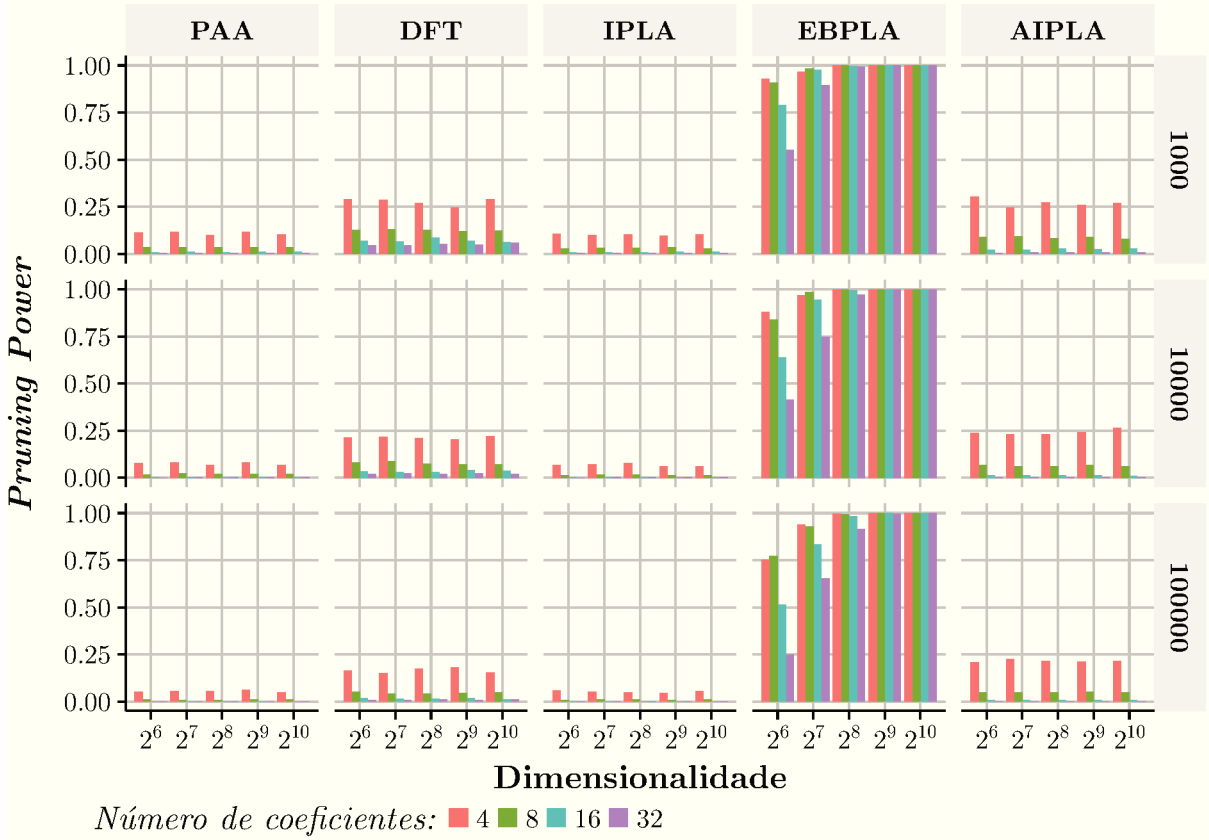


Figura 18 – Complemento do *pruning power*, sendo $1 - pruning\ power$, das representações PAA, DFT, IPLA, EBPLA e AIPLA. Os experimentos foram executados para sequências de tamanho 64 a 1024, geradas a partir de um modelo de caminhada aleatória. Essas sequências foram aproximadas com cada representação utilizando 4, 8, 16 e 32 para conjuntos contendo 10^3, 10^4 e 10^5 sequências numéricas, destacado verticalmente. Nota-se que a representação AIPLA, proposta nesta dissertação, obteve bons resultados, diferentemente da representação EBPLA. Utilizando-se 16 coeficientes a diferença entre *pruning power* é mínima para as outras representações.

A economia em acessos à memória secundária e_{disco} é igual à proporção $e_{disco} = disco_{NG}/disco_{outra}$, onde $disco_{NG}$ é a quantidade de acessos à memória secundária obtida pela estratégia NG e $disco_{outra}$ é a quantidade obtida por uma outra estratégia de agrupamento de consultas. Dessa maneira, se a quantidade de acessos à memória secundária da estratégia NG for maior que a quantidade de outra estratégia, então, essa estratégia foi mais econômica e $e_{disco} > 1$, caso contrário, não vale a pena o uso dessa estratégia de agrupamento e $e_{disco} < 1$. Uma formulação similar foi utilizada para computar a economia em cálculos de distância obtida por uma estratégia de agrupamento de consulta.

Nesses experimentos foram avaliados dois tipos de consultas por similaridade: consulta por abrangência, apresentada no Capítulo 2, e consulta do tipo contém; e dois tipos de estruturas de indexação: *R-Tree* e *M-Tree*. A consulta do tipo contém verifica a existência de uma sequência de consulta dentre as sequências previamente indexadas. De outra maneira, a consulta contém é simplesmente uma consulta por abrangência com raio de

abrangência igual a 0. Se alguma sequência é encontrada nessa busca, então, existe uma sequência no índice exatamente igual à sequência de consulta e, assim, retorna-se verdadeiro, caso contrário, retorna-se falso. No caso de consultas por abrangência escolheu-se o raio $\epsilon = 1/4\sqrt{E(D)}$, onde $E(D)$ é a energia média de um conjunto de sequências.[1] A energia média E de um conjunto de sequências D é dada por $E(D) = (\sum_{i=1}^{n} \sum_{j=1}^{d} |D_{ij}|^2)/n$, onde D_{ij} é o j-ésimo elemento da sequência i de tamanho d e n é a quantidade de sequências no conjunto. Esse valor permitiu recuperar aproximadamente a mesma quantidade de resultados para cada combinação de parâmetros.

Para realização dos experimentos foram gerados conjuntos contendo 2^5 sequências a partir de um modelo de caminhada aleatória de tamanhos 2^p, onde $p \in [2, 10]$. Posteriormente, todas as sequências foram normalizadas para o intervalo $[0, 1]$ utilizando-se a fórmula $s_i^{norm} = (s_i - \min(S))/(\max(S) - \min(S))$, onde s_i é o i-ésimo valor de S, tratado pela estrutura de indexação como uma dimensão, e, $\min(S)$ e $\max(S)$ são, respectivamente, o menor e o maior valor na sequência S. Por último, cada conjunto de sequências normalizadas foram previamente indexadas em *R-Trees* e *M-Trees* para realização de consultas por similaridade em lote.

As consultas por similaridade em lote foram organizadas da seguinte maneira: dado um lote contendo N sequências de consulta, primeiramente utilizou-se uma das estratégias para agrupá-las e, posteriormente, realizou-se uma busca para cada grupo formado. Nesse experimento foram utilizados lotes com tamanhos: 25, 100 e 1000. No caso das estratégias *N-Random Grouping* (NRG) e *K-Medoids Grouping* (KMG), escolheu-se o número de grupos para ser o logaritmo base 2 e a raiz quadrada do tamanho do lote. Na apresentação dos resultados, manteve-se somente os melhores resultados dentre as duas escolhas. No caso da estratégia *Maximum Capacity Grouping* (MCG), escolheu-se uma capacidade máxima tal que os grupos resultantes possuíssem quantidades de sequências similares àquelas das estratégias NRG e KMG.

A Figura 19 apresenta os resultados de economia em acessos à memória secundária das 5 estratégias avaliadas. Nesses experimentos, a estratégia *Single Grouping* (SG) obteve os melhores resultados na maioria dos cenários avaliados. Entretanto, na combinação: tipo de estrutura de indexação igual a *R-Tree* e ϵ igual a 0; todas as estratégias de agrupamento, exceto a estratégia NRG, apresentaram declínio em relação à economia em acessos à memória secundária conforme o aumento da dimensionalidade. Nota-se que o uso da estratégia NRG foi positivo em todos os cenários avaliados e, por isso, sugere-se que essa estratégia seja considerada em consultas por similaridade em lote. Durante a execução desses experimentos, observou-se que a estratégia *Adaptive Grouping* (AG) geralmente encontrou muitos grupos e, por isso, aproximou-se da abordagem tradicional.

Percebe-se também que a economia em acessos à memória secundária aumenta pro-

[1] Nesta dissertação não foram realizados experimentos para verificar a sensibilidade das consultas com o aumento da dimensionalidade, porém, os autores admitem que esse tipo de experimento seria interessante para a análise dos resultados.

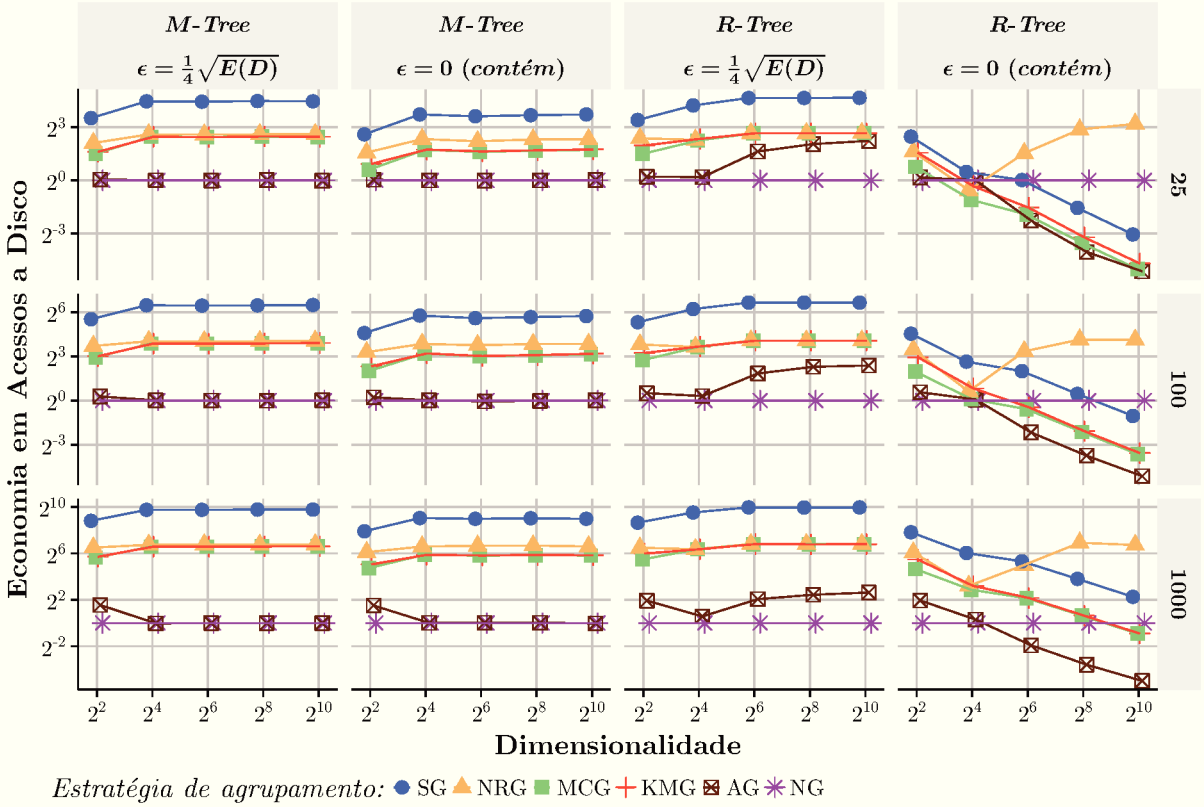


Figura 19 – Economia em acessos à memória secundária ao utilizar estratégias de agrupamento de consultas para realizar buscas por similaridade em lote. A estratégia de referência foi a abordagem NG, que realiza uma busca para cada sequência de consulta no lote. As estratégias comparadas foram SG, NRG, MCG, KMG e AG. Nota-se que a melhor estratégia na maioria dos cenários foi a estratégia SG. A estratégia NRG também obteve uma economia significativa em acessos à memória secundária, entretanto, diferente da estratégia SG, essa estratégia também obteve bons resultados nos cenários que utilizaram *R-Tree* e consultas do tipo contém ($\epsilon = 0$).

porcionalmente ao tamanho do lote. Por exemplo, a combinação de *R-Tree*, lote de tamanho 1000 e estratégia SG, usou aproximadamente 2^{10} vezes menos acessos à memória secundária para realizar as buscas por similaridade em lote com $\epsilon = {}^1/_4\sqrt{E(D)}$. O mesmo comportamento foi observado para a estrutura *M-Tree*.

A Figura 20 apresenta os resultados com relação à economia de cálculos de distância das estratégias de agrupamento. Percebe-se que somente os fatores tipo de estrutura de indexação e valor de ϵ têm influencia na economia de cálculos de distância. Nota-se que, quando $\epsilon = {}^1/_4\sqrt{E(D)}$, a diferença na quantidade de cálculos de distância das estratégias comparadas com a estratégia NG é insignificante. Entretanto, quando ϵ é igual a 0, desperdiça-se cálculos de distância ao utilizar das estratégias de agrupamento, exceto no cenário onde são utilizadas a estrutura *R-Tree* e a estratégia NRG.

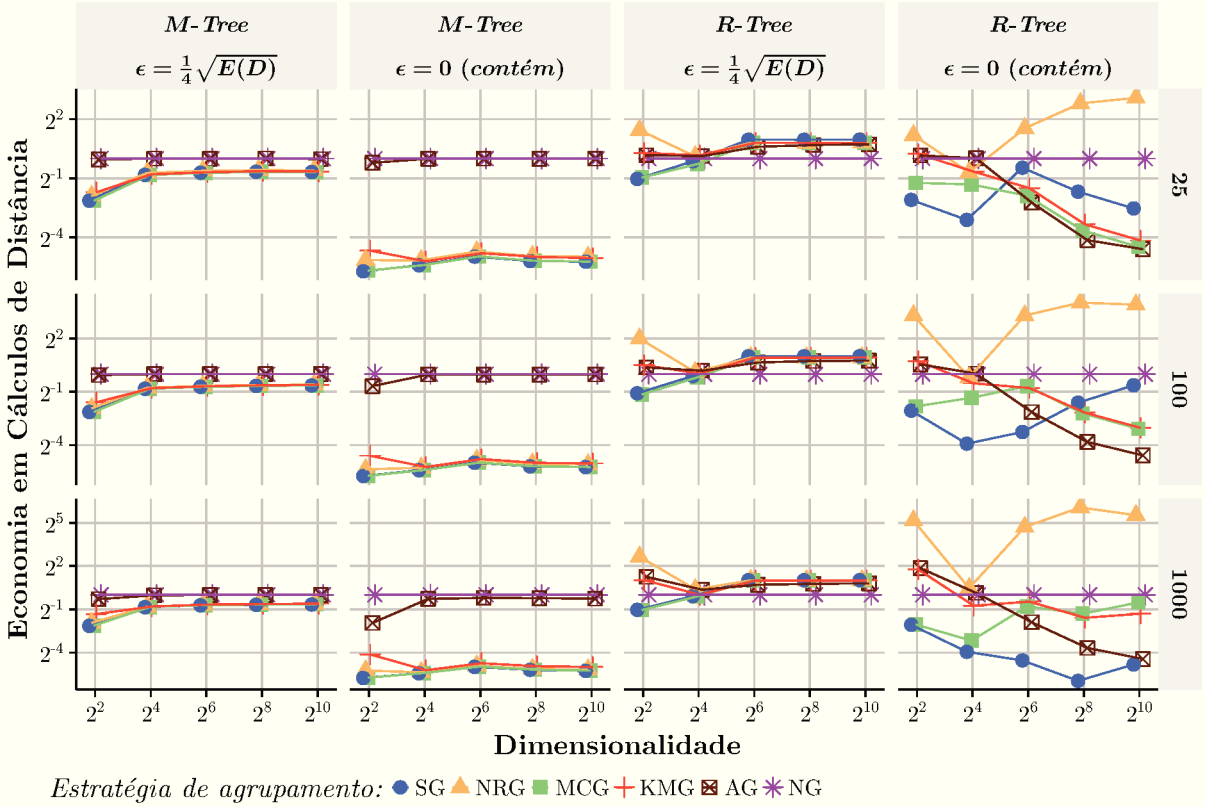


Figura 20 – Economia em cálculos de distância ao utilizar estratégias de agrupamento de consultas para realizar buscas por similaridade em lote. Como na Figura 19, as estratégias comparadas com a estratégia NG foram SG, NRG, MCG, KMG e AG e avaliaram-se os mesmos parâmetros. Nota-se que para consultas por abrangência com $\epsilon = {}^{1}/_{4}\sqrt{E(D)}$ o número de cálculos de distância foi semelhante entre todas as estratégias. Entretanto, para $\epsilon = 0$, deve-se analisar o tipo de estrutura utilizada e a estratégia de agrupamento, sendo a melhor, nos casos positivos, a estratégia NRG.

7.3 Considerações

Neste capítulo foram apresentados os resultados de experimentos para avaliação das novas representações de sequências numéricas e das estratégias de agrupamento de consultas estudadas. A primeira técnica utiliza aproximações para indexação e recuperação das sequências originais e, a segunda, agrupa sequências de consulta para reduzir a quantidade de acessos à memória secundária em consultas por similaridade em lote.

Na Seção 7.1, mostrou-se que o uso da representação EBPLA não é viável para indexação e recuperação de sequências. Apesar de utilizar um algoritmo estado da arte para extração das retas (KEOGH et al., 2001c), essa representação usa muitos coeficientes para o armazenamento de informações sobre envelopes de erro. Dessa maneira, poucas retas são usadas, em comparação com outras técnicas similares, para aproximar sequências.

Diferentemente, a representação AIPLA apresentou a melhor aproximação para as sequências correspondentes e obteve *pruning power* similar ao de técnicas comumente

utilizadas na literatura. Portanto, as suas retas descrevem melhor as tendências das sequências originais e podem ser utilizadas para evitar cálculos de distância entre sequências candidatas em buscas por similaridade. Dessa maneira, a representação AIPLA é viável em aplicações que usem tendências de sequências como parte de seu processamento.

Na Seção 7.2, mostrou-se que algumas estratégias para agrupamento de sequências de consulta podem ser utilizadas para otimizar o desempenho de buscas por similaridade em lote, tanto para *R-Trees* quanto para *M-Trees*. A estratégia SG, apesar de sua simplicidade, obteve os melhores resultados na maioria dos cenários avaliados. No caso de buscas por abrangência com raio $\epsilon > 0$, essa estratégia economizou até 2^{10} acessos à memória secundária enquanto o número de cálculos de distância permaneceu praticamente o mesmo. Entretanto, em outros cenários, deve-se avaliar a aplicação para a escolha da melhor estratégia.

No caso de buscas com raio de abrangência $\epsilon = 0$ em *M-Trees*, a maioria das estratégias economizaram em acessos à memória secundária, porém, aumentaram a quantidade necessária de cálculos de distância. Por isso, nesse caso, deve-se analisar o custo das operações de acesso à memória secundária e o custo de cálculo de distâncias para a escolha da melhor estratégia de agrupamento. No caso de buscas com raio de abrangência $\epsilon = 0$ em *R-Trees*, todas as estratégias, com exceção a estratégia NRG, perderam sua efetividade com o aumento da dimensionalidade das sequências. Desse modo, a estratégia NRG parece ser uma boa escolha para uso geral, uma vez que essa também foi a segunda melhor estratégia nos outros casos e possui um mecanismo para limitar a quantidade de memória principal utilizada durante buscas por similaridade em lote, diferentemente da estratégia SG.

Nota-se que ambas as técnicas podem ser usadas simultaneamente. Dessa maneira, pode-se reduzir a quantidade alta de memória primária necessária em buscas por similaridade em lote que utilizam estratégias de agrupamento de consulta. Além disso, obtém-se outros benefícios como, diminuição de acessos à memória secundária e cálculos de distância, aumentando-se o desempenho das consultas.

Capítulo 8

Conclusão

Esta dissertação propôs, desenvolveu, implementou e avaliou técnicas para otimizar consultas por similaridade em conjuntos de sequências numéricas previamente indexadas. A primeira abordagem utilizou representações reduzidas para indexação e recuperação de sequências numéricas em *M-Trees*. A segunda abordagem explorou técnicas de agrupamento de consultas para realização de consultas por similaridade em lote utilizando tanto *R-Trees* como *M-Trees*. Dessa maneira, pôde-se estudar o problema de indexação e recuperação de sequências sob diferentes perspectivas.

Foram encontrados poucos trabalhos que utilizaram técnicas adaptativas para extrair quantidades variáveis de coeficientes para aproximar cada sequência. A representação *Adaptive Piecewise Constant Approximation* (APCA) (KEOGH et al., 2001b) foi uma delas, porém, durante uma consulta por similaridade, computa-se uma nova representação APCA para cada representação recuperada do índice. Por isso, a seguinte hipótese foi proposta:

Hipótese 1. *A utilização de representações que usam um conjunto de coeficientes de tamanho adaptável às características das sequências numéricas pode acelerar consultas por similaridade no estilo correspondência por sequência inteira.*

Para validar essa hipótese, propôs-se e desenvolveu-se duas novas representações de sequências numéricas: a representação *Error-Bounded Piecewise Linear Approximation* (EBPLA) e a representação *Adaptive Indexable Piecewise Linear Approximation* (AIPLA). Essas representações utilizaram algoritmos adaptativos para extração de retas, ou tendências, que melhor representassem as sequências originais. Nos Capítulos 4 e 5 também foram propostas medidas *lower bounding* para indexação e recuperação das sequências originais sem a necessidade de haver uma extração de coeficientes para cada comparação, assim como na representação APCA. Posteriormente, no Capítulo 7 implementaram-se as duas representações e avaliaram-se o erro médio de aproximação e a eficiência durante a fase de recuperação por meio do seu *pruning power* (KEOGH; RATANAMAHATANA, 2005b).

A representação EBPLA, apresentada no Capítulo 4, adaptou o algoritmo *bottom-up*, apresentado em (KEOGH et al., 2001c), de modo que também fossem extraídas informações referentes aos erros de cada segmento de reta. Esse algoritmo utilizou o método de interpolação para obtenção de pontos, pertencentes às sequências originais, cujos erros dos segmentos de reta fossem limitados por um parâmetro ϵ. Intuitivamente, acreditava-se que o erro médio de aproximação dessa representação compensaria o espaço adicional necessário para armazenamento de seus coeficientes. Contudo, verificou-se no Capítulo 7 que tanto o erro de aproximação médio quanto o *pruning power* da representação EBPLA foram inferiores a de outras representações avaliadas em uma comparação justa. Portanto, não recomenda-se o seu uso para indexação e recuperação de sequências numéricas enquanto for necessário o armazenamento explícito dos coeficientes adicionais.

A outra representação proposta, a representação AIPLA, apresentada no Capítulo 5, utilizou um algoritmo adaptativo para extração de retas *Indexable Piecewise Linear Approximation* (IPLA) (CHEN et al., 2007). Esse algoritmo usou uma abordagem *top-down* em que, recursivamente, dividem-se segmentos em partes iguais até que todas as aproximações por retas IPLA tenham erro menor que um parâmetro ϵ_{seg}. Durante a computação da medida lb_{AIPLA}, no caso particular em que alinham-se mais de uma reta, pertencentes a uma das representações, com apenas uma de outra, divide-se a reta maior em pedaços e utiliza-se a medida lb_{IPLA} para computar a distância entre os pedaços e as retas correspondentes. Nesse caso, diminuiu-se 3ϵ, valor encontrado empiricamente, para que essa medida seja *lower bounding*. Durante os experimentos realizados no Capítulo 7, observou-se que a representação AIPLA possui o menor erro médio de aproximação dentre as representações avaliadas. Além disso, o seu *pruning power* foi similar ao das técnicas avaliadas, sendo possível sua utilização para descartar rapidamente sequências candidatas em consultas por similaridade. Por isso, essa representação pode melhorar o desempenho de consultas por similaridade ao mesmo tempo que melhores tendências são extraídas das sequências originais, satisfazendo a Hipótese 1.

Na segunda abordagem desta dissertação, estudou-se também uma técnica para realização de consultas por similaridade em lote. No decorrer dos estudos sobre as representações, mais especificamente durante a implementação das estruturas de indexação, percebeu-se que múltiplas consultas poderiam ser processadas simultaneamente durante uma única travessia na estrutura de indexação. Desse modo, estratégias de agrupamento de sequências de consulta poderiam reduzir o tempo de processamento de consultas por similaridade em lote. Por isso, a seguinte hipótese também foi proposta:

Hipótese 2. *Estratégias que agrupam sequências de consulta a fim de realizar consultas por similaridade em lote podem reduzir o tempo total para processamento de múltiplas consultas.*

Para verificar essa hipótese foram apresentadas 5 estratégias no Capítulo 6 para agrupar

sequências de consulta. Durante uma consulta por similaridade em lote, primeiramente agrupam-se sequências de consultas conforme a estratégia escolhida e, posteriormente, percorrem-se caminhos cujos *containers* de cada nó sobrepõem o *container* que engloba todas as consultas do grupo. Dessa maneira, evita-se a recuperação repetida de nós das estruturas de indexação para consultas que percorreriam caminhos similares nas estruturas de indexação. Posteriormente, no Capítulo 7, avaliaram-se a economia de acessos à memória secundária e do número de cálculos de distância em consultas por similaridade em lote utilizando as estratégias de agrupamento apresentadas.

Das 5 estratégias de agrupamento apresentadas no Capítulo 6 estão as estratégias: *Single Grouping* (SG), que cria somente um grupo para todas as consultas; *N-Random Grouping* (NRG), que escolhe N consultas aleatoriamente e forma N grupos a partir dessas sequências; *Maximum Capacity Grouping* (MCG), que cria grupos com uma capacidade máxima; *Adaptive Grouping* (AG), que usa uma função de custo para verificar se uma nova sequência deve ser inserida em um dos grupo criados ou se deve ser criado um novo grupo; e *K-Medoids Grouping* (KMG), que usa o algoritmo apresentado em (PARK; JUN, 2009) para encontrar K grupos a partir de K *medoids*. Observou-se, por meio dos experimentos realizados no Capítulo 7, que o uso dessas estratégias podem reduzir a quantidade de acessos à memória secundária mantendo-se o número de cálculos de distância nos casos mais frequentes. Isso aconteceu porque os nós recuperados das estruturas de indexação foram reutilizados por todas as consultas de um mesmo grupo de modo efetivo. Desse modo, o agrupamento de consultas otimizou o desempenho de consultas por similaridade, satisfazendo a Hipótese 2. Por exemplo, a estratégia SG, que cria somente um grupo contendo todas as consultas, reduziu até 2^{10} vezes com relação a quantidade de acessos à memória secundária em buscas por abrangência onde $\epsilon > 0$. A estratégia NRG, que cria N grupos a partir de N sequências aleatórias, também obteve bons resultados. Nota-se que essa estratégia permite limitar o uso de memória primária durante consultas por similaridade em lote.

Esse estudo sobre as estratégias de agrupamento para realização de consultas por similaridade em lote resultou em uma publicação na categoria *short*, que possui um limite de 6 páginas, intitulada *Empirical evaluation of strategies to process range queries of numeric sequences in batch-mode*, no 32º Simpósio Brasileiro de Banco de Dados (SBBD) no ano de 2017.

8.1 Sugestão de Trabalhos Futuros e Aplicações Derivativas desta Dissertação

Nesta dissertação foram identificadas algumas fraquezas nas técnicas estudadas. Por exemplo, percebeu-se que a representação EBPLA necessita do armazenamento de uma grande quantidade de coeficientes, referentes aos envelopes de erro de cada reta. Uma

possível solução para esse problema poderia usar o algoritmo *Feasible Space Window* (FSW), apresentado no Capítulo 3, para extração de coeficientes EBPLA. Essa técnica assegura que o erro em cada ponto não ultrapassa o parâmetro ϵ_p. Nota-se que outros algoritmos que também garantam essa afirmação também poderiam ser utilizados. Dessa maneira, não é necessário o armazenamento explícito dos erros δ^- e δ^+ de cada segmento para construção de envelopes.

Uma outra possibilidade de trabalho futuro seria prover à representação AIPLA uma medida *lower bounding* melhor. Mesmo produzindo aproximações com erros menores, a representação AIPLA não possui uma medida *lower bounding* melhor que a sua versão não adaptativa, a representação IPLA. Neste trabalho utilizou-se uma medida *lower bounding* pessimista que diminui 3ϵ de cada alinhamento de retas IPLA que não sejam do tipo IPLA (1 reta com 1 reta). Como o erro de aproximação da representação AIPLA é sempre melhor que sua versão não adaptativa, acredita-se que também exista uma outra medida *lower bounding* que faça com que o *pruning power* da representação AIPLA seja superior.

Neste trabalho considerou-se a quantidade de acessos à memória secundária e o número de cálculos de distância como os principais custos dos algoritmos de recuperação de sequências numéricas em consultas por similaridade em lote. Trabalhos futuros poderiam avaliar as estratégias de agrupamento apresentadas nesta dissertação utilizando outras medidas de qualidade como a quantidade de espaço necessário em memória primária para realizar consultas por similaridade em lote. Acredita-se que é importante mensurar o uso de memória principal nesse tipo de busca porque esse requisito pode aumentar consideravelmente ao agrupar múltiplas consultas.

Como as estratégias de agrupamento podem aumentar a quantidade necessária de memória principal em consultas por similaridade em lote, também seria interessante avaliar, em trabalhos futuros, o uso de representações reduzidas em consultas por similaridade em lote. Essa combinação poderia reduzir drasticamente o uso de memória principal uma vez que o espaço para armazenamento das representações reduzidas correspondem a apenas uma fração das sequências originais. Além disso, as representações também podem ser usadas para descartar rapidamente candidatos não promissores, otimizando o processamento de consultas por similaridade em lote.

Os experimentos conduzidos para avaliar todas as estratégias neste trabalho utilizaram um modelo de caminhada aleatória para geração de sequências numéricas. Outros trabalhos futuros poderiam utilizar outros modelos de geração para simular e verificar as particularidades de cada um. Desse maneira, pode-se avaliar as técnicas apresentadas nesta dissertação em diferentes contextos.

Outros trabalhos poderiam avaliar estruturas de indexação adicionais a *R-Tree* e *M-Tree* uma vez que podem sofrer efeitos distintos ao usar diferentes representações e estratégias de agrupamento de consultas para processamento em lote. Por exemplo, variações dessas estruturas tais como *R*-Tree* (BECKMANN et al., 1990) e *Slim-Tree* (TRAINA

et al., 2000), e outras estruturas em formato de árvore, como aquelas baseadas em pirâmides (BERCHTOLD; BöHM; KRIEGAL, 1998) poderiam ser estudadas. Nesse último caso, as estratégias de agrupamento seriam adaptadas para formar agrupamentos com o mesmo formato usado pelos *containers* da estrutura de indexação.

Por último, trabalhos futuros poderiam aplicar as técnicas estudadas nesta dissertação em problemas que se beneficiem da consulta por similaridade em lote e utilizem as informações precomputadas pela representação AIPLA durante o processamento de suas rotinas. Por exemplo, uma aplicação que usa imagens de satélites para monitorar o nível vegetativo de uma região ao longo do tempo poderia usar as técnicas propostas nesta dissertação da seguinte maneira: primeiramente, seria indexado um conjunto de sequências numéricas formadas a partir dos níveis médios de vegetação ao longo do tempo de áreas delimitadas. Posteriormente, uma das operações buscaria múltiplos padrões vegetativos para recuperar áreas com comportamentos similares utilizando consultas por similaridade em lote. Além disso, as representações AIPLA, usadas para indexação e recuperação, também poderiam ser utilizadas para análise de tendências das sequências originais, sendo possível observar intervalos temporais com crescimentos e declínios vegetativos e comparar com ações humanas realizadas naquela região.

Referências

AGRAWAL, R.; FALOUTSOS, C.; SWAMI, A. Efficient similarity search in sequence databases. In: _____. **Foundations of Data Organization and Algorithms: 4th International Conference, FODO '93 Chicago, Illinois, USA, October 13–15, 1993 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993. p. 69–84. ISBN 978-3-540-48047-1. Disponível em: <https://doi.org/10.1007/3-540-57301-1_5>.

BATISTA, G. E. A. P. A. et al. Cid: an efficient complexity-invariant distance for time series. **Data Mining and Knowledge Discovery**, v. 28, n. 3, p. 634–669, 2014. ISSN 1573-756X. Disponível em: <http://dx.doi.org/10.1007/s10618-013-0312-3>.

BECKMANN, N. et al. The R*-Tree: An efficient and robust access method for points and rectangles. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 19, n. 2, p. 322–331, maio 1990. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/93605.98741>.

BERCHTOLD, S.; BöHM, C.; KRIEGAL, H.-P. The pyramid-technique: Towards breaking the curse of dimensionality. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 27, n. 2, p. 142–153, jun. 1998. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/276305.276318>.

BERCHTOLD, S.; KEIM, D.; KRIEGEL, H. An index structure for high-dimensional data. **Readings in multimedia computing and networking**, Morgan Kaufmann, p. 451, 2001.

CHAN, K.-P.; FU, A. W.-C. Efficient time series matching by wavelets. In: **15th International Conference on Data Engineering**. Sydney, NSW, Australia: IEEE, 1999. p. 126–133. ISSN 1063-6382.

CHEN, Q. et al. Indexable PLA for efficient similarity search. In: **Proceedings of the 33rd International Conference on Very Large Data Bases**. VLDB Endowment, 2007. (VLDB '07), p. 435–446. ISBN 978-1-59593-649-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1325851.1325903>.

CHUNG, F.-L. et al. Flexible time series pattern matching based on perceptually important points. **Workshop on Learning from Temporal and Spatial Data in International Joint Conference on Artificial Intelligence (IJCAI'01)**, Seattle, Washington, USA, p. 1–7, 2001.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-Tree: An efficient access method for similarity search in metric spaces. In: **Proceedings of the 23rd International Conference on Very Large Data Bases**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (VLDB '97), p. 426–435. ISBN 1-55860-470-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645923.671005>.

COOLEY, J.; LEWIS, P.; WELCH, P. The finite fourier transform. **IEEE Transactions on audio and electroacoustics**, IEEE, v. 17, n. 2, p. 77–85, 1969.

CORMEN, T. H. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

DAUBECHIES, I. Orthonormal bases of compactly supported wavelets. **Communications on pure and applied mathematics**, Wiley Online Library, v. 41, n. 7, p. 909–996, 1988.

DING, H. et al. Querying and mining of time series data: Experimental comparison of representations and distance measures. **Proc. VLDB Endow.**, VLDB Endowment, v. 1, n. 2, p. 1542–1552, ago. 2008. ISSN 2150-8097. Disponível em: <http://dx.doi.org/10.14778/1454159.1454226>.

DUNFORD, N.; SCHWARTZ, J. T. Linear operators. **Interscience, New York**, v. 1, 1958.

EISEN, M. B. et al. Cluster analysis and display of genome-wide expression patterns. **Proceedings of the National Academy of Sciences**, v. 95, n. 25, p. 14863–14868, 1998. Disponível em: <http://www.pnas.org/content/95/25/14863.abstract>.

FALOUTSOS, C.; RANGANATHAN, M.; MANOLOPOULOS, Y. Fast subsequence matching in time-series databases. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 23, n. 2, p. 419–429, maio 1994. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/191843.191925>.

FU, A. W.-C. et al. Scaling and time warping in time series querying. **The VLDB Journal**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 17, n. 4, p. 899–921, jul. 2008. ISSN 1066-8888. Disponível em: <http://dx.doi.org/10.1007/s00778-006-0040-z>.

FU, T.-c. et al. Representing financial time series based on data point importance. **Engineering Applications of Artificial Intelligence**, v. 21, n. 2, p. 277 – 300, 2008. ISSN 0952-1976. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0952197607000577>.

FUCHS, E. et al. Online segmentation of time series based on polynomial least-squares approximations. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 32, n. 12, p. 2232–2245, 2010.

GIONIS, A.; INDYK, P.; MOTWANI, R. Similarity search in high dimensions via hashing. In: **Proceedings of the 25th International Conference on Very Large Data Bases**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. (VLDB '99), p. 518–529. ISBN 1-55860-615-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645925.671516>.

GUO, C.; LI, H.; PAN, D. An improved piecewise aggregate approximation based on statistical features for time series mining. In: _____. **Knowledge Science, Engineering and Management: 4th International Conference, KSEM 2010, Belfast, Northern Ireland, UK, September 1-3, 2010. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 234–244. ISBN 978-3-642-15280-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15280-1_23>.

GUTTMAN, A. R-Trees: A dynamic index structure for spatial searching. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 14, n. 2, p. 47–57, jun. 1984. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/971697.602266>.

HAAR, A. Zur theorie der orthogonalen funktionensysteme. **Mathematische Annalen**, Springer, v. 69, n. 3, p. 331–371, 1910.

HAMILTON, J. D. **Time series analysis**. Princeton, NJ, USA: Princeton University Press, 1994. v. 2.

HAN, W.-S. et al. A new approach for processing ranked subsequence matching based on ranked union. In: **Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2011. (SIGMOD '11), p. 457–468. ISBN 978-1-4503-0661-4. Disponível em: <http://doi.acm.org/10.1145/1989323.1989371>.

HILTON, P.; PEDERSEN, J. Catalan numbers, their generalization, and their uses. **The Mathematical Intelligencer**, v. 13, n. 2, p. 64–75, mar. 1991. ISSN 0343-6993. Disponível em: <https://doi.org/10.1007/BF03024089>.

HUTCHINSON, J. et al. Monitoring vegetation change and dynamics on u.s. army training lands using satellite image time series analysis. **Journal of Environmental Management**, v. 150, p. 355 – 366, 2015. ISSN 0301-4797. Disponível em: <//www.sciencedirect.com/science/article/pii/S0301479714003971>.

JAIN, A.; KUMAR, A. M. Hybrid neural network models for hydrologic time series forecasting. **Applied Soft Computing**, v. 7, n. 2, p. 585 – 592, 2007. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1568494606000317>.

KEOGH, E. et al. Dimensionality reduction for fast similarity search in large time series databases. **Knowledge and Information Systems**, v. 3, n. 3, p. 263–286, 2001. ISSN 0219-1377. Disponível em: <http://dx.doi.org/10.1007/PL00011669>.

_____. Locally adaptive dimensionality reduction for indexing large time series databases. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 30, n. 2, p. 151–162, maio 2001. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/376284.375680>.

_____. An online algorithm for segmenting time series. In: IEEE. **Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on**. San Jose, CA, USA, 2001. p. 289–296.

KEOGH, E.; RATANAMAHATANA, C. A. Exact indexing of dynamic time warping. **Knowledge and Information Systems**, v. 7, n. 3, p. 358–386, 2005. ISSN 0219-3116. Disponível em: <http://dx.doi.org/10.1007/s10115-004-0154-9>.

_____. Exact indexing of dynamic time warping. **Knowledge and Information Systems**, v. 7, n. 3, p. 358–386, mar. 2005. ISSN 0219-3116. Disponível em: <https://doi.org/10.1007/s10115-004-0154-9>.

LEMIRE, D. A better alternative to piecewise linear time series segmentation. In: SIAM. **SIAM International Conference on Data Mining**. Minneapolis, MN, USA, 2007. p. 545–550.

LIU, X.; LIN, Z.; WANG, H. Novel online methods for time series segmentation. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 20, n. 12, p. 1616–1626, 2008.

MEYER, Y. **Wavelets and operators**. [S.l.]: Cambridge university press, 1995. v. 1.

MOON, Y.-S.; WHANG, K.-Y.; HAN, W.-S. General match: A subsequence matching method in time-series databases based on generalized windows. In: **Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2002. (SIGMOD '02), p. 382–393. ISBN 1-58113-497-5. Disponível em: <http://doi.acm.org/10.1145/564691.564735>.

MOON, Y.-S.; WHANG, K.-Y.; LOH, W.-K. Duality-based subsequence matching in time-series databases. In: **Proceedings 17th International Conference on Data Engineering**. [S.l.: s.n.], 2001. p. 263–272. ISSN 1063-6382.

MUEEN, A. et al. Exact discovery of time series motifs. In: _____. **Proceedings of the 2009 SIAM International Conference on Data Mining**. [s.n.], 2009. p. 473–484. Disponível em: <http://epubs.siam.org/doi/abs/10.1137/1.9781611972795.41>.

ORCHARD, M. T. A fast nearest-neighbor search algorithm. In: IEEE. **Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on**. [S.l.], 1991. p. 2297–2300.

OTTERMAN, M. Approximate matching with high dimensionality R-Trees. **M. Sc. Scholarly paper, Dept. of Computer Science, Univ. of Maryland, College Park, MD**, 1992.

PARK, H.-S.; JUN, C.-H. A simple and fast algorithm for k-medoids clustering. **Expert Systems with Applications**, v. 36, n. 2, Part 2, p. 3336–3341, 2009.

PATEL, J. et al. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. **Expert Systems with Applications**, v. 42, n. 1, p. 259 – 268, 2015. ISSN 0957-4174. Disponível em: <//www.sciencedirect.com/science/article/pii/S0957417414004473>.

PIMENTEL, M. A. et al. A review of novelty detection. **Signal Processing**, v. 99, p. 215 – 249, 2014. ISSN 0165-1684. Disponível em: <//www.sciencedirect.com/science/article/pii/S016516841300515X>.

QI, J. et al. Indexable online time series segmentation with error bound guarantee. **World Wide Web**, v. 18, n. 2, p. 359–401, mar. 2015. ISSN 1573-1413. Disponível em: <https://doi.org/10.1007/s11280-013-0256-y>.

RAKTHANMANON, T. et al. Searching and mining trillions of time series subsequences under dynamic time warping. In: **Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2012. (KDD '12), p. 262–270. ISBN 978-1-4503-1462-6. Disponível em: <http://doi.acm.org/10.1145/2339530.2339576>.

ROUSSEEUW, P. J.; KAUFMAN, L. **Finding Groups in Data**. [S.l.]: Wiley Online Library, 1990.

ROUSSOPOULOS, N.; KELLEY, S.; VINCENT, F. Nearest neighbor queries. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 24, n. 2, p. 71–79, maio 1995. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/568271.223794>.

TRAINA, C. et al. Slim-Trees: High performance metric trees minimizing overlap between nodes. In: _____. **Advances in Database Technology — EDBT 2000: 7th International Conference on Extending Database Technology Konstanz, Germany, March 27–31, 2000 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 51–65. ISBN 978-3-540-46439-6. Disponível em: <http://dx.doi.org/10.1007/3-540-46439-5_4>.

WELCH, P. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. **IEEE Transactions on audio and electroacoustics**, IEEE, v. 15, n. 2, p. 70–73, 1967.

XI, X. et al. Fast time series classification using numerosity reduction. In: **Proceedings of the 23rd International Conference on Machine Learning**. New York, NY, USA: ACM, 2006. (ICML '06), p. 1033–1040. ISBN 1-59593-383-2. Disponível em: <http://doi.acm.org/10.1145/1143844.1143974>.

YANKOV, D. et al. Detecting time series motifs under uniform scaling. In: **Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2007. (KDD '07), p. 844–853. ISBN 978-1-59593-609-7. Disponível em: <http://doi.acm.org/10.1145/1281192.1281282>.

ZHANG, Y.; GLASS, J. R. A piecewise aggregate approximation lower-bound estimate for posteriorgram-based dynamic time warping. In: INTERSPEECH. Florence, Tuscany, Italy, 2011. p. 1909–1912.