UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

VINICIUS DE PAULA SILVA
Orientador: Vander Luis de Souza Freitas

TSAPI: UMA APLICAÇÃO DISTRIBUÍDA PARA COMPARAÇÃO DE SÉRIES TEMPORAIS

Ouro Preto, MG
2022

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

VINICIUS DE PAULA SILVA

TSAPI: UMA APLICAÇÃO DISTRIBUÍDA PARA COMPARAÇÃO DE SÉRIES TEMPORAIS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Vander Luis de Souza Freitas

Ouro Preto, MG
2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S586t Silva, Vinicius De Paula.
TSAPI [manuscrito]: uma aplicação distribuída para comparação de séries temporais. / Vinicius De Paula Silva. - 2022.
43 f.: il.: color., gráf., tab., mapa.

Orientador: Prof. Dr. Vander Luis de Souza Freitas Freitas.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Graduação em Ciência da Computação .

1. Séries temporais. 2. Similaridade entre séries temporais. 3. Redes complexas. 4. Similaridade entre redes complexas. 5. Computação distribuída. I. Freitas, Vander Luis de Souza Freitas. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004

Bibliotecário(a) Responsável: Luciana De Oliveira - SIAPE: 1.937.800

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Vinicius de Paula Silva

TSAPI: uma aplicação distribuída para comparação de séries temporais

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 25 de Outubro de 2022.

Membros da banca

Vander Luis de Souza Freitas (Orientador) - Doutor - Universidade Federal de Ouro Preto
Rafael Alves Bonfim de Queiroz (Examinador) - Doutor - Universidade Federal de Ouro Preto
Eduardo José da Silva Luz (Examinador) - Doutor - Universidade Federal de Ouro Preto

Vander Luis de Souza Freitas, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 01/11/2022.

Documento assinado eletronicamente por **Vander Luis de Souza Freitas**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 01/11/2022, às 13:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0416021** e o código CRC **42F35E88**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

Esta mongorafia apresenta uma aplicação para comparação de séries temporais. Uma vez capturados, os dados de séries temporais podem ser comparados para verificar particularidades no sistema em estudo e assim explorar seu comportamento. As comparações podem ser realizadas por algoritmos e métricas tradicionais, sendo eles: Informação Mútua, Coeficiente de Correlação de Pearson e *Dynamic Time Warping*. Além disso, há também a possibilidade de se transformar as séries em redes complexas, as quais são comparadas com os algoritmos *GCD-11*, *NetLSD* e *Portrait Divergence*. Cada algoritmo ou métrica apresenta suas particularidades, sendo mais ou menos eficaz na captura das diferenças a depender do tipo de série. Como resultado, a aplicação gera uma matriz de similaridades, a partir da qual é possível checar o resultado das comparações. Internamente, a ferramenta proposta utiliza um *middleware* distribuído e tolerante a falhas, de modo que este possibilita a execução em paralelo dos métodos de comparação no formato de um *pipeline*, tendo em vista que, a depender do número total de séries temporais e do algoritmo escolhido, o tempo para as comparações pode ser alto.

Palavras-chave: Séries temporais. Similaridade entre séries temporais. Similaridade entre redes complexas. Computação distribuída.

Abstract

This work presents an application to compare time series. Once included, the time series data can be checked to verify the specifics of the system in comparison and thus explore its behavior. In this monograph, comparisons can be made using traditional algorithms and metrics, as follows: mutual information, Pearson's Correlation coefficient and *Dynamic Time Warping*. In addition, there is the possibility of transforming time series into complex networks, which are also compared with the algorithms: *GCD-11*, *NetLSD* and *Portrait Divergence*. Each algorithm or metric has its particularities, being more or less effective in capturing differences depending on the type of series. As a result, the application generates a similarity matrix, from which it is possible to check distance between time series. Internally, the proposed tool uses a distributed and fault-tolerant *middleware*, so that it allows the parallel execution of the comparison methods in the form of a *pipeline*, considering that, depending on the number of time series and the chosen algorithm, the time for comparisons can be high.

Keywords: Time series. Similarity between time series. Similarity between complex networks. Distributed systems.

Lista de Ilustrações

Lista de Abreviaturas e Siglas

VM	Máquina Virtual
DTW	Dinamic Time Warping
MI	Informação Mútua
CP	Correlação de Pearson
DCTIF	Dynamical Characterization With the Top Integral Function
DCSD	Dynamical Characterization With Symbolic Dynamics
VG	Visibility Graph
NetLSD	Network Laplacian Spectral Descriptor

Lista de Símbolos

r	Coeficiente de correlação de Pearson
$G(N, L)$	Grafo G com conjunto de nós N e conjunto de arestas L
$\bar{x}$	Média dos valores na série x
$\bar{y}$	Média dos valores na série y
$X_{n \times m}$	Matriz de custo mínimo de n linha e m colunas
GCM	Graphlet Correlation Matrix
M	Valor inteiro que representa o tamanho da janela deslizante
A_{ij}	Matriz de adjacência
S_{ij}	Matriz de similaridade

Sumário

1 Introdução

A infinidade de dispositivos tecnológicos presentes em nosso cotidiano possibilita que dados ambientais, comportamentais ou financeiros sejam coletados e guardados a todo momento. Dados numéricos capturados de alguma variável sequencialmente ao longo do tempo caracterizam uma série temporal (ESLING; AGON, 2012), cuja análise pode revelar informações sobre diferentes sistemas.

Na meteorologia, através de sensores é possível registrar os níveis diários de precipitação ou temperatura de alguma região. Em Mello e Viola (2013) os dados coletados são dispostos em formato de pluviograma e depois analisados para identificar as regiões mineiras mais atingidas por chuvas intensas, portanto, suscetíveis a erosão e inundação. No mercado financeiro a variação das cotações de ativos é registrada diariamente, o que é útil para treinar modelos de inteligência artificial capazes de prever as cotações futuras (KROLLNER et al., 2010). Na medicina, a captação da frequência cardíaca, ou seja, o intervalo de tempo entre os batimentos cardíacos, pode ser registrado (LUZ et al., 2016). Agliari et al. (2020) utilizaram estes dados para alimentar uma rede neural capaz de detectar anomalias na frequência cardíaca de pacientes e então identificar doenças como fibrilação atrial ou insuficiência cardíaca.

Frequentemente, quando trabalhando com séries temporais, é necessário identificar séries similares para formação de *clusters* para extração de dados, ou procurar em um banco de dados séries com determinado nível de correlação (WU; AGRAWAL; ABBADI, 2000). Essas tarefas são realizadas por funções de distância (do inglês *distance functions*) ou similaridade — rotinas que recebem como entrada duas séries temporais e retornam o quanto elas são similares.

Dependendo do domínio do sistema e as características das séries, escolher uma função de distância ideal pode trazer melhores resultados para a aplicação (relações lineares e não lineares), ou seja, capturar características das séries e compará-las com mais detalhes. Ferreira et al. (2021) demonstram que em um conjunto de séries temporais de medições de temperatura, o coeficiente de Correlação de Pearson (FILHO; JÚNIOR, 2009) identifica conexões de curto alcance (em relação a distâncias geográficas), porém, no mesmo conjunto de dados, *Dynamic time warping* (SAKOE; CHIBA, 1978) identificou as de longo alcance. Diante disso, o presente trabalho traz algoritmos tradicionais amplamente utilizados na literatura, que se enquadram como funções de similaridade: Correlação de Pearson, Informação mútua (GIERLICHS et al., 2008) e *Dynamic time warping*.

Além dos algoritmos previamente citados, esta monografia também utiliza algoritmos baseados em redes complexas para comparar séries temporais. Redes complexas tem sido utilizadas para resolver problemas em diversas áreas de pesquisa, como biologia e medicina (BARABÁSI; PÓSFAI, 2016). Ao transformar séries temporais em redes complexas é possível analisar as

características da rede e então compará-las por meio de algoritmos com o mesmo objetivo das métricas já apresentadas, para mensurar o quanto as redes são similares. O presente trabalho emprega as seguintes técnicas para transformação de séries temporais em redes complexas: *Visibility Graph* (LACASA et al., 2008), *Dynamical Characterization With the Top Integral Function* e *Dynamical Characterization With Symbolic Dynamics* (LACERDA; FREITAS; MACAU, 2016). Após a transformação das séries para redes empregamos os seguintes algoritmos para realizar a comparação das redes complexas resultantes e verificar a similaridade: *GCD-11* (YAVEROGLU et al., 2014), *NetLSD* (TSITSULIN et al., 2018) e *Portrait Divergence* (BAGROW; BOLLT, 2019).

A aplicação apresentada nesta monografia realiza a comparação de séries temporais a partir de diversos algoritmos disponíveis para a escolha do usuário. O seu principal diferencial está na possibilidade de utilização de algoritmos baseados em redes complexas para efetuar as comparações. Além disso, utiliza-se computação distribuída, possibilitando escalabilidade para comparar uma grande quantidade de séries temporais com maior velocidade.

Esta aplicação é oferecida ao usuário no formato de uma API devido à facilidade proporcionada por este formato de aplicação na troca de informações entre usuário e sistema. Afinal, uma API é uma *interface* de programação de aplicação utilizada para criação de *softwares* capazes de se comunicarem com outras plataformas. No decorrer desta monografia serão apresentados os padrões e protocolos estabelecidos para que o usuário a utilize e consiga efetuar comparações entre séries temporais.

1.1 Justificativa

Plataformas online como a Microsoft (2022) e Timescale (2022) oferecem somente a Correlação de Pearson como método de comparação de séries temporais. Sendo essa uma tarefa útil na análise desse tipo de dado, como pontuado por Ferreira et al. (2021), múltiplos algoritmos ou métricas podem ser usados para enriquecer a análise, além da Correlação de Pearson.

Esta monografia propõe uma aplicação denominada *TSAPI*, que significa *Time Series API*. Trata-se de uma API online capaz de comparar um conjunto de séries temporais fornecidas como entrada pelo usuário. A comparação é realizada através de algum algoritmo ou métrica dentre os vários disponibilizados, paralelamente e distribuída, buscando o melhor tempo possível para entrega dos resultados, incluindo como diferencial a possibilidade da escolha de algoritmos baseados em redes complexas. Essas funcionalidades são incluídas em uma API de fácil utilização pelos usuários, contribuindo com pesquisadores e analistas de dados interessados em séries temporais.

1.2 Objetivo geral e objetivos específicos

Esta monografia visa o desenvolvimento e validação da aplicação *TSAPI*, distribuída no formato de API, utilizada para comparação de séries temporais de forma paralela e distribuída. Os objetivos específicos são:

- Apresentar uma documentação simples e objetiva da aplicação proposta;
- Comparação de séries temporais paralelamente, a partir de um *middleware* que permite a distribuição de carga;
- Retornar para o usuário os resultados da comparação das séries temporais.

1.3 Organização do Trabalho

A monografia está organizada como segue. Revisão de literatura no Capítulo 2, onde são apresentados trabalhos relacionados e fundamentação teórica. No capítulo 3 é apresentado como ocorreu o desenvolvimento da aplicação, desde a elaboração da documentação até o desenvolvimento da arquitetura proposta. No capítulo 4 são apresentados resultados a respeito da aplicação proposta. No capítulo 5.1 são apresentadas as considerações finais.

1.3.1 Estrutura da Monografia

Capítulo 1: Introdução.

Capítulo 2: Revisão Bibliográfica/ Embasamento Teórico (com o referencial teórico e trabalhos relacionados).

Capítulo 3: Desenvolvimento.

Capítulo 4: Resultados e Discussões.

Capítulo 5.1: Conclusão (e trabalhos futuros).

2 Revisão Bibliográfica

Este capítulo versa sobre a revisão bibliográfica necessária para entendimento do trabalho. A Seção 2.1 apresenta os trabalhos relacionados e a Seção 2.2 a fundamentação teórica.

2.1 Trabalhos Relacionados

Séries temporais são conjuntos de observações medidas no tempo, utilizadas para modelagem e análise de problemas em diversos domínios (EHLERS, 2007). Seja para prever o futuro ou observar características de certos ambientes, a extração de informações está relacionada diretamente com a forma de análise escolhida. Diante das várias possibilidades de análise de séries temporais esta seção está dividida entre: ferramentas disponíveis no mercado para análise e comparação de séries temporais, e trabalhos acadêmicos que utilizam algoritmos de comparação de séries temporais.

No que se refere a ferramentas disponíveis no mercado, pode-se afirmar que a maioria delas está ligada a análise de dados e a *Business Intelligence*, campo conhecido por fazer análises de grandes volumes de dados sobre uma empresa e suas operações (SANTOS et al., 2019). Disponibilizada de forma open source, Metabase (2022) é uma ferramenta *web* criada em 2015 com código hospedado no *GitHub*[1] que oferece um ambiente completo de análise de dados. Ela pode ser executada por meio de um pacote *.jar*[2] ou por uma imagem *Docker*[3] em qualquer sistema com suporte a estas tecnologias. Ao se conectar com algum banco de dados, esta ferramenta consegue monitorar as pesquisas realizadas pelos usuários para obter *insights* sobre os dados em banco. Essas pesquisas originam gráficos, e seus resultados podem ser salvos e organizados em *dashboards* visualizáveis (SANTOS et al., 2019). Por meio de uma *interface* de fácil interação, a ferramenta permite que os usuários utilizem funções de agregação para obter dados do banco e visualizar a série em gráfico de barra, linha ou histograma, sendo possível também sobrepor duas séries capturadas em datas diferentes. Assim como é mostrado na Figura 2.1, a ferramenta expõe dados de séries temporais no banco de dados do usuário de forma simples e objetiva, facilitando a comparação por meio da visualização, porém não oferece algoritmos ou métricas de comparação de séries, diferente desta monografia, que utiliza algoritmos para a comparação das séries.

[1] Disponível em <https://github.com/metabase>
[2] Disponível em <https://www.java.com/pt-BR/>
[3] Disponível em <https://hub.docker.com/r/metabase/metabase/>

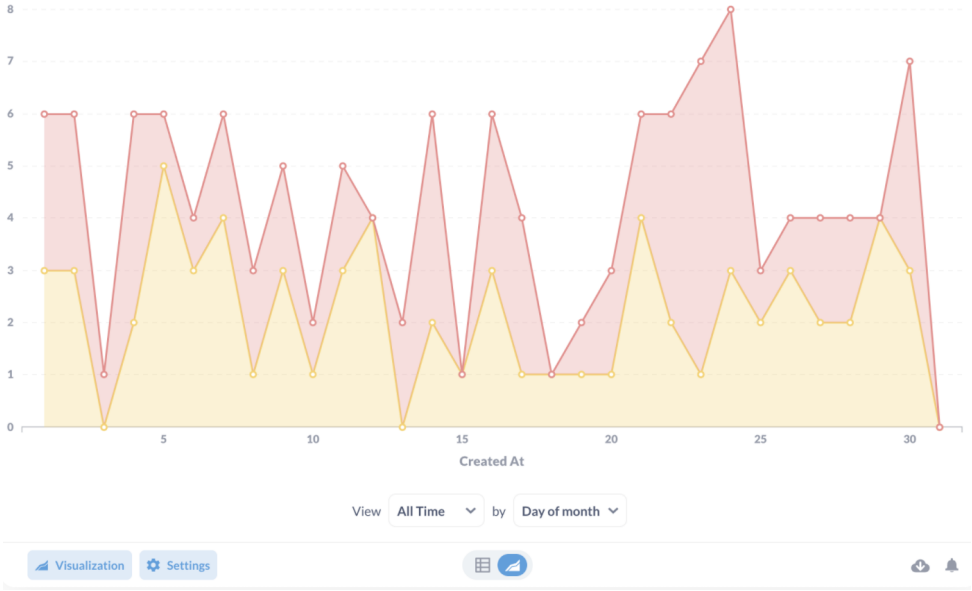

Figura 2.1 – Séries temporais exibidas na plataforma Metabase (METABASE, 2022)

Outra ferramenta comercial popular, incluída no ecossistema baseado em computação em nuvem *Azure*, é a *Azure Data Explorer* (MICROSOFT, 2022), que facilita a captação de séries temporais em tempo real por meio da computação distribuída e implementa funções para visualização e processamento desses dados. Com ela é possível aplicar operações nas séries como: operação de filtragem para detecção de mudança e suavização de ruídos; análise de regressão para ajustar a melhor linha a uma série temporal para detecção geral de tendências e; detecção de sazonalidade para observar padrões periódicos entre as séries (MICROSOFT, 2022). Em relação à comparação entre séries por meio de algoritmos, o único método que a ferramenta oferece é a Correlação de Pearson, também implementada na presente monografia. Todo o processamento e análise de dados citados são aplicados em uma página na *web* e as funções são executadas por meio da linguagem de consulta *Kusto*. A Figura 2.2 é exibe a visualização das séries na ferramenta.

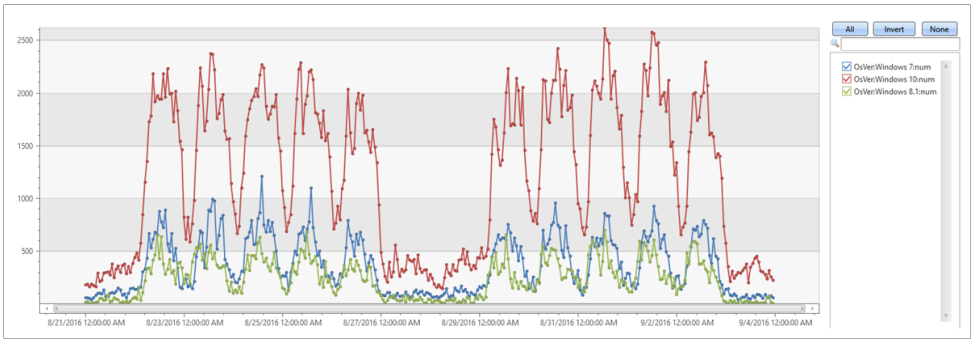

Figura 2.2 – Séries temporais exibidas na plataforma Azure Data Explorer (MICROSOFT, 2022)

No que se refere a trabalhos acadêmicos, Ferreira et al. (2021) utilizou várias técnicas de comparação de séries temporais para identificar fenômenos climáticos. Apesar de não apresentar *interface web* ou interação com usuários, sua importância como referência para este trabalho vem

do emprego de diversos algoritmos de comparação de séries temporais em várias séries, tendo como resultado uma matriz contendo as similaridades entre os pares de séries, como proposto nesta monografia. Ao utilizar séries temporais obtidas a partir de variações climáticas em volta da terra, foi possível criar uma rede funcional para cada algoritmo de comparação de séries. Em cada uma das redes, os *links* são formados a partir de um limiar que representa a similaridade mínima entre as séries de duas regiões. Dentre os algoritmos/métricas de similaridade, estão incluídos nesta monografia os seguintes: Correlação de Pearson (FILHO; JÚNIOR, 2009), Informação Mútua (GIERLICHS et al., 2008) e *Dynamic Time Warping* Sakoe e Chiba (1978).

Assim como visto em Ferreira et al. (2021), nesta monografia é apresentada uma API capaz de comparar um grande número de séries. Entretanto, as opções de algoritmos para comparação de séries temporais não se limitam às abordagens clássicas, mencionadas no parágrafo anterior, abrangendo também as baseadas em redes complexas, a partir das quais se converte as séries em redes (FREITAS; LACERDA; MACAU, 2019; LACASA et al., 2008) e avalia a similaridade entre elas (TANTARDINI et al., 2019). Diferenciando-se de Microsoft (2022), que utiliza a linguagem *Kusto* para obter as séries temporais, o presente trabalho de pesquisa permite que o usuário envie cada uma das séries, e a função de comparação é escolhida por meio de parâmetros enviados para a *API*.

2.2 Fundamentação Teórica

Nesta seção serão apresentados os conceitos iniciais sobre comparação de séries temporais (Subseção 2.2.1), os algoritmos clássicos utilizados (Subseção 2.2.2), os algoritmos de transformação de séries temporais em redes complexas (Subseção 2.2.3), e os algoritmos utilizados para comparar redes complexas (Subseção 2.2.4).

2.2.1 Comparação de Séries Temporais

O objetivo desta seção é demonstrar os algoritmos/métricas utilizados para quantificar a similaridade/diferença entre pares de séries temporais. Muitas técnicas realizam essa tarefa, resultando em uma distância (ou abstração do conceito de distância) entre as séries.

Existem algumas categorias de técnicas de comparação de séries temporais. Na subseção 2.2.2 (Algoritmos Tradicionais), o coeficiente de *Correlação de Pearson* e a *Informação Mútua* são classificados como *Feature-based Distance Functions*, em português Funções de Distância Baseadas em Características, que de acordo com Ferreira et al. (2021) são funções que extraem características descritivas das séries, e então as compara. Outra categoria utilizada neste trabalho da qual o algoritmo *Dynamic Time Warping* faz parte é conhecida como *Shape-based Distance Functions*, em português Funções de distância baseadas na forma. Algoritmos desta classe comparam a forma geral de duas séries temporais (FERREIRA et al., 2021). O resultado do processo de comparação de N séries temporais é uma matriz de similaridades S_{ij} de dimensão

$N \times N$. Dentro de cada posição (linha e coluna) dessa matriz encontramos o resultado da comparação entre duas séries temporais. Essa matriz pode ser utilizada, inclusive, na criação de redes funcionais (EGUÍLUZ et al., 2005), servindo como uma matriz de adjacências ponderada.

Na Subseção 2.2.3 são relatados os algoritmos utilizados neste trabalho para transformar séries temporais em redes complexas e logo depois, na Subseção 2.2.4, é descrito como os algoritmos responsáveis por comparar redes complexas trabalham.

2.2.2 Algoritmos Tradicionais

Este tópico apresenta técnicas já consolidados e amplamente utilizados na literatura para comparar séries temporais. Elas utilizam principalmente de métodos estatísticos e computacionais.

2.2.2.1 Coeficiente de Correlação de Pearson

Uma métrica capaz de avaliar relações lineares e, consequentemente, utilizada para comparar séries temporais é o Coeficiente de Correlação de Pearson. Essa comparação ocorre calculando-se o grau de correlação entre as séries, que vai determinar o quanto duas variáveis aleatórias estão associadas linearmente. Filho e Júnior (2009) esclarecem que duas variáveis se associam quando guardam semelhanças na distribuição dos valores, ou seja, através da distribuição das frequências ou pelo compartilhamento da variância. O Coeficiente de Correlação de Pearson é dado pela equação:

$$r = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma(x_i - \bar{x})^2 \Sigma(y_i - \bar{y})^2}}. \tag{2.1}$$

tal que $r \in [-1, 1]$, e $\bar{x}$ e $\bar{y}$ são respectivamente as médias das séries $X = \{x_1, x_2, \cdots, x_n\}$ e $Y = \{y_1, y_2, \cdots, y_n\}$. Quando $r = 1$ as duas séries são perfeitamente correlacionadas, ou seja, quando os valores de uma aumentam, os da outra também seguem a tendência de aumento. Por outro lado, $r = -1$ indica correlação negativa: aumento nos valores de uma série indicam decréscimo na outra e vice-versa.

A implementação do algoritmo está disponível no formato de um repositório no *GitHub*[4] de nome *SciPy*, a versão utilizada é a 1.9.2. Empregaram-se os parâmetros originais do algoritmo, fixados pela biblioteca.

2.2.2.2 Informação Mútua

Outra medida utilizada neste trabalho capaz de comparar séries temporais é o índice de Informação Mútua. Vinda do campo da teoria da informação, a informação mútua é uma medida que informa o quanto uma série temporal depende da outra de forma não linear. Mensura-se o

[4] https://github.com/scipy/scipy

quanto é possível saber sobre a série Y apenas ao observar a série X (GIERLICHS et al., 2008). A equação a seguir ilustra o cálculo da Informação Mútua:

$$I(X;Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X,Y) = I(Y;X), \tag{2.2}$$

de forma que $H(X)$, $H(X|Y)$ e $H(X,Y)$ são a entropia marginal de X, entropia condicional de X dado Y e entropia conjunta de X e Y, respectivamente. O resultado está contido no seguinte intervalo $0 \leq I(X;Y) \leq H(X)$. Logo, para obter a similaridade máxima, $I(X;Y)$ deve atingir um valor próximo a $H(X)$, sendo a quantidade de informação que pode-se extrair da série X observando apenas uma amostra. Quanto maior o valor retornado, maior será a similaridade entre as séries e caso o algoritmo retorne o valor 0 as séries são consideradas independentes, i.e., não existe similaridade entre elas.

A implementação do algoritmo utilizada está disponível no formato de um repositório no *GitHub*[5] de nome *SkLearn*. Utiliza-se aqui a versão 1.1.2 da biblioteca. O algoritmo empregado nesta monografia é uma variação que permite computar a informação mútua de variáveis contínuas apresentado por (KRASKOV; STÖGBAUER; GRASSBERGER, 2004), e os parâmetros originais fixados pela biblioteca não foram alterados.

2.2.2.3 Dynamic Time Warping

O algoritmo *Dynamic Time Warping* (DTW) compara duas séries temporais obtidas no mesmo espaço encontrando o alinhamento não linear ótimo entre elas. Foi desenvolvido por Sakoe e Chiba (1978) com o objetivo de reconhecer padrões de fala, e faz uso da programação dinâmica para diminuir o custo computacional. O algoritmo recebe como entrada duas séries temporais A e B de tamanho N e M respectivamente, e a partir delas constrói uma matriz de custo mínimo $X_{N\times M}$. Através dela é possível observar o melhor alinhamento entre as séries, assim como evidenciado na Figura 2.3.

A partir do alinhamento obtido é possível calcular a distância entre as séries. A média da distância euclidiana entre todos os pontos é considerada pelo presente trabalho para verificar a similaridade entre as séries.

[5] https://https://github.com/scikit-learn/scikit-learn

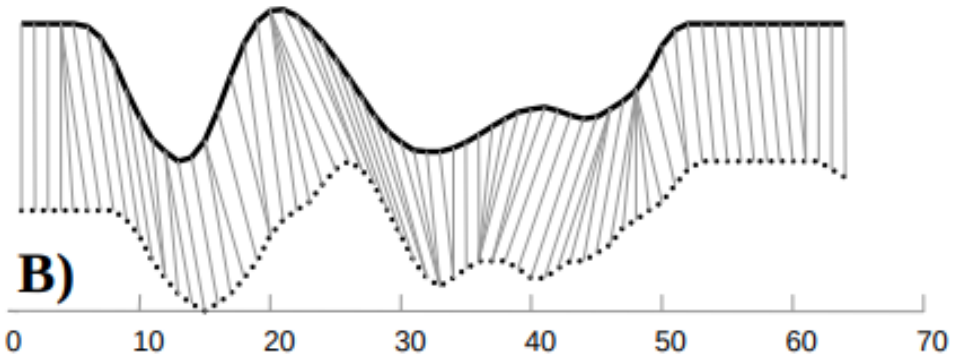

Figura 2.3 – Exemplo de alinhamento entre duas séries temporais, a partir do algoritmo DTW. Fonte: (KEOGH; PAZZANI, 2001)

A implementação do algoritmo utilizada está disponível no formato de um repositório no *GitHub*[6] de nome *dtw-python*. Utiliza-se aqui a versão 1.3.0. O algoritmo foi empregado nesta monografia sem alteração dos parâmetros padronizados e pré-definidos pela biblioteca.

2.2.3 Transformação de Séries Temporais em Redes Complexas

Uma rede complexa é definida por um grafo $G(N, L)$, onde N é o número de nós e L o número de arestas. Uma forma de se representar esse tipo de estrutura é a partir da chamada matriz de adjacência A_{ij}, onde $A_{ij} = 1$ quando existe uma conexão entre os nós i e j, ou $A_{ij} = 0$, caso contrário.

A partir da conversão de uma série temporal para uma rede complexa é possível utilizar métricas de rede para caracterizar e, por conseguinte, comparar a similaridade destas séries. Os algoritmos que realizam essa conversão procuram armazenar características das séries nas redes geradas.

Esforços têm sido aplicados para conseguir extrair características de séries temporais utilizando redes complexas (FREITAS; LACERDA; MACAU, 2019). Ao utilizar métodos baseados em redes complexas, esta monografia auxilia pesquisadores na exploração e avanço desta área de estudo.

As próximas Subseções apresentam algoritmos para conversão de séries temporais em redes complexas.

2.2.3.1 *Dynamical Characterization With the Top Integral Function*

O algoritmo *Dynamical Characterization With the Top Integral Function* mapeia a sequência de valores $x_0, x_1, ..., x_n \in [0, 1]$ da série temporal em valores de 1 até N utilizando a *Top integral function*:

[6] https://github.com/DynamicTimeWarping/dtw-python

$$Y_k = N * x_k = min\{i \in \mathbb{Z} | N * x_k \leq i\}, \tag{2.3}$$

sendo N o número de nós da rede a ser gerada, Y_k é o inteiro associado a x_k na posição k da série temporal, i é o menor valor inteiro do qual $N * x_k \leq i$, e cada Y_k corresponde a um nó na rede. As conexões da rede são estabelecidas a cada par de valores consecutivos (Y_k, Y_{k+1}). Nos testes realizados por Freitas, Lacerda e Macau (2019) o algoritmo consegue diferenciar séries caóticas de periódicas através de algumas métricas das redes resultantes, como o grau médio.

A implementação do algoritmo utilizada está disponível no formato de um repositório no *GitHub*[7]. O repositório apresenta uma única versão com o algoritmo corretamente implementado. Os parâmetros de execução não foram alterados.

2.2.3.2 *Dynamical Characterization With Symbolic Dynamics*

O algoritmo *Dynamical Characterization With Symbolic Dynamics* (DCSD) (LACERDA; FREITAS; MACAU, 2016) realiza a conversão de uma série temporal para uma rede complexa a partir da divisão da série em duas classes. Considerando uma série com valores no intervalo $[0, 1]$, cada ponto z é convertido para 0 se z está entre $[0, 0.5]$, e em 1 caso z esteja entre $(0.5, 1]$. Depois disso caminhamos na nova série binária gerada com uma janela deslizante de tamanho M, de modo que os bits da janela são convertidos para um número decimal. Avançamos uma casa e repetimos a conversão até o fim da série binária assim como mostrado na Figura 2.4.

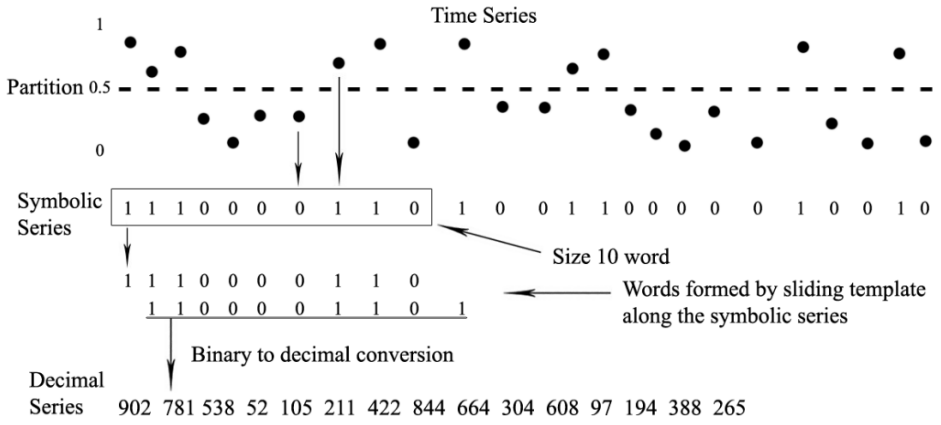

Figura 2.4 – Geração da série decimal a partir da conversão da série original para binária utilizando uma janela deslizante de tamanho $M = 10$. Fonte: (LACERDA; FREITAS; MACAU, 2016).

A rede é gerada observando o vetor de números decimais, de forma que cada valor é um novo nó, e as conexões são estabelecidas a cada par de valores consecutivos. *Loops* são permitidos visto que pode haver repetição de valores no caso de séries periódicas ou com algum comportamento repetitivo, e somente pode existir uma aresta conectando um par de nós.

[7] https://github.com/vanderfreitas/DCTIF_DCSD/tree/master/DCTIF

Assim como observado por Lacerda, Freitas e Macau (2016), o método consegue detectar períodos nas séries temporais através da observação de ciclos na rede gerada. Caso a série tenha somente 1 valor a rede também terá somente 1 nó, caso a série tenha 5 valores que se repetem periodicamente, teremos uma rede com 5 nós, formando um ciclo.

A implementação do algoritmo utilizada está disponível no formato de um repositório no *GitHub*[8]. O repositório apresenta uma única versão com o algoritmo corretamente implementado. Os parâmetros de execução não foram alterados e são os mesmos definidos pelo autor.

2.2.3.3 *Visibility Graph*

O método *Visibility Graph* (VG), apresentado por Lacasa et al. (2008), é conhecido por mapear séries temporais em redes complexas por meio de um critério de visibilidade entre os valores. Para facilitar a compreensão desse critério, imagine o gráfico de uma série temporal em formato de gráfico de barras, de forma que o valor de cada elemento é representado pelo eixo y e os elementos são posicionados no eixo x assim como mostra a Figura 2.5.

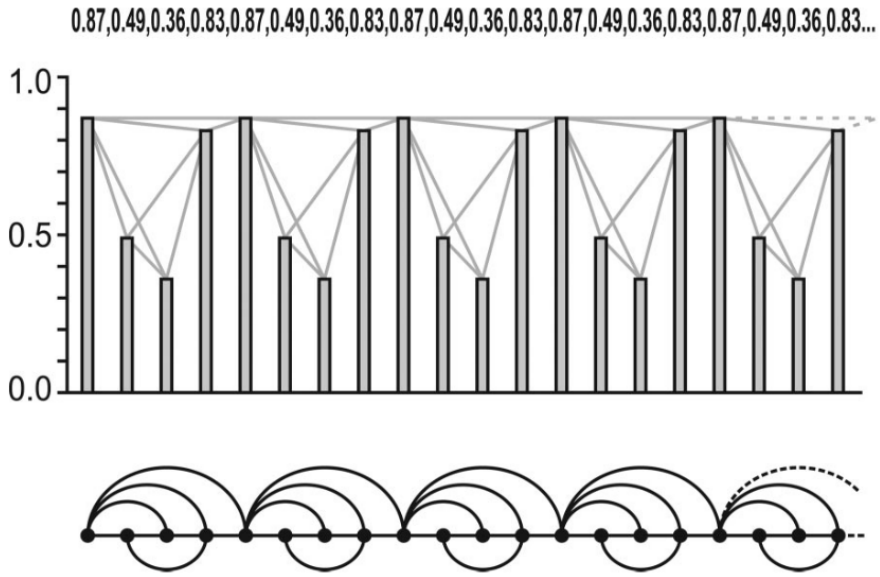

Figura 2.5 – Geração do grafo correspondente a uma série temporal com 20 valores. Fonte: (LACASA et al., 2008).

Assim como mostrado na Figura 2.5, uma aresta é estabelecida quando é possível criar uma reta de uma barra até outra sem cruzar outra barra. Formalmente, o critério de visibilidade pode ser escrito da seguinte forma: dois valores (t_a, y_a) e (t_b, y_b) formam uma conexão se a condição da equação a seguir se satisfizer para quaisquer outros pontos (t_c, y_c) posicionados entre eles:

$$y_c < y_b + (y_a - y_b)\frac{t_b - t_c}{t_b - t_a}. \tag{2.4}$$

[8] https://github.com/vanderfreitas/DCTIF_DCSD/tree/master/DCSD

A partir das redes geradas por este método, Lacasa et al. (2008) constatou que séries periódicas se transformam em redes regulares, séries aleatórias em redes aleatórias e séries fractais em redes livres de escala.

A implementação do algoritmo utilizada está disponível no formato de um repositório no *GitHub*[9] cujo nome é *ts2vg*, a versão utilizada é a 1.0.0. Os parâmetros utilizados foram os pré-definidos pela biblioteca.

2.2.4 Comparação de Redes Complexas

A partir das redes geradas pelos algoritmos da seção anterior, o próximo passo é mensurar suas similaridades e diferenças, a partir de suas características topológicas. Os métodos utilizados neste trabalho são aplicados em áreas como economia internacional para comparar a estrutura de vendas de diferentes categorias de produtos; em transporte, na comparação das redes de vôo de diferentes companhias aéreas, e outras aplicações (TANTARDINI et al., 2019).

Os algoritmos de transformação mostrados na Seção 2.2.3 geram redes de tamanhos diferentes. Diante disso, para comparar estas redes, os algoritmos utilizados neste trabalho pertencem à classe *Unknown node-correspondence (UNC)*, os quais permitem comparar qualquer par de grafos, com diferentes tamanhos, densidades ou vindos de diferentes campos, ou aplicações, e o mais importante, são grafos não direcionados e sem pesos nas arestas. Esses métodos resumem a estrutura global do grafo por meio de cálculos estatísticos, os valores obtidos são então utilizados para definir uma distância entre dois grafos, ou seja, é encontrado um valor que caracteriza o quanto eles são diferentes (TANTARDINI et al., 2019).

As subseções a seguir apresentam diferentes métodos para a comparação de redes.

2.2.4.1 GCD-11

O GCD-11 recorre à contagem de *Graphlets* para definir um valor de distância entre duas redes. *Graphles* são pequenos subgrafos encontrados em redes e são importantes para capturar características presentes na topologia de cada uma. Considerando-se todos os *Graphlets* de 2 a 5 nós, há 30 formatos possíveis. Para diferenciá-los, basta observar a quantidade de nós e a órbita em que cada nó pode ser classificado, a partir de 73 tipos diferentes de órbitas. A órbita de um nó é obtida ao avaliar as características dos nós conectados ao mesmo. A Figura 2.6 mostra quais são os tipos de *Graphlets*, classificando-os pelo número de nós. A figura também mostra as órbitas de cada nó contidos nós *Graphlets*, elas são denominadas pela letra G seguida pelo número da órbita.

[9] https://github.com/CarlosBergillos/ts2vg

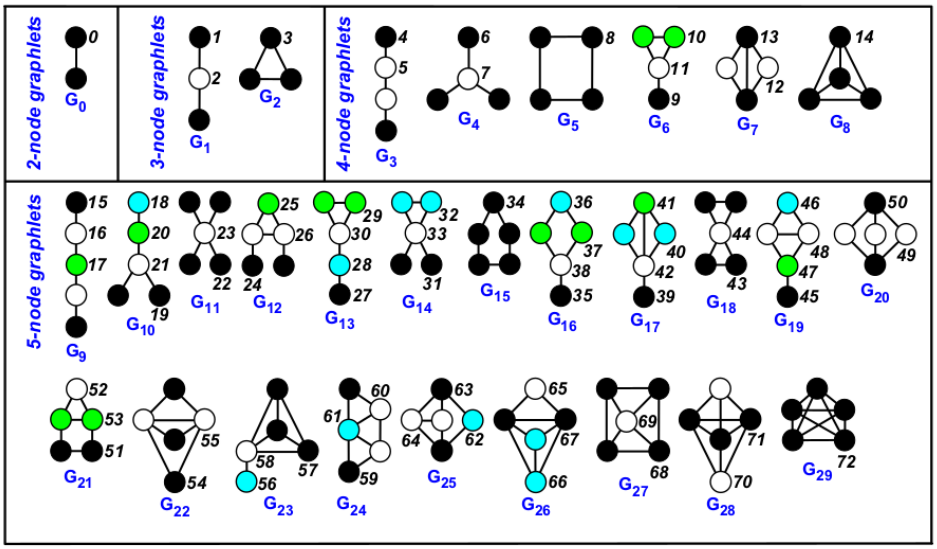

Figura 2.6 – 30 tipos diferentes de *Graphlets*, de 2 até 5 nós, e as 73 possíveis órbitas de cada nó. Fonte: (TANTARDINI et al., 2019).

O algoritmo *GCD-11*, apresentado por Yaveroglu et al. (2014), não utiliza todos os *graphlets* listados. Dispensando as órbitas consideradas redundantes ele obteve melhor eficiência na comparação. Logo, o algoritmo utiliza apenas as 11 primeiras órbitas listadas.

Para uma rede com N nós e um vetor de tamanho 11 em que cada posição representa uma órbita equivalente à numeração apresentada na Figura 2.6, a contagem de órbitas em que cada nó está inserido é adicionada a uma matriz $N \times 11$ elemento por elemento. Após o armazenamento da matriz, o Coeficiente de Correlação de Spearman é computado entre todos os pares de colunas, resultando em uma matriz 11×11 conhecida como *Graphlet Correlation Matrix (GCM)* (TANTARDINI et al., 2019). A distância entre duas redes R_1 e R_2 é definida então como a distância euclidiana entre a parte triangular superior das matrizes GCM_1 e GCM_2.

A implementação do algoritmo utilizado nesta monografia é encontrada no site de um dos autores[10].

2.2.4.2 NetLSD

O *Network Laplacian Spectral Descriptor* (NetLSD) (TSITSULIN et al., 2018), enquadra-se na classe de métodos espectrais, ou seja, ele usa o espectro das matrizes de adjacência das redes para o processo de comparação.

A captura de características das redes ocorre propagando ondas em vários nós do grafo e observando como elas se espalham com o passar do tempo. Os caminhos obtidos a partir da propagação das ondas são chamados de *assinatura de rastreamento* — uma sequência de dados obtidos a partir da observação do grafo em tempos diferentes. A distância entre dois grafos é obtida ao comparar suas *assinaturas de rastreamento*.

[10] Disponível em <http://www0.cs.ucl.ac.uk/staff/natasa/GCD/>

O algoritmo utilizado nesta monografia encontra-se disponível na biblioteca disponibilizada pelos autores do algoritmo em forma de repositório na ferramenta *GitHub*[11].

2.2.4.3 Portrait Divergence

Assim como o NetLSD, este método também se enquadra na classe de métodos espectrais. Criado por Bagrow e Bollt (2019), o método usa a distribuição do tamanho dos caminhos mínimos em um grafo, utilizada para construir uma matriz conhecida como *network portrait* M_{lk}. As linhas são representadas por $l = 0, 1, ..., d$ tal que d é o diâmetro do grafo, e as colunas são representadas por $k = 0, 1, ..., N - 1$, caracterizado pelo número de nós que alcançam k nós por meio do caminho mínimo de distância l. Para Tantardini et al. (2019), a matriz *network portrait* é uma ótima forma de sumarizar informações topológicas de um grafo, tais como número de nós e arestas, distribuição de graus e número de caminhos mínimos de tamanho l.

Para obter a distância entre dois grafos G_1 e G_2 é calculada a probabilidade $P(k, l)$ de ao selecionar aleatoriamente dois nós de distância l e, para um dos nós, este ter k nós a l nós de distância. Após obter as distribuições de probabilidade, o algoritmo utiliza o método *Jensen-Shannon Divergence* para verificar a similaridade entre elas.

A implementação do método utilizado nesta monografia encontra-se disponível no repositório de um dos autores [12].

2.2.5 Tecnologias utilizadas

Visando oferecer os métodos comparadores para os usuários de forma simples, esta monografia apresenta um sistema disponibilizado no formato de uma *API*. Nas próximas subseções são apresentadas as ferramentas utilizadas na construção de cada parte do mesmo.

2.2.5.1 Flask

O Flask é conhecido por ser um micro-framework de código aberto que permite a criação de sistemas *Web* utilizando a linguagem *Python*. Sua principal vantagem perante os outros *frameworks* de mesma finalidade é a simplicidade e velocidade proporcionadas pelo seu núcleo pequeno, porém altamente flexível, que permite utilizar a ferramenta para diferentes tarefas (VYSHNAVI; MALIK, 2019). Nesta monografia o seu uso foi direcionado para criação de microserviços que se comunicam para computação de algoritmos e métricas apresentados nas seções acima. A versão utilizada neste projeto é a 2.1.2.

[11] Disponível em <https://github.com/xgfs/NetLSD>
[12] Disponível em <https://github.com/bagrow/network-portrait-divergence>

2.2.5.2 Apache Kafka

Conhecido apenas como Kafka[13], este é um *middleware* de *streaming* de eventos em tempo real capaz de receber informações obtidas de várias fontes de dados, processá-las e então enviá-las para seus destinatários de forma distribuída e tolerante a falhas (WANG et al., 2015). Foi desenvolvido originalmente para a rede LinkedIn para processar dados em tempo real em poucos segundos.

Ele oferece um serviço no formato *produtor/consumidor* para processamento de *streams* de dados, esse processo é exemplificado na Figura 2.7. Um produtor pode adquirir dados de várias aplicações como *streaming* de vídeo, dados de sensores ou transferências bancárias. Então estes dados são processados como mensagens e enviados ao *Kafka Cluster* que nada mais é do que um banco de dados distribuído tolerante a falhas. Ao nível lógico, cada mensagem enviada ao *cluster* pode ser categorizada em diferentes tópicos. O *cluster* é responsável também por distribuir essas mensagens a todos os *Stream Processors* interessados e inscritos em um ou mais tópicos. Um *Stream Processor* pode assumir o papel de consumidor e também produtor, devolvendo dados processados para algum tópico específico dentro do *Kafka Cluster* (WU; SHANG; WOLTER, 2020).

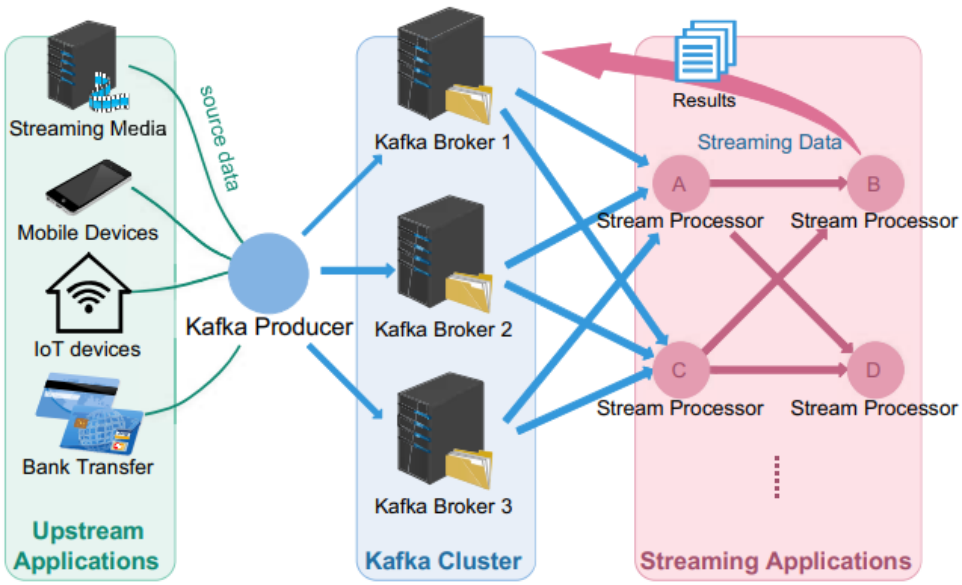

Figura 2.7 – Exemplo de uso do Apache Kafka (WU; SHANG; WOLTER, 2020).

Nesta monografia essa ferramenta foi utilizada para distribuir dados em um *pipeline* de execução, possibilitando que as séries temporais fossem processadas paralelamente. Além disso, ao separar o fluxo de dados no formato *pipeline* possibilitamos que partes do sistema sejam construídas a partir de linguagens ou *frameworks* diferentes, deixando que o desenvolvedor trabalhe com a melhor ferramenta para resolução do problema. A versão do Apache Kafka utilizada é a 3.2.0, última versão estável do projeto disponível até o momento.

[13] Disponível em <https://kafka.apache.org/>

3 Desenvolvimento

Este capítulo apresenta as características internas e externas da API apresentada nesta monografia, cujo objetivo principal é oferecer aos usuários um ambiente capaz de comparar séries temporais através de vários algoritmos e métricas. Diante disso, este capítulo está dividido da seguinte forma. Na Seção 3.1 é descrita a arquitetura interna proposta, evidenciando as estratégias empregadas para garantir paralelismo e velocidade no cálculo das comparações. Na Seção 3.2 é demonstrado como o usuário interage com o sistema.

3.1 Arquitetura interna

A API proposta nesta monografia permite que usuários enviem várias séries temporais para comparação, assim como a opção de algoritmo que será utilizado para gerar os valores de distância entre elas. Dependendo do número de séries ou a complexidade do algoritmo escolhido para realizar as comparações, o sistema pode demorar a retornar o resultado aos usuários. Diante da possível espera pelos resultados, o sistema faz o acompanhamento dos cálculos constantemente e notifica ao usuário o progresso das comparações. As duas modalidades de comparação de séries temporais oferecidas pelo sistema são:

1. Comparar por algoritmos tradicionais (Informação Mútua, Correlação de Pearson, *Dynamic Time Warping*).
2. Transformar as séries em redes complexas (DCTIF, DCSD e VG) e então comparar as redes (GCD-11, Portrait Divergence e NetLSD).

3.1.1 Processamento utilizando algoritmos tradicionais

O fluxo apresentado na Figura 3.1 expõe como os dados trafegam pela parte interna do sistema ao executar a modalidade 1, ou seja, comparar séries temporais através de algoritmos tradicionais.

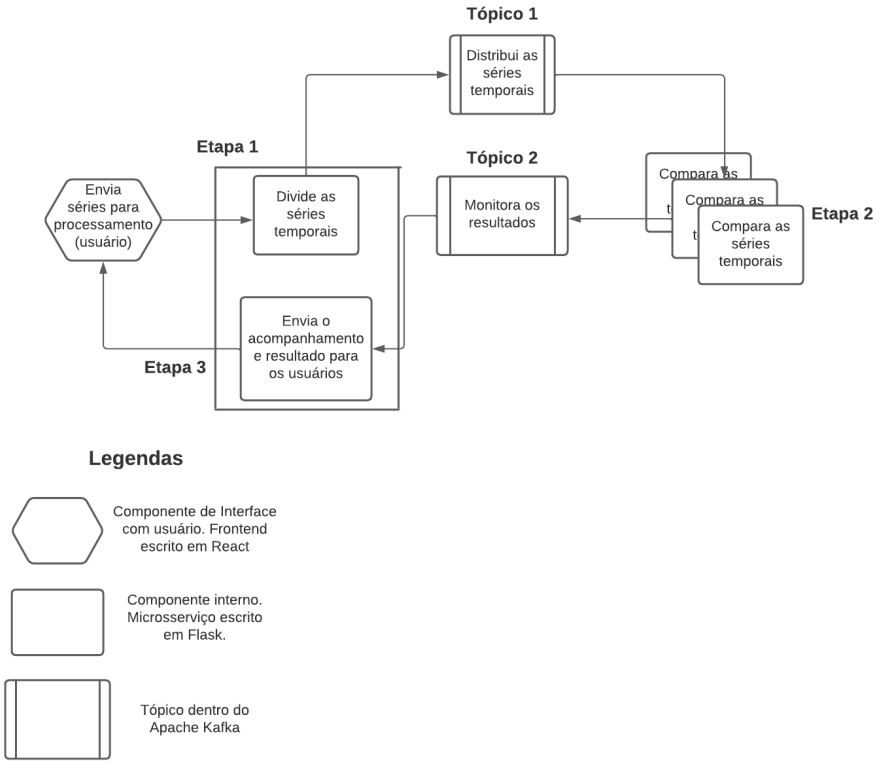

Figura 3.1 – Arquitetura interna de funcionamento para comparações utilizando algoritmos tradicionais.

- Etapa 1: Nesta primeira parte o sistema recebe todas as séries temporais enviadas pelo usuário e realiza a divisão do trabalho. Sabendo da quantidade de máquinas virtuais comparadoras de séries, a divisão é realizada da seguinte forma: Seja N o número de séries, caso N seja par então teremos $(N-1)*(N/2)$ comparações a realizar, e caso N seja ímpar teremos $((N-1)*(N/2))+N/2$. Sendo assim, se M é o número de máquinas virtuais disponíveis para comparar séries, e Z é o número de comparações a serem realizadas, cada máquina recebe Z/M comparações para realizar, sendo que em casos em que Z/M é um número não inteiro, não é possível paralelizar completamente o trabalho, de forma que pequenas diferenças no número de comparações realizadas por cada máquina podem existir.

  Os dados enviados pelo usuário são recebidos pela Etapa 1 e devem seguir o tipo de dado evidenciado na Figura 3.2, conforme descrito abaixo:

  – Na linha 2 é enviado um código de usuário, que deve ser único. Este dado servirá para que a API consiga identificar e processar séries temporais de múltiplos usuários.
  – Na linha 3 é enviado um conjunto de séries temporais, cada uma é representada através de um vetor de valores do tipo *float*.
  – Na linha 4 o usuário pode enviar o algoritmo que será usado para comparar as séries temporais.

– Na linha 5 é enviada a quantidade de séries temporais que serão processadas. Graças a este atributo, o usuário pode enviar várias séries temporais diversas vezes, tendo em vista que uma requisição pode não suportar o tamanho dos dados. A API irá esperar até que a quantidade de séries total seja enviada.

– Na linha 6 o usuário envia a posição do pacote de dados. Tendo em vista a possibilidade de envio de múltiplos pacotes simultaneamente para um mesmo conjunto de séries, no caso em que é impossível enviar todas elas somente em um pacote devido à capacidade da rede, uma posição é necessária para garantir a ordem das séries temporais no conjunto total de dados enviados.

```
1 {
2     "user_code": char[],
3     "time_series": float[[]],
4     "traditional_alg": int,
5     "len_time_series": int,
6     "position": int
7 }
```

Figura 3.2 – Tipo de dado enviado pelo usuário à API, na Etapa 1.

O balanceamento de carga de processamento é realizado nesta etapa, de forma que as máquinas recebam informações para trabalhar com um número equivalente de comparações de séries temporais.

Para tanto, considere um exemplo com 9 séries e 5 máquinas ilustrado na matriz apresentada na Figura 3.3. Note que ela possui 9 linhas e 9 colunas, assim como o número de séries temporais do conjunto (9), além disso, nas posições em que temos o valor 0, nenhuma comparação deverá ser realizada. Por outro lado, nas posições em que encontramos o valor 1 deve-se realizar a comparação entre a linha e coluna desta posição. Tomando como exemplo a linha 1, temos um total de 8 comparações a serem realizadas já que a partir da coluna 2 todos os valores encontrados são iguais a 1, ou seja, a série temporal 1 deverá ser comparada com todas as outras séries do conjunto.

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

Figura 3.3 – Matriz de comparações a serem realizadas entre 9 séries temporais.

Cada máquina terá acesso a todas as séries, e as comparações são divididas a partir do seguinte esquema demonstrado na Figura 3.4:

Máquina	Séries	Total de Comparações
1	1(8)	8
2	2(7), 9(0)	7 + 0 = 7
3	3(6), 8(1)	6 + 1 = 7
4	4(5), 7(2)	5 + 2 = 7
5	5(4), 6(3)	4 + 3 = 7

Figura 3.4 – Quantidade de comparações realizadas por cada máquina.

- A máquina 1 irá comparar a série 1 com as demais, resultando em 8 comparações (linha 1 da matriz da Figura 3.3).
- A máquina 2 irá comparar as séries 2 e 9, de forma que a série 2 será confrontada com as séries 3,4,$\cdots$,9, e a série 9 não será comparada com nenhuma outra série (linhas 2 e 9 da matriz da Figura 3.3). Serão 7 comparações no total.
- A máquina 3 irá comparar as séries 3 e 8, de forma que a série 3 será confrontada apenas com as séries de índice maior do que o dela, portanto 4, 5,$\cdots$,9. Similarmente, a série 8 será comparada apenas com a série 9. Ao todo, também 7 comparações.

De forma geral, cada série é comparada apenas com aquelas cujo índice seja maior que o dela, pois a matriz de similaridade é simétrica. Isso significa que basta, portanto, computar sua diagonal superior. Além disso, para que o balanceamento ocorra de forma equânime, é preciso que as máquinas recebam sempre uma série com valor baixo de índice e outra com o valor alto (mais para o fim). Séries do início serão comparadas mais vezes e séries do fim

menos. Em geral, quando o número de séries e máquinas são múltiplos, o balanceamento é perfeito. Quando não é o caso, a carga não será a mesma entre as máquinas, assim como demonstrado no exemplo acima.

- Tópico 1: Neste tópico são distribuídas as séries temporais e quais comparações cada máquina virtual deverá fazer. A cada máquina virtual é atribuído um valor k diferente, que além de identificá-la esse valor também vai definir quais comparações essa máquina fará. Cada máquina tem acesso a todas as séries temporais e também a quais ela deverá comparar.

  Este tópico envia para as máquinas da Etapa 2 o conjunto de dados evidenciado na Figura 3.5. A diferença deste conjunto para o conjunto anteriormente apresentado está na linha 4, onde o atributo é utilizado para cada máquina identificar quais séries temporais serão processadas. Este atributo é composto por um conjunto de vetores, em que cada posição será processada por uma máquina.

```
1 {
2     "user_code": char[],
3     "time_series": float[[]],
4     "work_order": int[[]],
5     "traditional_alg": int,
6     "len_time_series": int,
7     "position": int
8 }
```

Figura 3.5 – Tipo de dado enviado pela Etapa 1 ao Tópico 1 e recebido pelas máquinas da Etapa 2.

- Etapa 2: Nesta etapa ocorrem as comparações de séries temporais. Consiste em um conjunto de máquinas virtuais capazes de realizar comparações com os algoritmos tradicionais descritos. Cada máquina virtual recebe as séries temporais vindas do tópico 1 e partição 0 junto do trabalho que deverá ser realizado, previamente calculado na Etapa 1. Cada máquina virtual tem sua numeração k, assim elas conseguem identificar qual será o trabalho que deverão realizar.

- Tópico 2: O resultado de cada comparação é enviado para este tópico, de forma que seja possível acompanhar o progresso dos algoritmos e enviar resultados ao usuário. O progresso das comparações é enviado conforme o tipo de dado na Figura 3.6. O atributo *message* informa ao usuário a quantidade de comparações realizadas até o momento, enviando uma mensagem a cada 10000 comparações.

```
{
    "results": {
        "status": string,
        "message": string
    }
}
```

Figura 3.6 – Tipo de dado enviado ao usuário para acompanhamento.

- Etapa 3: Nesta etapa, uma conexão é realizada com o tópico 2 para acompanhar o processo de comparação e enviar para o usuário o progresso sempre que for atualizado. Por fim, quando o processo termina, este processo também é responsável por enviar ao usuário o resultado de cada posição da matriz de similaridade S_{ij}. Este conjunto é ilustrado na Figura 3.7.

```
{
    "results": {
        "line": int,
        "column": int,
        "value": float
    }
}
```

Figura 3.7 – Resultado da comparação entre duas séries enviado ao usuário.

3.1.1.1 Processamento utilizando redes complexas

A arquitetura proposta para comparação de séries por meio de algoritmos tradicionais e os baseados em redes é bem similar, porém na baseada em redes foram acrescentadas algumas etapas a mais no fluxo de comparação que irão transformar as séries em redes e depois comparar as redes. A Figura 3.8 apresenta a arquitetura correspondente.

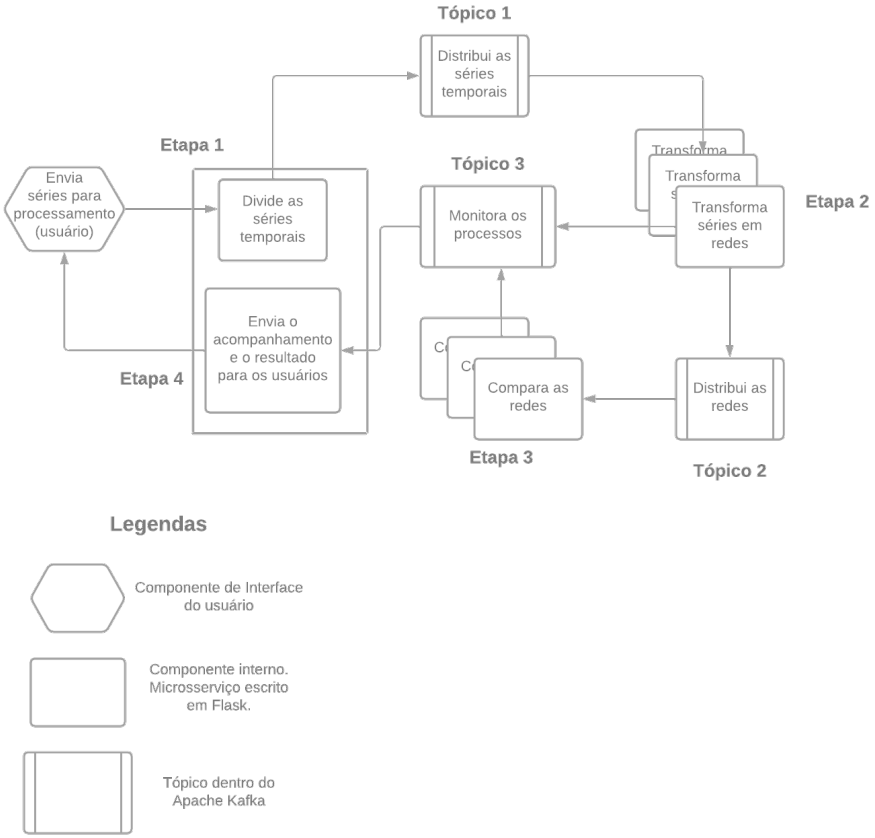

Figura 3.8 – Arquitetura interna de funcionamento para comparações utilizando redes complexas.

- Etapa 1: Nesta etapa as séries são recebidas pelo sistema e enviadas ao Tópico 1. As séries são separadas para dividir o trabalho das máquinas virtuais dedicadas a transformar séries em redes da melhor forma. Seja N o número total de séries enviadas pelo usuário e M o número total de máquinas virtuais, serão enviados para cada máquina um total de N/M séries, de forma que haverá situações em que não será possível dividir o trabalho perfeitamente entre todas as máquinas virtuais.

  Nesta etapa são enviados os mesmo atributos mostrados na Figura 3.2 acrescentando o atributo na linha 6, que irá orientar as máquinas comparadoras de redes qual algoritmo utilizar. Os atributos enviados são mostrados na Figura 3.9.

```
1  {
2      "results": {
3          "user_code": char[],
4          "time_series": float[[]],
5          "converter_alg": int,
6          "comparator_alg": int,
7          "len_time_series" int,
8          "position": int
9      }
10 }
```

Figura 3.9 – Tipo de dado enviado pelo usuário ao sistema na Etapa 1 ao comparar séries utilizando redes complexas.

- Tópico 1: Este tópico é responsável por receber as séries e entregá-las para as máquinas virtuais que vão transformar as séries em redes. Seguindo a atribuição apresentada na Figura 3.4, cada máquina transforma um conjunto de séries temporais previamente definidas na Etapa 1, esta definição será inserida por meio da variável apresentada na linha 8 no código apresentado na Figura 3.10.

```
1  {
2      "results": {
3          "user_code": char[],
4          "time_series": float[[]],
5          "converter_alg": int,
6          "comparator_alg": int,
7          "len_time_series" int,
8          "work_order": int[[]],
9          "position": int
10     }
11 }
```

Figura 3.10 – Tipo de dado enviado pela Etapa 1 para o Tópico 1.

- Etapa 2: Nesta etapa ocorrem as transformações de séries em redes. Nela estão várias máquinas virtuais capazes de realizar esta transformação executando os algoritmos previamente citados. Sendo assim, o conjunto de todas as redes geradas por cada máquina é enviado para a partição 0 no Tópico 2. Além disso, é enviado para o Tópico 3 uma mensagem com o progresso do processamento nesta etapa.

- Tópico 2: Este tópico recebe as redes geradas pela Etapa 2. Todas as redes são recebidas na partição 0. O tipo de dado enviado deste tópico para a Etapa 3 é demonstrado na Figura 3.11. Na linha 4 enviamos a rede gerada pela Etapa 2 no formato de matriz de adjacências.

```
{
    "results": {
        "user_code": char[],
        "network": float[][],
        "comparator_alg": int,
        "len_time_series" int,
        "work_order": int[[]],
    }
}
```

Figura 3.11 – Tipo de dado enviado pela Etapa 2 ao Tópico 2.

- Etapa 3: A comparação das redes ocorre nesta etapa. Após aguardar todas as redes vindas da partição 0 no Tópico 2, as máquinas virtuais se organizam conforme o exemplo demonstrado na Figura 3.4 e realizam as comparações conforme a quantidade total de máquinas disponíveis. Cada máquina virtual tem sua numeração, que pode ser um valor de 1 até o número de máquinas disponíveis. A quantidade de redes comparadas por cada uma depende da quantidade total de comparações a serem realizadas. Sendo M o número total de máquinas virtuais, cada máquina recebe um número $\sigma \in [1..M]$. Cada máquina sabe quais comparações deve realizar graças ao ordenamento realizado na Etapa 1. Os resultados das comparações são enviados ao Tópico 3 na partição 1.

- Tópico 3: Este tópico recebe uma mensagem de cada etapa para que o usuário possa ser notificado a cada avanço no fluxo descrito. Este tipo de mensagem é enviada para a partição 0, o tipo de dado enviado foi apresentado na Figura 3.6. Além disso, ele também recebe o resultado das comparações das redes.

- Etapa 4: Esta etapa possui duas funções: acompanhar o fluxo dos dados no sistema enviando ao usuário as mensagens indicando o progresso das comparações. E enviar ao usuário o resultado das comparações, o tipo de dado é o mesmo que mostramos na Figura 3.7.

A API é projetada para atender a múltiplas comparações simultaneamente, de diferentes conjuntos de dados, ao executar qualquer das modalidades apresentadas. Para garantir isso, a API reserva um *id*, identificado como o atributo *user_code*, diferente para cada conjunto de dados, de forma que este valor acompanha os dados a cada etapa nas arquiteturas apresentadas.

3.2 Iteração com o usuário

Uma vez instanciada, a API oferece um único *endpoint* que estabelece uma conexão do tipo *socket* com os usuários. Esta conexão é essencial para que o usuário possa acompanhar o processamento e receber os resultados das comparações de suas séries temporais enviadas.

Cada conjunto de dados a ser processado tem um *ID* específico e único, identificado durante a execução do *pipeline* como o atributo *user_code*, do qual o usuário deve gerar para que o sistema consiga processar diferentes conjuntos paralelamente.

O envio das séries temporais deve respeitar o tipo de dado descrito na Figura 3.2 para o processamento utilizando algoritmos tradicionais e Figura 3.9 para o processamento utilizando redes complexas. A Figura 3.12 evidencia um exemplo de uso da API utilizando a linguagem de programação *python*. Na linha 13 podemos observar que o usuário está enviando seis séries temporais para o processamento utilizando o algoritmo tradicional de número 1 (linha 19), ou seja, o *Coeficiente de Correlação de Pearson*, seguindo a ordem dos algoritmos apresentados na Seção 2.2.2.

```
1  import socketio
2
3  url = 'http://127.0.0.1:5000'
4  sio = socketio.Client()
5  sio.connect(url)
6
7  @sio.on('connect')
8  def connect():
9      sio.emit('connected')
10
11 user_message = {
12     'user_code': 'user_code',
13     'time_series': [[1,0,1,0,1,0,1,0,1,0,1],
14                     [2,1,2,1,2,1,2,1,2,1,2],
15                     [0,1,2,3,2,1,0,1,2,3,2],
16                     [1,2,3,4,5,1,2,3,4,5,1],
17                     [0,1,2,3,4,5,4,3,2,1,0],
18                     [0,1,2,3,4,5,6,7,8,9,0]],
19     'traditional_alg': 1,
20     'len_time_series': 6,
21     'position': 0
22 }
23
24 sio.emit('message', user_message)
```

Figura 3.12 – Código em *Python* de uma chamada da API

Ao definir o conjunto de séries temporais a serem enviadas para comparação, o usuário deve dividir o conjunto de séries em pequenas partes e iniciar o envio, uma vez que, a depender do tamanho do conjunto é impossível enviar tudo de uma só vez. Neste caso, a API cuidará para garantir que, mesmo enviando os dados de forma dividida, a ordem não seja perdida. Para isso o

usuário deve enviar obrigatoriamente o atributo *position*, já definido nas seções anteriores para execução das duas modalidades possíveis.

4 Experimentação Prática

Este capítulo apresenta um estudo de caso para validação do desenvolvimento e implementação da TS API. A Seção 4.1 descreve os dados utilizados para experimentação. A Seção 4.3 descreve as métricas de avaliação de desempenho utilizadas. A Seção 4.2 descreve como os experimentos são realizados, definindo a configuração de *hardware* utilizada e o planejamento da rotina de testes. Por fim, a Seção 4.4 apresenta e discute os resultados obtidos durante a realização dos testes.

4.1 Dados utilizados

Os dados utilizados para o estudo de caso da API foram extraídos da região serrana do estado do Rio de Janeiro, mais precisamente no Pico do Couto, estando a $1771,94$ metros de altitude, na Latitude de $S22°27'51''$ e Longitude de $W43°17'50''$.

As informações são apresentadas em arquivos contendo 1755 valores numéricos que indicam a intensidade do evento de precipitação em mm/h sendo distribuídas em 45 linhas e 39 colunas. Como a resolução espacial dos pontos é de $1km$, cada arquivo é a representação de uma janela temporal em uma região de $1755km^2$, sendo $45km$ distribuídos de norte a sul e $39km$ de leste a oeste. A resolução temporal é de 1 hora, compreendendo o intervalo de 10 dias entre 24 de janeiro e 2 de fevereiro de 2012 (CERON et al., 2020). Sendo assim, foram utilizadas como entrada um total de 1755 séries temporais com um total de 240 observações cada uma.

A Figura 4.1 apresenta a área da região onde os dados foram extraídos. A tabela 4.1 lista as coordenadas dos vértices da região retangular pintada de verde, em graus decimais.

Similaridade	Latitude	Longitude
Ponto inferior esquerdo	-22,433056	-42,764444
Ponto superior esquerdo	-22,0284753	-42,764444
Ponto superior direito	-22,0284753	-42,386195
Ponto inferior direito	-22,433056	-42,386195

Tabela 4.1 – Coordenadas da região de estudo. Fonte: (DINIZ, 2022).

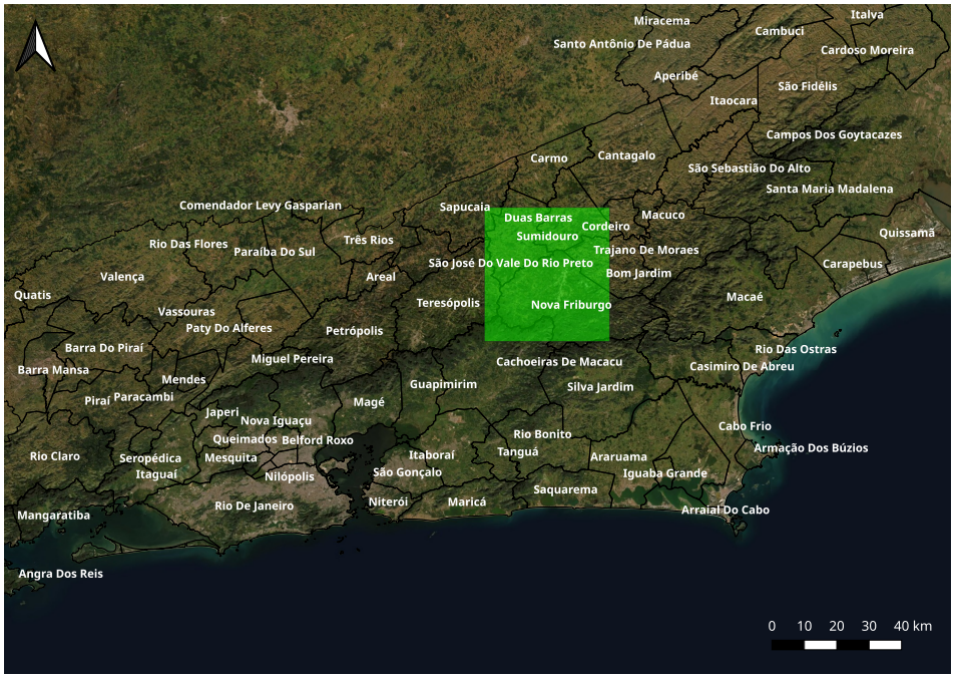

Figura 4.1 – Área da região em estudo delimitada pelo retângulo verde. Fonte: (DINIZ, 2022).

Devido a características de alguns algoritmos descritos nas seções anteriores, cada série do conjunto foi normalizada para que seus valores encontrem-se entre 0 e 1.

Abaixo, na Figura 4.2, mostramos duas séries temporais com baixa correlação: à esquerda a série temporal de índice 0, ou seja, a primeira série registrada localizada no canto superior esquerdo apresentado em verde na Figura 4.1, e à direita a série temporal de índice 756, localizada na região central.

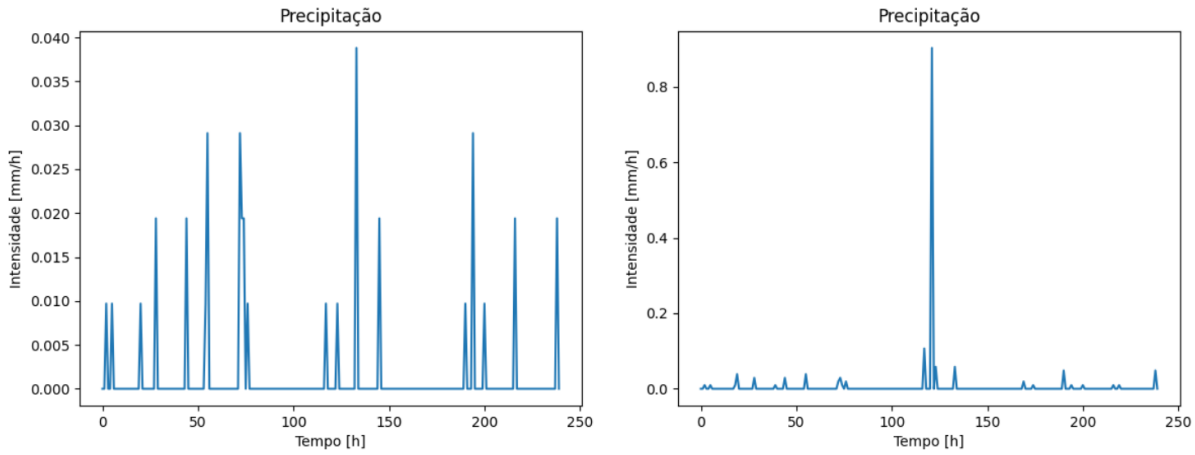

Figura 4.2 – Representação das séries temporais de índice 0 à esquerda, e índice 756 à direita.

Como visto na série à esquerda na Figura 4.2, 19 das 240 observações da série são valores diferentes de 0. O elevado número de valores nulos é uma tendência vista em todo o conjunto de séries. Para um total de 421200 observações, 384865 são referentes a valores nulos, correspondendo a $91,37\%$ do total (DINIZ, 2022).

4.2 Descrição dos Experimentos

Nesta seção é apresentado como os testes foram realizados, a configuração de cada máquina virtual utilizada, o número de máquinas virtuais em cada teste e as ferramentas utilizadas.

Para realizar a experimentação da API foram utilizadas máquinas virtuais instanciadas na plataforma *Oracle Cloud*[1]. Em cada etapa do *pipeline* descrito anteriormente nas Figuras 3.1 para algoritmos clássicos e 3.8 para algoritmos baseados em redes, teremos uma ou mais máquinas virtuais. Todas as máquinas são do tipo *VM.Standard3.Flex* que possuem *8 GB* de memória *RAM* e 1 núcleo de *CPU* físico com 2 *threads* de execução de hardware.

Já os *clusters* da ferramenta *Apache Kafka* foram instanciados na plataforma *Confluent Cloud*[2]. Foram instanciados um *cluster* para o *pipeline* baseado em algoritmos clássicos e outro para o *pipeline* baseado em redes complexas. Os dois são do tipo *basic* e permitem um armazenamento máximo de *5 TB*, uma taxa de entrada de dados de *250 MB/s* e saída de *750 MB/s*.

A rotina de testes planejada para extrair as métricas ao executar algoritmos clássicos pode ser visualizada na Tabela 4.2 em que definimos o número de máquinas virtuais em execução na etapa descrita na Seção 3.1.1.

Número de Máquinas Virtuais	Algoritmos
2	DTW, MI, Peason
3	DTW, MI, Peason
4	DTW, MI, Peason

Tabela 4.2 – Testes planejados para a execução de algoritmos clássicos.

Abaixo é definido o número de máquinas virtuais em cada teste para extração de métricas ao executar algoritmos baseados em redes complexas.

Executando o processo descrito na Etapa 2 (3.1.1.1), em que ocorre a transformação de séries temporais em redes complexas, temos a rotina de testes definida na Tabela 4.3, em que são utilizadas diferentes quantidades de máquinas virtuais para transformação de todo o conjunto de séries temporais em redes complexas, em paralelo.

Máquinas Virtuais	Algoritmos de transformação de séries temporais em redes complexas
2	DCSD, DCTIF, VG
3	DCSD, DCTIF, VG
4	DCSD, DCTIF, VG

Tabela 4.3 – Testes planejados para a execução de algoritmos de transformação de séries em redes complexas.

[1] Disponível em <https://www.oracle.com/br/cloud/>
[2] Disponível em <https://www.confluent.io/cloud>

Executando o processo descrito na Etapa 3 (3.1.1.1), em que ocorrem as comparações de redes complexas, seguimos a rotina de testes definida na Tabela 4.4, que estabelece a quantidade de máquinas virtuais disponíveis durante a comparação das redes de forma paralela.

Máquinas Virtuais	Algoritmo de Comparação
2	NetLSD
3	NetLSD
4	NetLSD

Tabela 4.4 – Testes planejados para a execução do algoritmo de comparação de redes complexas.

Os demais algoritmos apresentados para comparação de redes complexas, *GCD-11* e *Portrait Divergence*, não foram incluídos no planejamento devido ao tempo de execução apresentado ao executar as comparações para o conjunto de dados proposto, tornando-se inviável para o número de máquinas virtuais disponíveis, devido ao elevado tempo de execução.

A API fará $1.539.135$ comparações nas duas modalidades ao receber os dados propostos para teste. Já a divisão de trabalho para cada máquina virtual, a partir do esquema proposto na Figura 3.4, se dará da seguinte maneira:

- Ao utilizar apenas 2 máquinas virtuais: a máquina de número 1 fará 769.568 comparações e a máquina de número 2 fará 769.567 comparações.
- Ao utilizar 3 máquinas virtuais: a máquina de número 1 fará 513.046 comparações, a máquina 2 fará 513.045 comparações e a 3 fará 513.044 comparações.
- Empregando 4 máquinas virtuais temos as seguintes: a máquina de número 1 fará 384.785 comparações, a 2 fará 384.784, a 3 fará 384.783 e a 4 fará 384.783 comparações.

4.3 Métricas de avaliação

Utilizam-se as seguintes métricas para avaliar o desempenho completo do sistema:

- Tempo em que o usuário demora a receber os resultados das comparações a partir do envio das séries temporais. Registra-se o tempo obtido ao utilizar diferentes configurações da API, com número variado de *workers* (máquinas virtuais).
- Tempo total gasto no tráfego de dados na rede entre os consumidores e produtores.

Após realizados os testes com algoritmos clássicos comparamos os resultados obtidos com o tempo de execução da comparação dos mesmos dados em somente uma máquina executando os algoritmos sequencialmente. Já para a execução da modalidade 2, que utiliza redes complexas, não

foi possível obter o tempo da execução sequencial, pois, diante da capacidade de processamento das máquinas virtuais, o tempo ficaria muito alto, tornando-se inviável.

Além disso, é possível avaliar o comportamento do *cluster kafka* ao lidar com os dados através da extração de métricas como a quantidade de mensagens recebidas e quantidade de mensagens enviadas para ser possível dimensionar os recursos gastos durante o envio de dados dentro das duas modalidades de comparação e seus *pipelines* de execução.

4.4 Análise dos resultados

Nesta seção, são apresentados os resultados obtidos através da experimentação prática proposta para verificar o desempenho da API. Também é apresentada a análise das métricas conforme as configurações definidas para cada *pipeline* de execução.

Para a primeira modalidade de execução, em que se compara séries temporais através de algoritmos clássicos, após a realização de qualquer teste, têm-se as seguintes métricas do *cluster kafka* ao observar o número de mensagens recebidas pelos tópicos:

- No primeiro tópico (3.1.1), temos um total de 40 mensagens recebidas, enviadas diretamente pelo usuário, devido à divisão das 1755 séries em 40 partes não necessariamente de mesmo tamanho. Esta divisão foi realizada devido à necessidade de enviar todas as séries através da rede.

- Já o segundo tópico (3.1.1) recebe os resultados das comparações realizadas por cada máquina virtual, tomando como exemplo o número de comparações descrito no Item 4.2, onde a primeira máquina virtual realiza 769.568 comparações. Para enviar estes resultados o número de comparações é dividido em 80 partes. Logo teremos 80 mensagens enviadas por cada uma máquina virtual responsável por comparar as séries. Para o Item de exemplo em que temos duas máquinas virtuais, teremos um total de 160 mensagens recebidas pelo Tópico 2.

Ao observar o número de mensagens enviadas pelos tópicos na primeira modalidade de execução, têm-se os resultados:

- No primeiro tópico (3.1.1) são enviadas as séries temporais para todas as máquinas virtuais comparadoras. No caso em que temos 40 mensagens representando todo o conjunto de séries, teremos que o número de mensagens enviadas por este tópico é igual ao número de mensagens que chegaram (40), multiplicado pelo número de máquinas virtuais comparadoras que receberão estas mensagens.

- Já no segundo tópico, como apenas uma máquina estará recebendo suas mensagens, teremos que o número de mensagens recebidas (definidas em 4.4) será igual ao número de mensagens enviadas.

Para o segundo *pipeline*, em que se transforma séries em redes complexas e então as comparam, têm-se os seguintes resultados obtidos ao observar o número de mensagens recebidas pelos tópicos:

- Nó primeiro tópico tem-se que, assim como na modalidade anterior, 40 mensagens enviadas pelo usuário são recebidas.
- Já no segundo tópico são recebidas todas as redes geradas a partir das séries temporais. Logo teremos um total de 1755 mensagens, cada uma contendo uma rede.
- No terceiro tópico são recebidos todos os resultados das comparações realizadas por cada máquina virtual. Como os resultados são divididos, por cada máquina virtual, em 80 mensagens. No caso em que utilizamos duas máquinas para comparar, teremos 160 mensagens enviadas ao tópico.

Ao observar o número de mensagens enviadas, temos os seguintes resultados:

- No primeiro tópico, todas as mensagens recebidas serão enviadas para as máquinas virtuais transformadoras de séries em redes. Logo, ao chegar 40 mensagens, teremos 160 mensagens enviadas para o caso de duas máquinas virtuais.
- Já no segundo tópico serão enviadas 1755 mensagens para cada máquina virtual comparadora de redes. Logo, no caso em que temos duas máquinas virtuais, serão enviadas 3510 mensagens.
- Já no terceiro tópico, como apenas uma máquina estará recebendo suas mensagens, teremos que o número de mensagens recebidas (definidas em 4.4) será igual ao número de mensagens enviadas.

Ao executar o *pipeline* responsável por comparar redes complexas, observa-se que o número de mensagens é substancialmente maior pelo fato de haver mais tópicos, já que primeiro ocorre a transformação das séries em redes. Devido ao tamanho das redes, estas são enviadas em mensagens com somente uma rede dentro, e depois o tópico responsável envia para as máquinas virtuais comparadoras de redes, aumentando muito o número de mensagens a serem enviadas e por conseguinte temos um maior tempo de execução.

4.4.1 Tempo de execução ao utilizar algoritmos clássicos

Observando os resultados obtidos na Figura 4.3, referentes ao desempenho da API ao executar o algoritmo *DTW* em diferentes configurações, é possível concluir que:

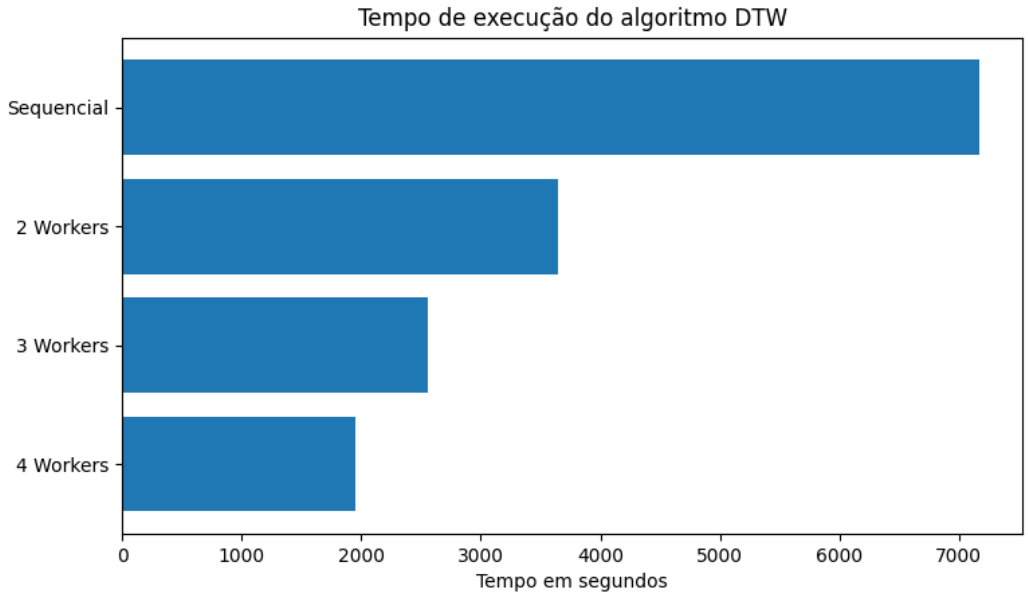

Figura 4.3 – Tempo de execução obtido ao executar o algoritmo *DTW* em diferentes configurações.

- Devido à alta complexidade do algoritmo, o tempo de execução sequencialmente é muito alto para apenas uma máquina.
- Ao utilizar a API com apenas 2 *workers* observa-se que o tempo de execução diminui 49.15% em relação à execução de forma sequencial, evidenciando que, para o algoritmo *DTW*, o uso da API já apresenta vantagens.

Da mesma forma, a partir da observação dos resultados evidenciados na Figura 4.4, referentes ao desempenho da API ao executar o algoritmo *Correlação de Pearson* em diferentes configurações, é possível concluir que:

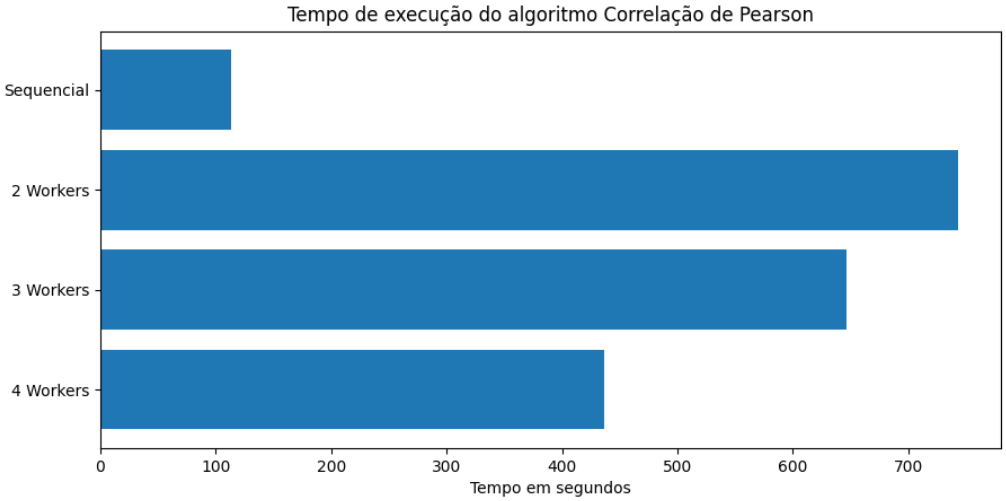

Figura 4.4 – Tempo de execução obtido ao executar o algoritmo *Correlação de Pearson* nas configurações pré-definidas para teste.

- Devido à alta velocidade de execução do algoritmo, observa-se que a execução na configuração sequencial apresenta tempo de apenas 113 segundos.
- Já a execução da API com 2 *workers*, ou seja, duas máquinas virtuais comparadoras de séries temporais, leva cerca de 743 segundos para retornar os resultados ao usuário. O tempo em que os dados trafegam durante o *pipeline* de execução foi a causa da grande diferença entre a execução sequencial e as demais.
- Mesmo aumentando o número de *workers* para 4, não é possível superar o desempenho do algoritmo sequencial.

A Figura 4.5 exibe os dados obtidos ao utilizar o algoritmo *Informação Mútua* nas diferentes configurações planejadas para testes. A partir da Figura 4.5 observa-se que:

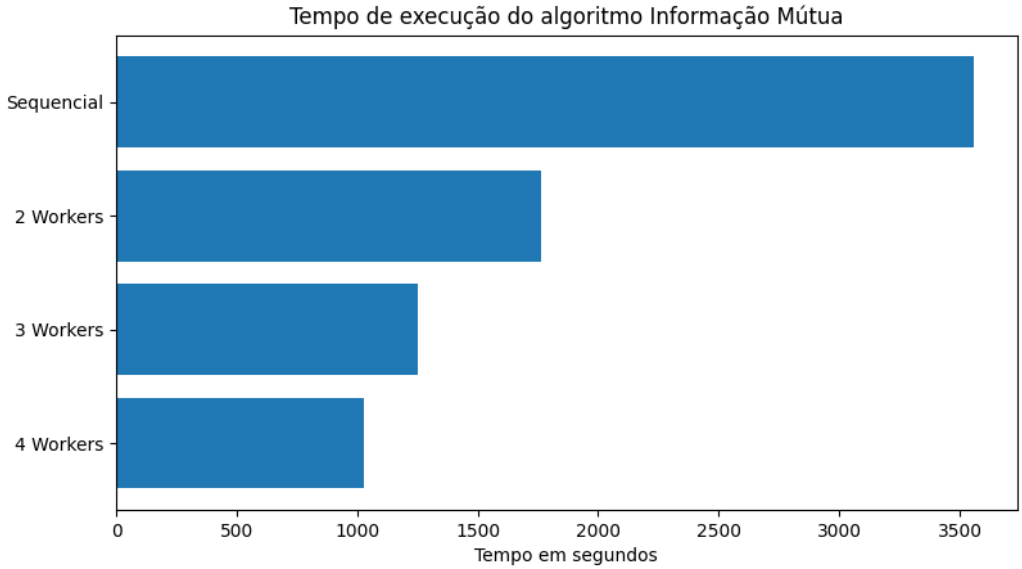

Figura 4.5 – Tempo de execução obtido ao executar o algoritmo *Informação Mútua* nas configurações pré-definidas para teste.

- O algoritmo apresenta tempo de 3558 segundos durante sua execução sequencial.
- Já a execução da API com 2 *workers*, leva cerca de 1760 segundos para retornar os resultados ao usuário. Apesar do tempo de rede contribuir bastante para o tempo de execução, podemos observar que a API é efetiva neste caso. Sendo que em entradas de dados maiores que a apresentada neste teste, a API será ainda mais eficiente.

4.4.2 Tempo de execução de algoritmos baseados em Redes Complexas

Observando os resultados obtidos na Figura 4.6, referentes ao desempenho da API ao executar o algoritmo *Visibility graph* para transformar séries temporais em redes complexas e o algoritmo *NetLSD* para comparar as redes geradas, em diferentes configurações, é possível concluir que:

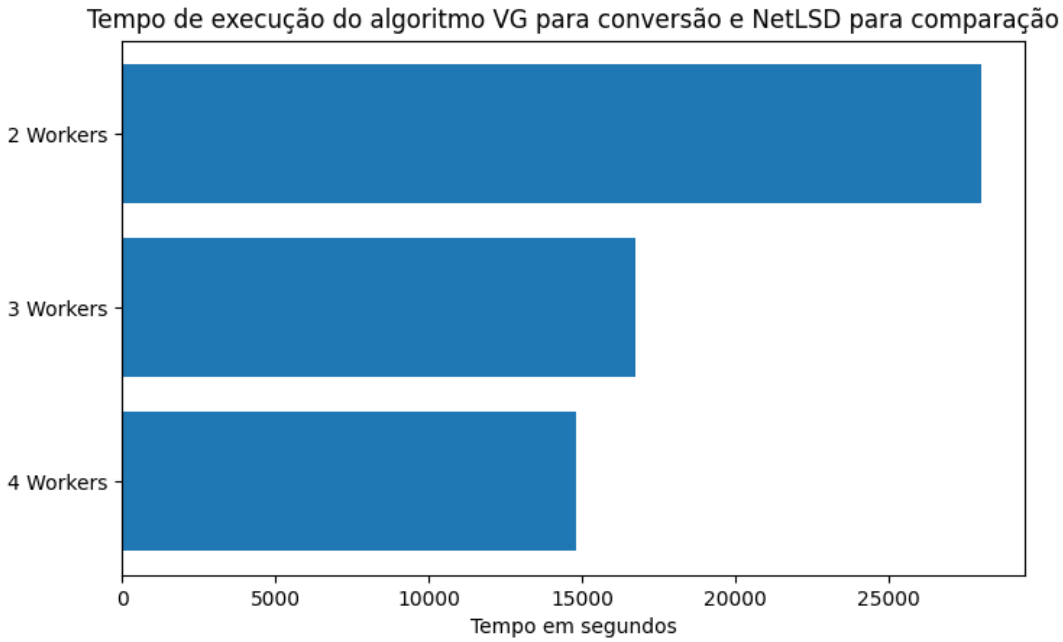

Figura 4.6 – Resultados obtidos utilizando VG para transformação e NetLSD para comparação de redes.

- O *pipeline* de execução demora cerca de 28021 segundos para retornar todos os resultados para o usuário ao utilizar 2 *workers*.
- Ao utilizar 3 máquinas virtuais *workers* observamos uma queda de 40.05% no tempo total de execução da API.

A Figura 4.7 exibe os dados obtidos ao utilizar o algoritmo *DCTIF* em conjunto com o algoritmo *NetLSD* nas diferentes configurações planejadas para testes. A partir da Figura 4.7 observa-se que:

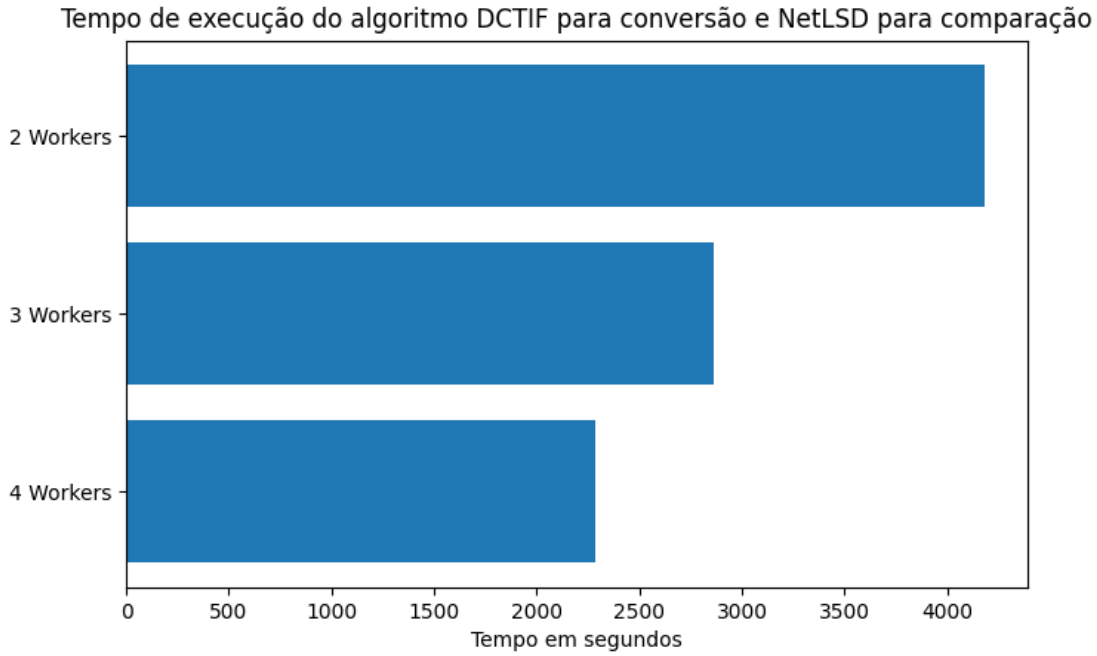

Figura 4.7 – Resultados obtidos utilizando DCTIF para transformação e NetLSD para comparação de redes.

- O *pipeline* de execução demora 4179 segundos para retornar todos os resultados para o usuário ao utilizar 2 *workers*.
- Temos uma queda de 31.58% ao comparar o tempo de execução do *pipeline* com 3 e 2 *workers*.
- E cerca de 20% de queda no tempo de execução ao comparar o tempo ao utilizar 3 e 4 *workers*.

Da mesma forma, a partir da observação dos resultados evidenciados na Figura 4.8, referentes ao desempenho da API ao executar os algoritmos *DCSD* seguido do *NetLSD* em diferentes configurações, é possível concluir que:

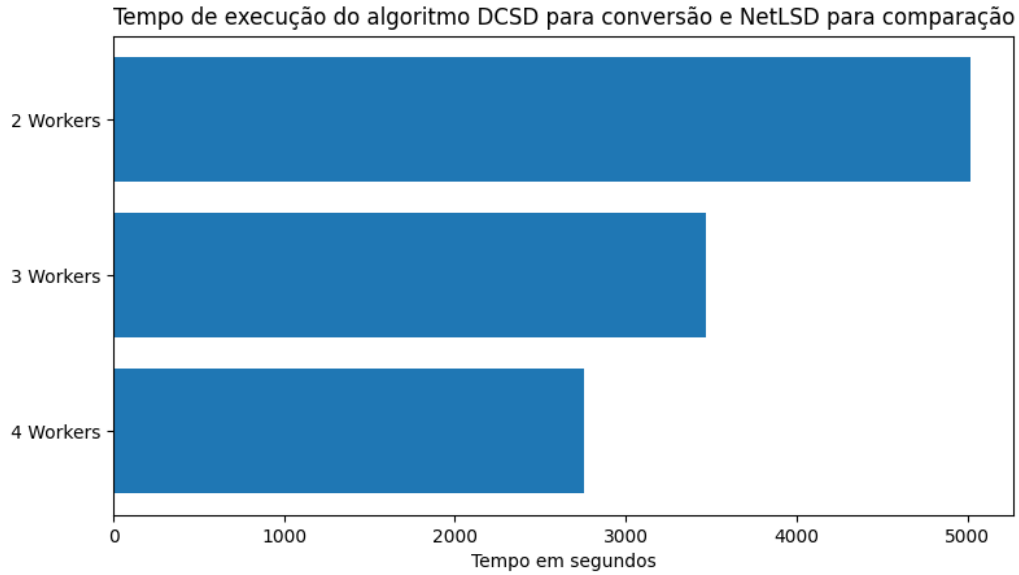

Figura 4.8 – Resultados obtidos utilizando DCSD para transformação e NetLSD para comparação de redes.

- O *pipeline* de execução demora 5020 segundos para retornar todos os resultados para o usuário ao utilizar 2 *workers*.
- Temos uma queda de 30.95% ao comparar o tempo de execução do *pipeline* com 3 e 2 *workers*.
- E cerca de 20.45% de queda no tempo de execução ao comparar o tempo ao utilizar 3 e 4 *workers*.

Podemos concluir que ao utilizar redes complexas geradas pelo algoritmo *Visibility Graph* obtemos um tempo total de execução muito maior do que o observado ao utilizar os algoritmos *DCTIF* e *DCSD*. Isso se deve ao fato de que a rede gerada pelo *VG* é muito maior que a gerada pelos demais, visto que o algoritmo cria um nó para cada observação da série temporal. Na Tabela 4.5 são apresentados valores da média da quantidade de nós e o número médio de conexões das redes geradas por cada um dos algoritmos testados, podemos notar a diferença nos valores encontrados por cada algoritmo. As redes geradas pelo algoritmo *DCTIF* possuem um número muito menor de nós ao comparar com o *VG*, já que este algoritmo cria nós sempre que encontra novos valores na série, e como demonstrado anteriormente, as séries temporais possuem muitos valores nulos.

Tabela 4.5 – Propriedades das redes geradas pelos algoritmos testados.

Algoritmo	Média do número de nós	Média do número de conexões
Visibility Graph	240	1723,92
DCTIF	6,59	14,76
DCSD	13,61	29,30

5 Considerações Finais

Neste capítulo, são apresentadas as considerações finais sobre o trabalho proposto e resultados obtidos. Na Seção 5.1 são apresentadas as conclusões obtidas e na Seção 5.2 as possibilidades de trabalhos para o futuro.

5.1 Conclusão

Em suma, esta monografia propõe uma API, denominada TS API, para comparação de séries temporais utilizando algoritmos tradicionais, bastante conhecidos na literatura, e também algoritmos baseados em redes complexas. O principal objetivo é oferecer aos usuários uma API eficaz e rápida ao efetuar estas comparações.

Devido à falta de ferramentas deste tipo na *web*, e a crescente demanda por pesquisas nesta área, principalmente na área de redes complexas, que tem crescido ao longo dos últimos anos, a aplicação TS API torna-se de suma importância para pesquisadores, que podem verificar o comportamento destes algoritmos ao receber como entrada diferentes tipos de séries temporais, e profissionais da área de análise de dados, que podem utilizar os algoritmos para simples comparação de séries temporais.

A partir dos resultados obtidos foi possível avaliar a versão da API apresentada nesta monografia. Assim como visto anteriormente, experimentos com diferentes algoritmos utilizando um número diferente de máquinas virtuais validaram a eficiência e melhores casos de uso da API. Diante disso, conclui-se que:

- Quanto aos testes utilizando os algoritmos tradicionais *DTW* e *Informação Mútua*, foi possível observar que a API provou-se eficiente em comparação com a execução de forma sequencial. Já o algoritmo *Correlação de Pearson* provou ser extremamente rápido e a execução sequencial mostrou-se superior em velocidade do que a execução paralela da API, que possui um tempo alto de transmissão de dados dentro por rede.

- Já os testes utilizando algoritmos baseados em redes complexas provaram-se eficientes em todos os casos, já que o tempo de execução dos mesmos sequencialmente é muito alto e sua captura é inviável. Além disso, a execução dos testes evidenciou características importantes dos algoritmos utilizados, como o tipo de rede gerada por cada um.

- Observou-se também que, para todos os testes, a diferença entre o tempo obtido ao usar 3 e 4 *workers* é muito menor que a obtida ao comparar o tempo entre 2 e 3 *workers*. Sendo asssim, espera-se que quanto mais *workers* utilizarmos, menor serão as diferenças encontradas, até que possamos chegar em um tempo mínimo em que observamos apenas o

tempo de deslocamento dos dados dentro da API. Esses testes podem ser realizados em trabalhos futuros.

5.2 Trabalhos Futuros

Nesta seção, são apresentadas possibilidades de trabalhos a realizar-se no futuro. Dessa forma, pretende-se:

- Realizar mais testes que comprovem a eficácia da API ao utilizar mais combinações entre algoritmos que transformam séries em redes complexas (*VG*, *DCTIF* e *DCSD*) e os que comparam redes (*GCD-11*, *Portrait Divergence* e *NetLSD*;
- Otimizar o desempenho da API em relação ao tempo gasto durante o tráfego de dados internos durante o processamento das séries temporais enviadas;
- Aprimorar as opções disponíveis para o usuário em relação a parâmetros dos algoritmos disponibilizados pela API;
- Coletar *feedback* de usuários buscando aprimorar a API aumentando a quantidade de algoritmos disponíveis para uso.

Referências

AGLIARI, E.; BARRA, A.; BARRA, O. A.; FACHECHI, A.; VENTO, L. F.; MORETTI, L. Detecting cardiac pathologies via machine learning on heart-rate variability time series and related markers. *Scientific Reports*, Nature Publishing Group, v. 10, n. 1, p. 1–18, 2020.

BAGROW, J. P.; BOLLT, E. M. An information-theoretic, all-scales approach to comparing networks. *Applied Network Science*, v. 4, n. 1, p. 45, Jul 2019. ISSN 2364-8228. Disponível em: <https://doi.org/10.1007/s41109-019-0156-x>.

BARABÁSI, A.-L.; PÓSFAI, M. *Network science*. Cambridge: Cambridge University Press, 2016.

CERON, W.; SANTOS, L. B. L.; NETO, G. D.; QUILES, M. G.; CANDIDO, O. A. Community detection in very high-resolution meteorological networks. *IEEE Geoscience and Remote Sensing Letters*, v. 17, n. 11, p. 2007–2010, 2020.

DINIZ, I. d. S. Geração de redes funcionais a partir de séries temporais de um radar meteorológico. Repositório de monografias, Universidade Federal de Ouro Preto, 2022. Disponível em: <https://www.monografias.ufop.br/handle/35400000/4219>.

EGUÍLUZ, V. M.; CHIALVO, D. R.; CECCHI, G. A.; BALIKI, M.; APKARIAN, A. V. Scale-free brain functional networks. *Phys. Rev. Lett.*, American Physical Society, v. 94, p. 018102, Jan 2005. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevLett.94.018102>.

EHLERS, R. S. Análise de séries temporais. *Laboratório de Estatística e Geoinformação. Universidade Federal do Paraná*, v. 1, p. 1–118, 2007.

ESLING, P.; AGON, C. Time-series data mining. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 45, n. 1, p. 1–34, 2012.

FERREIRA, L. N.; FERREIRA, N. C. R.; MACAU, E. E. N.; DONNER, R. V. The effect of time series distance functions on functional climate networks. *The European Physical Journal Special Topics*, v. 230, n. 14, p. 2973–2998, Oct 2021. ISSN 1951-6401. Disponível em: <https://doi.org/10.1140/epjs/s11734-021-00274-y>.

FILHO, D. B. F.; JÚNIOR, J. A. S. Desvendando os mistérios do coeficiente de correlação de pearson (r). *Revista Política Hoje*, v. 18, n. 1, p. 115–146, 2009.

FREITAS, V. L.; LACERDA, J. C.; MACAU, E. E. Complex networks approach for dynamical characterization of nonlinear systems. *International Journal of Bifurcation and Chaos*, World Scientific, v. 29, n. 13, p. 1950188, 2019.

GIERLICHS, B.; BATINA, L.; TUYLS, P.; PRENEEL, B. Mutual information analysis. In: SPRINGER. *International Workshop on Cryptographic Hardware and Embedded Systems*. [S.l.], 2008. p. 426–442.

KEOGH, E. J.; PAZZANI, M. J. Derivative dynamic time warping. In: SIAM. *Proceedings of the 2001 SIAM international conference on data mining*. [S.l.], 2001. p. 1–11.

KRASKOV, A.; STÖGBAUER, H.; GRASSBERGER, P. Estimating mutual information. *Phys. Rev. E*, American Physical Society, v. 69, p. 066138, Jun 2004. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevE.69.066138>.

KROLLNER, B.; VANSTONE, B. J.; FINNIE, G. R. et al. Financial time series forecasting with machine learning techniques: a survey. In: *ESANN*. [S.l.: s.n.], 2010.

LACASA, L.; LUQUE, B.; BALLESTEROS, F.; LUQUE, J.; NUNO, J. C. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 105, n. 13, p. 4972–4975, 2008.

LACERDA, J.; FREITAS, V.; MACAU, E. Dynamical characterization of nonlinear systems through complex networks. In: *Proceeding of the Series of the International Conference on Nonlinear Science and Complexity*. [S.l.: s.n.], 2016. p. 1–4.

LUZ, E. J. d. S.; SCHWARTZ, W. R.; CÁMARA-CHÁVEZ, G.; MENOTTI, D. Ecg-based heartbeat classification for arrhythmia detection: A survey. *Computer methods and programs in biomedicine*, Elsevier, v. 127, p. 144–164, 2016.

MELLO, C. R. d.; VIOLA, M. R. Mapeamento de chuvas intensas no estado de minas gerais. *Revista Brasileira de Ciência do Solo*, SciELO Brasil, v. 37, p. 37–44, 2013.

METABASE. *Metabase Github repository*. 2022. Disponível em: <https://github.com/metabase/metabase>.

MICROSOFT. *Azure Data Explorer*. 2022. Disponível em: <https://docs.microsoft.com/pt-br/azure/data-explorer/>.

SAKOE, H.; CHIBA, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, IEEE, v. 26, n. 1, p. 43–49, 1978.

SANTOS, B.; SÉRIO, F.; ABRANTES, S.; SÁ, F.; LOUREIRO, J.; WANZELER, C.; MARTINS, P. *Open Source Business Intelligence Tools: Metabase and Redash*. [s.n.], 2019. 467-474 p. Disponível em: <https://doi.org/10.5220/0008351704670474>.

TANTARDINI, M.; IEVA, F.; TAJOLI, L.; PICCARDI, C. Comparing methods for comparing networks. *Scientific reports*, Nature Publishing Group, v. 9, n. 1, p. 1–19, 2019.

TIMESCALE. *Timescale*. 2022. Disponível em: <https://www.timescale.com/>.

TSITSULIN, A.; MOTTIN, D.; KARRAS, P.; BRONSTEIN, A.; MÜLLER, E. Netlsd: Hearing the shape of a graph. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 2018. (KDD '18).

VYSHNAVI, V. R.; MALIK, A. Efficient way of web development using python and flask. *Int. J. Recent Res. Asp*, v. 6, n. 2, p. 16–19, 2019.

WANG, G.; KOSHY, J.; SUBRAMANIAN, S.; PARAMASIVAM, K.; ZADEH, M.; NARKHEDE, N.; RAO, J.; KREPS, J.; STEIN, J. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 8, n. 12, p. 1654–1655, 2015.

WU, H.; SHANG, Z.; WOLTER, K. Learning to reliably deliver streaming data with apache kafka. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. [S.l.: s.n.], 2020. p. 564–571.

WU, Y.-L.; AGRAWAL, D.; ABBADI, A. E. A comparison of dft and dwt based similarity search in time-series databases. In: *Proceedings of the ninth international conference on Information and knowledge management*. [S.l.: s.n.], 2000. p. 488–495.

YAVEROGLU, ; MALOD-DOGNIN, N.; DAVIS, D.; LEVNAJIć, Z.; JANJIC, V.; KARAPANDZA, R.; STOJMIROVIC, A.; PRZULJ, N. Revealing the hidden language of complex networks. *Scientific reports*, v. 4, p. 4547, 04 2014.