Explorando Centralidades Interurbanas com Algoritmos de Dijkstra e Kruskal para Otimização de Rotas de Entregas

Emanuel Vieira Tavares
Engenheria da Computação
Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG
Divinópolis, Minas Gerais, Brasil
emanuelvieiratavaresvt@gmail.com

***Resumo*—A otimização de rotas de entrega é uma preocupação frequente para a área de logística. Esse tipo de problema é discutido por meio de algumas abordagens computacionais que possibilitam soluções eficientes para o problema, como a Teoria dos Grafos, a Pesquisa Operacional, estudos de Algoritmos Genéticos e Inteligência Artificial. O presente estudo emprega uma abordagem do problema com Teoria dos Grafos, utilizando a linguagem de programação Python. O grafo é modelado partir de uma base de dados que contempla 59 cidades na região oeste da Alemanha, representando-as como vértices e as distâncias entre elas como arestas em um grafo ponderado não direcionado de forma a conectar toda a região da base de dados. O objetivo é identificar cidades estratégicas para a inserção de centros de distribuição, visando otimizar as rotas logísticas.**

A solução proposta emprega os algoritmos de Dijkstra e Kruskal. O algoritmo de Dijkstra é utilizado para calcular os menores caminhos entre pares de cidades, proporcionando uma classificação das cidades mais frequentemente visitadas nas rotas. Por outro lado, o algoritmo de Kruskal é aplicado para visualizar as conexões mais robustas entre as cidades, avaliando suas centralidades na rede interurbana. A combinação desses algoritmos resulta na seleção estratégica de três cidades ideais para a instalação de centros de distribuição, otimizando assim as rotas de entrega na região delineada pelo grafo.

***Index Terms*—Otimização de rotas, Teoria dos Grafos, algoritmo de Dijkstra, algoritmo de Kruskal**

I. Introdução

Otimizar rotas de entrega é um problema frequente do qual empresas de logística e transportadoras enfrentam, uma vez que a eficiência nesse processo impacta diretamente nos custos e na reputação destas. Desde o início dos serviçoes de entregas de correspondências, a definição estratégica de rotas tem sido uma preocupação constante para as organizações. Com a evolução dos meios de comunicação e o crescimento do comércio online, a demanda por soluções eficazes na otimização logística tornou-se ainda mais importante.

Para esse tipo de abordagem, otimização é equivalente a encontrar estratégias que possibilitam redução de tempo, custo e distância nos caminhos. Seguindo esta ideia, ao conhecer as cidades em um determinado conjunto, juntamente com as relações entre os caminhos que as interligam e as respectivas distâncias, é possível identificar cidades que exercem maior influência umas sobre as outras, a partir da relação de rotas de conexão entre si. Esta análise de rotas, com identificação de cidades estratégicas, constitui um ponto crucial para a otimização.

Este artigo aborda a modelagem computacional em grafos do problema, a partir de uma base de dados. Dado o conjunto de dados, aplica-se a Teoria dos Grafos e algoritmos específicos para explorar soluções, com a linguagem de programação Python.

A base de dados utilizada é composta por informações fundamentais: cidade de origem, cidade de destino e a distância correspondente. Cada cidade é representada como um vértice no grafo, e a existência de um caminho entre duas cidades é expressa por uma aresta. Dessa forma, o modelo de grafo fornece uma representação eficaz das rotas possíveis em uma determinada região, permitindo uma análise aprofundada da relação entre as cidades.

O foco principal deste estudo é a otimização de rotas, direcionado para identificar cidades que possuem conexões mais robustas e são frequentemente visitadas e, dessa forma, definir polos estratégicos para a inserção de centros de distribuição que otimizam as rotas. Para alcançar esse objetivo, este artigo apresenta uma abordagem utilizando o algoritmo de Kruskal, visando encontrar a árvore geradora mínima. Essa estratégia revela as cidades mais interconectadas na região. Adicionalmente, para quantificar a participação das cidades nas rotas, aplica-se o algoritmo de Dijkstra em todas as combinações de rotas possíveis, calculando assim os menores caminhos entre as cidades para encontrar possíveis centros com uma perspectiva distinta.

Esta abordagem metodológica visa contribuir para a eficiência logística, fornecendo informações valiosas sobre a seleção estratégica de centros de distribuição em um conjunto de cidades. As seções seguintes detalham uma abordagem teórica de grafos e os algoritmos de Dijkstra e Kruskal (II,III e IV), em seguida a metodologia adotada e, por fim, os resultados obtidos

e a conclusão (VII e VIII).

II. Teoria dos Grafos

A Teoria dos Grafos é um campo matemático que estuda a representação e análise de relações entre objetos distintos. [4]Um grafo é uma estrutura composta por vértices (ou nós) e arestas (ou arcos) que conectam esses vértices. Essa área de estudo fornece conceitos que modelam e resolvem uma variedade de problemas, incluindo a exploração de rotas entre cidades.

A. Conceitos Básicos em Teoria dos Grafos

Antes de entrar em detalhes específicos sobre a aplicação da Teoria dos Grafos na otimização de rotas, é importante destacar conceitos primordiais:

- **Grafo:** [4]Um grafo é uma coleção de vértices e arestas que conectam esses vértices. Se não houver direção nas arestas, ele é chamado de grafo não direcionado. Se houver direção, é chamado de grafo direcionado.
- **Vértices e Arestas:** Vértices são pontos no grafo, muitas vezes representando entidades, enquanto as arestas são linhas que conectam esses vértices, representando relações entre essas entidades.
- **Grafo Ponderado:** Um grafo ponderado atribui valores numéricos (pesos) às arestas, representando alguma medida, como distância, tempo ou custo.

B. Aplicação da Teoria dos Grafos no Estudo

A base de dados utilizada pode ser associada aplicando os conceitos de Teoria dos Grafos. Associa-se cada cidade a um vértice correspondente que é conectado por uma aresta ponderada a partir do valor da distância em quilômetros entre duas cidades. Dessa forma, aplicando a relação entre todas as cidades da base de dados é possível modelar esse agrupamento como um grafo ponderado e aplicar os algoritmos de Kruskal e Dijkstra para encontrar possíveis centros que otimizam as rotas entre o conjunto de cidades. Vale ressaltar que a fim de simplificar o problema, o grafo utilizado para modelar as relações de caminho entre as cidades é um grafo não dirigido. Para o problema, isso implica que uma cidade possui o mesmo caminho para ida e para volta, como se fisicamente a estrada fosse uma via de mão dupla.

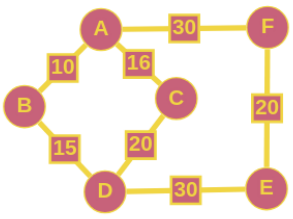

Figura 1: Caminhos entre cidades representados por um grafo.

III. Visão teórica do Algoritmo de Kruskal

O algoritmo de Kruskal é uma técnica eficaz para encontrar uma Árvore Geradora Mínima (AGM) em um grafo ponderado não direcionado. [2]Uma AGM é uma subárvore que conecta todos os vértices do grafo com arestas cujas somas de pesos são minimizadas. Uma característica fundamental do algoritmo de Kruskal é sua utilização de estruturas de dados chamadas florestas, que representam conjuntos de árvores desconexas.

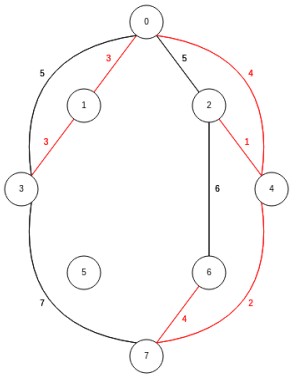

Figura 2: Grafo e a AGM em vermelho.

A. Passos do Algoritmo de Kruskal

1) Ordenação das Arestas: Todas as arestas do grafo são ordenadas em ordem não decrescente de peso.

2) Inicialização da Floresta: Uma floresta é inicializada para representar a AGM. Inicialmente, cada vértice do grafo é uma árvore isolada.

3) Iteração sobre as Arestas: As arestas são iteradas em ordem crescente de peso. Para cada aresta, verifica-se se a inclusão dela na floresta resultaria na formação de ciclos. Se não houver ciclos, a aresta é adicionada à AGM.

4) Condição de Parada: O processo é repetido até que todas as arestas sejam examinadas ou até que a floresta se torne uma única árvore, indicando que a AGM foi completamente formada.

B. Pseudocódigo

1) Funções Auxiliares:

- **ordenar_arestas(grafo):** Classifica todas as arestas do grafo com base nos pesos.
- **forma_ciclo(AGM, aresta):** Verifica se a adição da aresta à AGM formará um ciclo.
- **adicionar_aresta(AGM, aresta):** Adiciona a aresta à floresta representando a AGM.

Algorithm 1 Kruskal para Árvore Geradora Mínima

```
function KRUSKAL(grafo)
    ORDENAR_ARESTAS(grafo) {Passo 1}
    AGM ← FLORESTA() {Floresta inicialmente vazia}
    for cada aresta no grafo do
        if not FORMA_CICLO(AGM, aresta) then {Passo 3}
            ADICIONAR_ARESTA(AGM, aresta) {Adiciona à AGM}
        if AGM tem um número suficiente de arestas then {Passo 4}
            break {Árvore geradora mínima formada}
    return AGM
=0
```

C. Complexidade do Algoritmo de Kruskal

A complexidade do algoritmo de Kruskal é determinada principalmente pelas operações de ordenação das arestas e pela verificação de ciclos. Vamos examinar cada uma dessas operações.

1) Ordenação das Arestas: A ordenação das arestas é uma etapa crucial no algoritmo de Kruskal. Se utilizarmos um algoritmo de ordenação eficiente, como o QuickSort ou o MergeSort, a complexidade será $O(E \log E)$, onde E é o número de arestas no grafo.

2) Verificação de Ciclos (Estrutura de União-Busca): A verificação de ciclos é realizada utilizando uma estrutura de união-busca (Union-Find). As operações básicas desta estrutura, união e busca, têm complexidade amortizada $O(\alpha(V))$, onde $\alpha(V)$ é a função inversa de Ackermann e cresce muito lentamente.

3) Iteração sobre as Arestas: A iteração sobre todas as arestas tem uma complexidade linear $O(E)$.

4) Complexidade Total: A complexidade total do algoritmo de Kruskal é dominada pelo custo da ordenação das arestas, resultando em $O(E \log E)$ ou, em muitos casos práticos, $O(E \log V)$, onde V é o número de vértices no grafo.

D. Considerações:

No problema de otimização de rotas, o algoritmo é utilizado para visualizar as centralidades a partir da árvore geradora mínima.A aplicação e implementação do algoritmo em python é abordada nas próximas seções.

IV. Visão Teórica do Algoritmo de Dijkstra

O algoritmo de Dijkstra é utilizado para encontrar os caminhos mais curtos entre os vértices de um grafo ponderado, onde os pesos representam as distâncias entre os vértices. [3] Operando sobre um grafo direcionado e ponderado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas com pesos não negativos, o algoritmo busca o caminho mais curto de um vértice inicial s para todos os outros vértices em V.

A seguir, aborda-se uma sequência para expor o funcionamento, de forma genérica, do algoritmo.

A. Passo 1: Inicialização

Inicializar um vetor de distâncias $d[]$ com distâncias iniciais para todos os vértices, exceto o vértice inicial s, que tem distância 0. Inicializar um conjunto S que conterá os vértices cujas distâncias mais curtas a partir de s são conhecidas. Inicialmente, $S = \emptyset$.

B. Passo 2:Iteração

Enquanto S não contiver todos os vértices:

- Escolher um vértice u que não está em S e tem a menor distância $d[u]$.
- Adicionar u a S.
- Para cada vértice v adjacente a u:
  - Se $d[u] + w(uv) < d[v]$, onde $w(uv)$ é o peso da aresta de u para v, então atualizar $d[v] = d[u] + w(uv)$.

C. Passo 3: Resultado

Após a conclusão do algoritmo, o vetor $d[]$ conterá as distâncias mais curtas de s para todos os outros vértices no grafo.

D. Pseudocódigo

Algorithm 2 Dijkstra(G, s)

```
for cada vértice v em G do
    d[v] ← ∞
d[s] ← 0
S ← conjunto vazio
while S não contiver todos os vértices do
    Escolha um vértice u não em S com a menor distância d[u]
    Adicione u a S
    for cada vértice v adjacente a u do
        if d[u] + w(uv) < d[v] then
            d[v] ← d[u] + w(uv)
Retorne d =0
```

E. Complexidade de Tempo:

O algoritmo de Dijkstra possui uma complexidade de tempo de $O((V + E) \log V)$ em que V são os vértices e E as arestas do grafo.

F. Considerações:

O algoritmo de Dijkstra é eficaz para encontrar os caminhos mais curtos em grafos ponderados com arestas de peso não negativo. No problema de otimização de rotas de entregas o algoritmo é utilizado para combinar todos os menores caminhos a fim de classificar as cidades mais frequentes em todas as possíveis rotas. Dessa maneira, a complexidade do algoritmo é multiplicada pela quantidade de vezes que o algoritmo é executado para todas as combinações.

A aplicação e implementação do algoritmo em python também é abordada nas próximas seções.

V. Trabalhos Correlatos

A partir da revisão da literatura sobre o tema de otimização de rotas, observam-se formas distintas de tratar o problema. Tratando-se do tema propriamente dito, há vários trabalhos que estudam computacionalmente o problema partindo da aplicação de Algoritmos Genéticos e Pesquisa Operacional.

A exemplo, Malaquias (2006) [9] desenvolveu um trabalho que analisa a cadeia de abastecimento no setor farmacêutico, focando nos desafios enfrentados no modelo atual de entregas de medicamentos às farmácias. O objetivo principal é otimizar o processo logístico de uma distribuidora de medicamentos, reduzindo custos e atrasos nas entregas. O estudo de campo diagnóstico levou à proposta de um núcleo de roteirização de veículos. Para enfrentar a complexidade do problema, o autor escolheu a abordagem com Algoritmos Genéticos devido à sua robustez diante das características específicas da distribuição de medicamentos.

Outro estudo de otimização relacionado foi abordado por Detofeno [8], que apresenta uma metodologia para a obtenção de uma solução otimizada do problema de geração de rotas na coleta de resíduos urbanos. Para o seu desenvolvimento, foi utilizada uma combinação de técnicas da área de Pesquisa Operacional a fim de modelar o problema.

Essas abordagens não aplicam a Teoria dos Grafos, entretanto, quando efetua-se uma busca por trabalhos específicos que empregam o uso dos algoritmos de Dijkstra e Kruskal, é possível encontrar abordagens relacionadas à otimização. Sarkar (2015) [7], por exemplo, propõe um trabalho relacionado à alteração da topologia de redes de energia elétrica para atender a objetivos importantes, como restaurar conectividade, minimizar perdas de energia, manter estabilidade e maximizar a capacidade de transferência de energia. Dessa forma, o algoritmo de Kruskal é utilizado como parte integrante dessa abordagem para encontrar a árvore geradora máxima, contribuindo assim para a otimização da distribuição da rede elétrica.

Outra abordagem direcionada é tratada por Chen (2014) [6] com a aplicação do algoritmo de Dijkstra para otimização de caminhos para evacuação de emergência. O estudo aborda a questão de evacuações de emergência em situações como terremotos, furacões, incêndios, acidentes químicos, acidentes nucleares e ataques terroristas. Para modelar a evacuação de veículos em uma rede viária dinâmica, o autor emprega o algoritmo de Dijkstra para calcular o caminho mais curto em um grafo ponderado, adaptando-se aqui à situação de evacuação em casos de emergência.

Portanto, como mostrado, os estudos abordam perspectivas distintas para otimização, aplicando modelos computacionais para solução de problemas, como no problema abordado neste artigo para a otimização de rotas de entrega. Entretanto, este estudo aborda uma nova perspectiva de aplicação do algoritmo de Dijkstra e Kruskal em conjunto para explorar as centralidades e encontrar cidades que otimizam caminhos, o que não se observou na análise dos trabalhos correlatos. Dessa forma, a abordagem aqui apresentada torna-se uma contribuição valiosa para tratar o problema.

VI. Metodologia

A. Linguagem de Programação e Ferramentas

A linguagem de programação utilizada para manipulação dos dados, modelagem dos grafos e aplicação dos algoritmos foi a linguagem Python. Com as diversas bibliotecas e possibilidades de aplicações, é a ferramenta ideal para explorar o problema. As seguintes bibliotecas da linguagem foram utilizadas:

1) NumPy: NumPy é uma biblioteca para a linguagem de programação Python que suporta arrays e matrizes multidimensionais, juntamente com funções matemáticas para operar nesses arrays. Usada no problema para operações numéricas, criação de arrays e manipulação de dados.

2) Matplotlib: Matplotlib é uma biblioteca para criação de gráficos em Python. Utilizada para plotar as visualizações dos dados dos grafos.

3) NetworkX: NetworkX é uma biblioteca para a criação, manipulação e estudo da estrutura, dinâmica e funções de redes complexas. Utilizada para representar e manipular os grafos de forma mais simples, calcular árvores geradoras mínimas, realizar análises de conexões e verificar o caminho mais curto entre dois pontos. O uso dessa biblioteca é crucial para análise e desenvolvimento do problema, pois há algoritmos definidos como métodos simples de aplicar ao grafo já modelado. O método `minimum_spanning_tree` recebe o objeto de grafo da NetworkX e retorna a árvore geradora mínima. Dessa forma, é possível verificar as conexões entre as cidades.

4) Pandas: Pandas é uma biblioteca que oferece estruturas de dados e ferramentas de análise de dados para a linguagem de programação Python. Utilizada para manipulação de dados em formato de DataFrame, facilitando operações como leitura de CSV.

5) Heapq: Heapq é um módulo Python que fornece implementações de filas de prioridade usando heaps. Usado na implementação do algoritmo de Dijkstra para encontrar o menor caminho em um grafo ponderado.

6) CSV: Biblioteca padrão do Python para leitura e escrita de arquivos CSV. Usado para ler e escrever dados em formato Comma-Separated Values (CSV).

Além disso, vale ressaltar que o ambiente de desenvolvimento utilizado foi o Visual Studio Code com a extensão Jupyter Notebooks para gerenciar os códigos com a extensão .ipynb que permite mesclar linhas de código com textos formatados.

B. Base de Dados

A base de dados selecionada para este estudo tem como objetivo principal possibilitar a aplicação

abrangente de algoritmos de teoria dos grafos, com enfoque na exploração de centralidades e otimização para quaisquer conjuntos de cidades que se relacionam a partir das seguintes informações: cidade de origem, cidade de destino e a distância correspondente.

A base escolhida, denominada **"West Germany 59,"** faz parte de um conjunto de datasets intitulado **"CITIES,"** disponibilizado pelo professor Dr. John Burkardt da Florida State University. Esta base oferece informações geográficas relacionadas a 59 cidades localizadas no Oeste da Alemanha.

A estrutura da base de dados é composta por três arquivos distintos:

- **nomes.txt:** Este arquivo contém a lista dos nomes das 59 cidades associadas ao estudo, que são:.
- **coordenadas.txt:** Aqui, encontramos a lista correspondente das coordenadas geográficas de cada uma das 59 cidades, proporcionando uma dimensão espacial às análises.
- **distancias.txt:** Representando uma matriz, este arquivo estabelece as distâncias entre as cidades. Cada coordenada (x, y) na matriz indica a distância entre a cidade x e a cidade y, formando a base para análises de conectividade e otimização de rotas.

É perceptível que esse dataset base apresenta informações relevantes, mas ele não se enquadra no modelo padrão de base de dados que modela o grafo das cidades forma mais simples: (origem, destino, distância). Para obter os dados dessa forma, foi necessário manipular os arquivos disponíveis para moldar uma base no formato Comma-Separated Values (CSV), um formato de arquivo simples usado para armazenar dados tabulares (como planilhas ou bancos de dados) em um formato de texto simples, onde cada linha do arquivo representa uma linha da tabela e os valores são separados por vírgulas (ou outros delimitadores, como ponto e vírgula).

C. Modelando o Dataset para o Problema

Para modelar o grafo que representa a região Oeste da Alemanha com as relações das rotas interurbanas, foi necessário analisar detalhadamente a base de dados "CITIES" [5] e modelar um CSV que melhor representa as conexões entre as cidades.

Entretanto, como mostrado, o dataset básico [5] possui somente informações sobre as coordenadas de cada cidade e as distâncias entre elas. Para encontrar as relações interurbanas, foi necessário manipular as informações contidas e plotar um gráfico representando cada cidade a partir de sua coordenada, com as bibliotecas Numpy e Matplotlib, como mostra a Fig.3.

Com o auxílio do Google Maps para verificar a existência de rotas entre as cidades e a partir da análise das coordenadas geográficas exibidas, um arquivo CSV de relações de caminho "Relações.csv"entre as cidades foi construído. A partir do arquivo CSV com

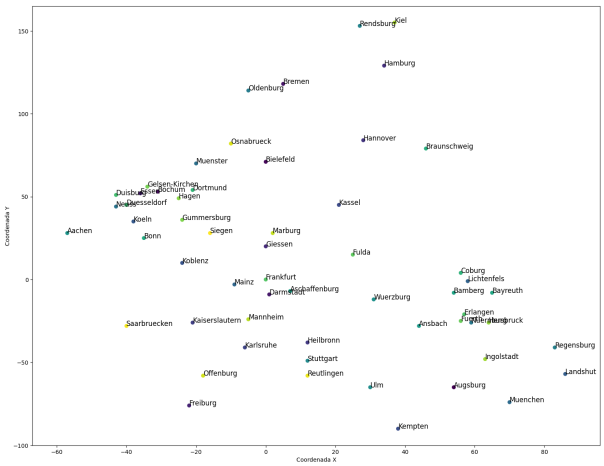

Figura 3: Cidades em suas respectivas coordenadas.

as relações entre as cidades e com a matriz de distâncias fornecida pelo dataset "CITIES", foi gerado um arquivo resultante de saída com as relações das cidades e as distâncias entre elas, ou seja, com o formato ideal para modelar o grafo.

Com todas informações de modelagem obtidas (coordenadas, relações de conexão e distância de cada caminho possível), o grafo foi modelado como um objeto da biblioteca NetworkX. Para isso, obtém-se os dados lendo o arquivo csv com a base de dados modelada e cria-se o grafo instanciando o objeto. É importante destacar que o grafo também considera as coordenadas de cada cidade. Dessa forma, é possível visualisar os vértices em posições semelhantes às reais.

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


# Caminho para o arquivo CSV n
caminho_arquivo = 'dataset/saida.csv'

# Carregar o CSV para um DataFrame do pandas
df = pd.read_csv(caminho_arquivo)

# Ler nomes das cidades
with open('59 cidades alem s/nomes.txt', 'r'
    , encoding='utf-8') as nomes_file:
    nomes = [linha.strip() for linha in
    nomes_file]

# Ler coordenadas das cidades
with open('59 cidades alem s/coordenadas.txt
    ', 'r') as coordenadas_file:
    coordenadas = [tuple(map(float, linha.
    strip().split())) for linha in
    coordenadas_file]

# Criar o dicion rio de cidades
cidades = dict(zip(nomes, coordenadas))

# Criar grafo
G = nx.Graph()

# Adicionar n s com as coordenadas
    espec ficas
for cidade, coordenada in cidades.items():
```

```
    G.add_node(cidade, pos=coordenada)

# Adicionar arestas do dataframe
for index, row in df.iterrows():
    G.add_edge(row['Origem'], row['Destino'],
     Distancia=row['Distancia'])
```

Listing 1: [1]Código para modelagem do grafo com NetworkX.

Partindo do objeto do grafo criado, uma visualização da rede de conexões entre as cidades foi obtida ao aplicar as funções de plotagem da Matplotlib:

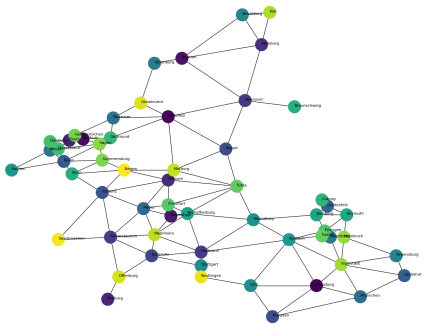

Figura 4: Grafo com a representação das rotas entre as cidades do dataset.

É importante destacar que o grafo também considera as coordenadas de cada cidade. Dessa forma, é possível visualisar os vértices em posições semelhantes às reais. Com essa visualizção da disposição das cidades do grafo, é possível observar, à grosso modo, as conexões interurbanas e analisar a possibilidade de rotas a diferentes pontos do grafo e estimar possíveis regiões interessantes para inserção de centros. Entretanto, é com a aplicação dos algoritmos que conclusões mais precisas são obtidas.

D. Aplicação do Algoritmo de Djkistra

Como já exposto, o algoritmo de Dijkstra está desempenhando o papel de encontrar o menor caminho entre todas as possíveis combinações de cidades do grafo que modela o problema.

Quando se aplica o algoritmo a todos os pares de vértices, é possível enumerar a frequência em que cada cidade é visitada, dada todas as possibilidades de caminhos. Com essa classificação de cidades mais frequentes, analisando o módulo da frequência com que elas aparecem, é possível designar um conjunto de cidades candidatas a se tornarem centros para otimização de rotas. Ou seja, as cidades que mais são visitadas possibilitam otimizar as rotas.

A função funciona mantendo uma fila de prioridade da bibilioteca Heapq, onde os nós são priorizados com base em suas distâncias acumuladas a partir do nó de origem. Durante a execução, a função explora gradualmente os nós, atualizando as distâncias acumuladas e registrando o caminho mais curto para cada nó. O resultado retornado é o menor caminho e a distância total do caminho entre os nós de origem e destino.

```
import heapq

def dijkstra(G, source, target, weight='
    Distancia'):
    # Inicializacao
    dist = {node: float('inf') for node in G.
    nodes}
    dist[source] = 0
    previous = {node: None for node in G.
    nodes}
    heap = [(0, source)]  # (dist ncia at
    o n , n )

    while heap:
        current_dist, current_node = heapq.
    heappop(heap)

        # Se o n  atual j  foi visitado,
    ignore-o
        if current_dist > dist[current_node]:
            continue

        # Relaxamento das arestas
        for neighbor, data in G[current_node
    ].items():
            edge_weight = data.get(weight, 1)
      # Peso padr o    1 se n o
    especificado
            new_dist = dist[current_node] +
    edge_weight

            if new_dist < dist[neighbor]:
                dist[neighbor] = new_dist
                previous[neighbor] =
    current_node
                heapq.heappush(heap, (
    new_dist, neighbor))

    # Reconstruir o caminho
    path = []
    current = target
    while previous[current] is not None:
        path.insert(0, current)
        current = previous[current]
    path.insert(0, source)

    return path, dist[target]
```

Listing 2: [1]Função em Python do algoritmo de Dijkstra.

E. Aplicação do Algoritmo de Kruskal

Como já exposto, o algoritmo de Dijkstra está desempenhando o papel de encontrar o menor caminho entre todas as possíveis combinações de cidades do grafo que modela o problema.

Quando se aplica o algoritmo a todos os pares de vértices, é possível enumerar a frequência em que cada cidade é visitada, dada todas as possibilidades de caminhos. Com essa classificação de cidades mais frequentes, analisando o módulo da frequência com que elas aparecem, é possível designar um conjunto de cidades candidatas a se tornarem centros para otimização de rotas. Ou seja, as cidades que mais são visitadas possibilitam otimizar as rotas.

Para isso, também seguindo os princípios teóricos abordados anteriormente, a função “kruskal” foi implementada para retornar o menor caminho a partir

de um objeto de grafo da biblioteca NetworkX como parâmetro da função.

A função implementa o algoritmo de Kruskal para encontrar a árvore geradora mínima em um grafo não direcionado e ponderado. Ela começa com cada vértice em uma árvore separada, ordena as arestas por peso e, em seguida, une as árvores conectando arestas de menor peso, garantindo que todos os vértices estejam conectados na árvore geradora mínima final. O resultado é a árvore geradora mínima do grafo de entrada.

```python
def kruskal(graph):
    # Inicializa uma floresta onde cada
    v rtice est  em uma  rvore  separada
    forest = {vertex: {vertex} for vertex in
    graph.nodes}

    # Ordena todas as arestas em ordem n o
    decrescente de peso
    edges = sorted(graph.edges(data=True),
    key=lambda x: x[2]['Distancia'])

    # Inicializa a  rvore  geradora m nima
    mst = nx.Graph()

    for edge in edges:
        u, v, weight = edge
        tree_u = next((tree for tree in
    forest.values() if u in tree), None)
        tree_v = next((tree for tree in
    forest.values() if v in tree), None)

        # Verifica se as extremidades da
    aresta est o em  rvores  diferentes
        if tree_u != tree_v:
            mst.add_edge(u, v, weight=weight[
    'Distancia'])

            # Une as  rvores  em uma  nica
     rvore
            # Une as  rvores  em uma  nica
     rvore
            tree_u.update(tree_v)
            for vertex in tree_v:
                forest[vertex] = tree_u

    return mst
```

Listing 3: [1]Função em Python do algoritmo de Kruskal.

VII. Resultados e Análises

Com base nos algoritmos enunciados, é possível estabelecer resultados relacionados às cidades mais adequadas a se tornarem centros de distribuição. Para isso, o problema foi separado em duas vertentes de solução: algoritmo de Kruskal e algoritmo de Dijkstra.

Como mostrado, o algoritmo de Kruskal explora as centralidades encontrando a árvore geradora mínima e o algoritmo de Dijkstra explora quais cidades são mais frequentes conhecendo o menor caminho entre pares de cidades. Os dois algoritmos foram aplicados no grafo e os resultados obtidos serão apresentados e analisados a seguir.

A. Dijkstra

O primeiro passo para analisar os resultados desse algoritmo foi verificar se o cálculo de menor caminho estava correto. Para isso foi adicionado um seletor para selecionar dois vértices (cidades) e retornar o menor caminho entre eles.

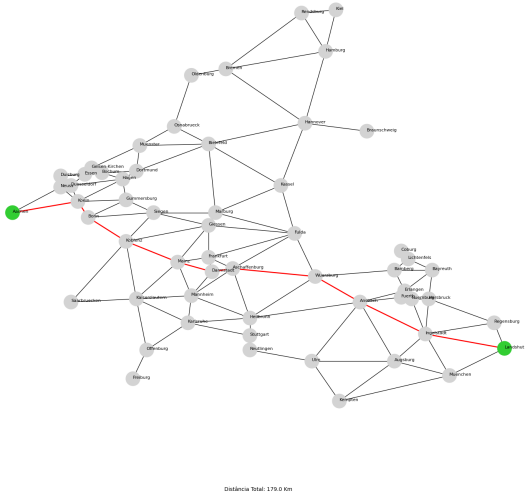

Figura 5: Menor caminho entre Aanchen e Landshut.

Com a verificação bem sucedida, o algoritmo de Dijkstra foi aplicado para todas as combinações de pares de cidades para obter um ranking das cidades que mais aparecem nos caminhos entre dois vértices do grafo.

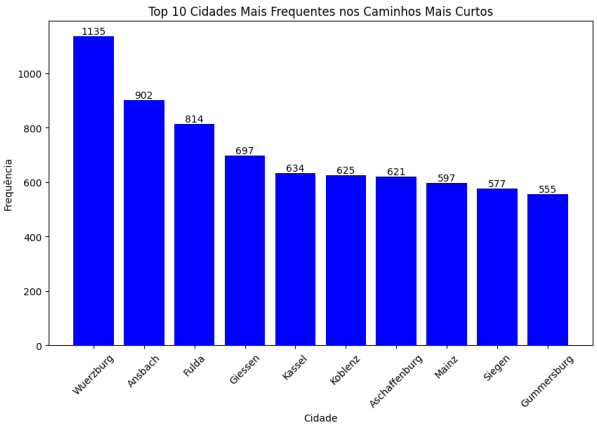

Figura 6: 10 cidades mais frequentes em rotas.

As cidades com as frequências mais significantes foram Wuerzburg (1135), Ansbach(902), Fulda(814) e Giessen(697).

Essas quatro cidades apresentam frequências de aparição em rotas elevadas em relação às demais. Isso indica que as conexões que as cidades fazem são importantes para interligar as rotas entre a região apresentada. Apesar de não estar tão próxima ao centro da região modelada pelo grafo, a cidade de Wuerzburg foi a mais frequente em rotas entre as demais. Dessa forma, pelo algoritmo de Dijkstra, essa seria uma boa cidade para se inserir um centro de distribuição considerando a sua influência geográfica na região.

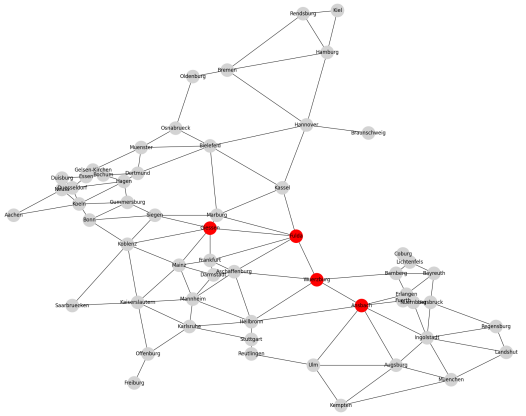

Figura 7: Cidades em destaque após aplicar Dijkstra.

B. Kruskal

Para encontrar as cidades com mais conexões indicadas por esse algoritmo, a função "kruskal"apresentada anteriormente foi aplicada ao grafo para se obter a árvore geradora mínima. A partir da AGM, observa-se as cidades que mais se conectam de modo que todos os vértices do grafo estejam interligados. Por isso, as cidades com maior número de conexões têm influência entre nas rotas da região apresentada.

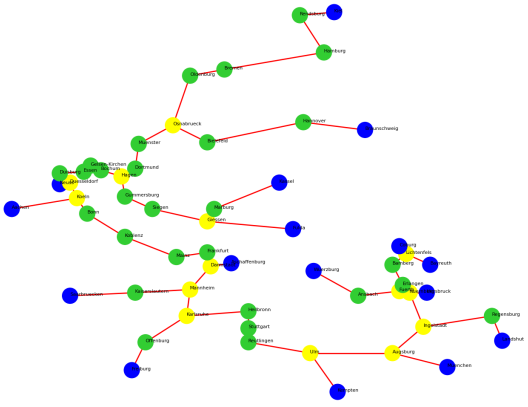

Figura 8: Árvore Geradora Mínima aplicando algoritmo de Kruskal

A imagem anterior evidencia as conexões que as cidades fazem. Em amarelo estão as cidades que fazem três conexões, em verde as que fazem duas conexões e em azul as que fazem somente uma conexão. Com a visualização da árvore após aplicar o algoritmo de Kruskal, foi perceptível que o maior número de conexões que as cidades mais influentes fazem não passou de três. Dessa forma, obteve-se uma sequência de cidades com conectividade maior em relações às demais: Osnabrueck, Duesseldorf, Koeln, Hagen, Giessen, Mannhein, Karlshuhe, Darmstadt, Ulm, Ingolstad, Augsburg, Lichtenfels, Ingolstadt e Nuemburg.

Apesar das 14 cidadedes listadas representarem somente 23.73%, o resultado obtido é inconclusivo para definir uma influencia nas rotas de forma a indicar um bom centro que otimize as rotas de entrega. Isso acontece porque nenhuma cidade se destaca significativamente entre as demais. Dessa forma, a única afirmação possível com a aplicação do algoritmo de Kruskal implementado em Python, é que essas 14 cidades apresentam centralidade maior em relação ao critério de conectividade.

Como o resultado da aplicação do algoritmo de Kruskal não apresentou uma solução clara para o problema, aplicou-se no grafo o método `minimum_spanning_tree` da biblioteca NetworkX. Apesar da documentação da biblioteca referir que essa função também implementa o algoritmo de Kruskal, o resultado da árvore geradora mínima foi diferente e surpreendetemente revelou informações significativas.

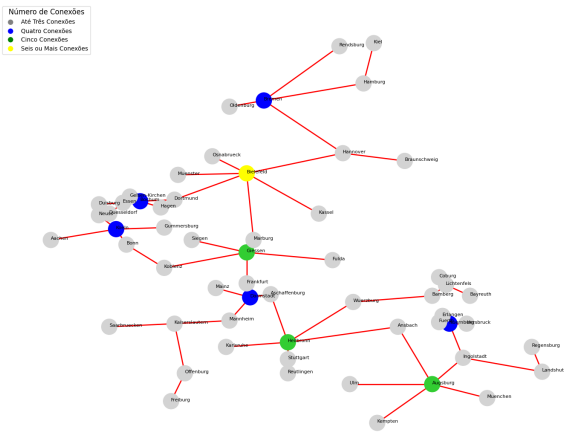

Figura 9: Árvore Geradora Mínima da NetworX

As cidades em cinza são as cidades que fazem até três conexões, as de azul fazem quatro, as de verde fazem cinco e a de amarelo faz seis conexões.

Como mostra a figura, esse método apresenta critérios de implementação distintos e retorna uma árvore geradora mínima com centralidades mais evidentes. Aplicando esse método é possível destacar quatro cidades que são influentes nas conexões com as demais: Giessen, Hielbronn e Augsburg, em verde com cinco conexões e Bielefeld em amarelo com seis conexões sendo a cidade mais influente nessa rede de centralidades indicada pela árvore geradora mínima da NetworkX.

C. Análise Geral

É perceptível que, com abordagens distintas para o problema, foi possível observar resultados variados partindo da modelagem do problema com Grafos. A aplicação do algoritmo Dijkstra foi mais consistente

em apontar com discrepancia a influência da cidade de Wuerzburg. Por outro lado, o algoritmo de Kruskal retornou uma árvore geradora mínima com conexões semelhantes para muitas cidades. Entretanto, ao aplicar o método `minimum_spanning_tree` a cidade de Bielefeld se detaca na análise de centralidades da árvore.

Dessa forma, é possível indicar Wuerzburg e Bielefeld como cidades influentes para se inserir centros de distribuição para otimizar as rotas na região. Além disso, destaca-se que após apresentar os resultados em três abordagens distintas, a cidade de Giessen esteve entre as mais influentes em todas as abordagens. Como está localizada ao centro da região, é uma importante cidade para a rede de conexão de rotas interubanas e foi frequente na análise da combinação de caminhos. Por isso, também está inclusa como solução à otimização das rotas de entregas instalando-se um centro de distribuição.

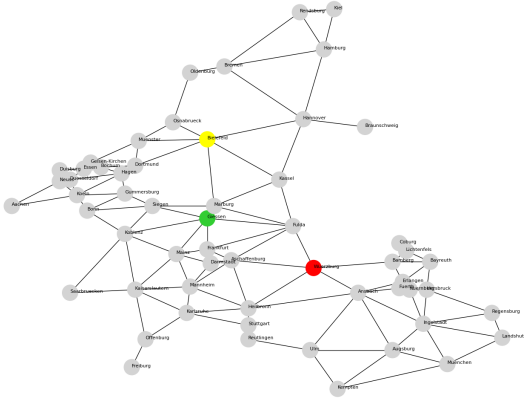

Figura 10: Centros de distribuição indicados após análise de resulatados.

A cidade de Bielefeld (amarelo) conecta as cidades da região norte, a cidade de Giessen (verde) conecta as cidades na região central e cidade de Wuerzburg (vermelho) conecta as cidades da região Sul.

Vale ressaltar que a análise realizada apresenta resultados baseados na relação geográfica entre as cidades. Os centro de distribuição escolhidos desconsideram as influencias econômicas da região.

VIII. Conclusão

Portanto, como foi abordado, para encontrar uma forma de otimizar rotas de entrega interurbanas, este estudo abordou uma modelagem para o problema utilizando Teoria dos Grafos. Os algoritmos de Dijkstra e Kruskal foram aplicados a um grafo ponderado e não direcionado, ou seja, considerando o mesmo caminho para ida e volta entre as cidades. Sendo assim, as coordenadas e conexões entre as cidades foram modeladas para encontrar posições favoráveis para instalação de centros de distribuição que otimizem as rotas logísticas.

Observou-se que os algoritmos abordam o problema sob perspectivas diferentes. Com o algoritmo de Kruskal, analisou-se as conexões das cidades a partir da árvore geradora mínima que permite a visualização das centralidades do grafo. Partindo dessa perspectiva, a cidade com as conexões mais relevantes foi a cidade de Bielefeld que conecta a região norte do grafo. Por outro lado, o algoritmo de Dijkstra abordou o problema calculando o menor caminho para todas as combinações de cidades e classificando as cidades mais visitadas em todas as possibilidades de rotas. Assim, foi possível determinar outro centro de distribuição, a cidade de Wuerzburg, mais frequente em rotas, que conecta as cidades da região sul. Com a aplicação dos dois algoritmos também foi possível observar o comportamento das rotas e conexões das demais cidades. Partindo dessa observação, notou-se que a cidade de Giessen, na região modelada pelo grafo, esteve frequente em todas as abordagens e apresenta uma localização geográfica estratégica na região central. Pelo destaque da cidade após a análise do algoritmo, Giessen foi escolhida como o terceiro centro de distribuição para otimização.

É importante destacar que a complexidade de ambos algoritmos aplicados dependem da quantidade de vértices e arestas, isto é, do número de cidades e conexões. Entretanto, a abordagem de Dijkstra utilizada depende também da quantidade de combinações entre pares de cidades. Por isso, para uma base de dados extremamente grande essa aplicação não seja tão eficiente e recomenda-se a utilização de outros algoritmos para encontrar relações de centralidades e cidades que se destacam na região. Além disso, vale ressaltar que embora aplicada a uma base de dados especificadas, almeja-se que os passos abordados possam ser aplicados para demais base de dados semelhantes. Como mencionado, basta atentar-se ao tamanho da base de dados e identificar se a solução apresentada apresentará execução satisfatória.

Diante dessa análise, conclui-se que foi possível determinar três centros de distribuição estratégicos que conseguem interligar, de forma efetiva, toda a região das 59 cidades do Oeste da Alemanha que a base de dados representa . Dessa forma, com a determinação de pontos estratégicos de partida a partir de uma análise computacional com Teoria dos Grafos, em geral, é possível reduzir tempo e custo para transportadoras no seu planejamento logístico.

Referências

[1] TAVARES, Emanuel V. *routeOtimization*. Disponível em: https://github.com/em4nuelvt/routeOtimization. Acesso em: 30 de Novembro de 2023.

[2] CORMEN, Thomas H. et al. *Introduction to Algorithms*. 3ª edição. The MIT Press, 2009. ISBN-13: 978-0262033848.

[3] SEDGEWICK, Robert; WAYNE, Kevin. *Algorithms*. 4ª edição. Addison-Wesley, 2011. ISBN-13: 978-0321573513.

[4] GROSS, Jonathan L.; YELLEN, Jay. *Graph Theory and Its Applications*. 2ª edição. CRC Press, 2005. ISBN-13: 978-1584885054.

[5] BURKARDT, Dr. John. *City Distance Datasets*. Disponível em: https://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html. 2011.

[6] CHEN, Yi-zhou, et al. *Path Optimization Study for Vehicles Evacuation Based on Dijkstra algorithm*. Nome do Jornal ou Conferência, Ano. Beijing, China: Editora.

[7] SARKAR, Dipu, et al. *Kruskal's Maximal Spanning Tree Algorithm for Optimizing Distribution Network Topology to Improve Voltage Stability*. Electric Power Components and Systems, vol. 43, no. 17, pp. 1921–1930, 2015.

[8] DETOFENO,Thober C. et al. *Otimização das Rotas de Coleta de Resíduos Urbanos, utilizando Técnicas de Pesquisa Operacional*. Universidade Federal do Paraná - Programa de Pós-Graduação em Métodos Numéricos em Engenharia. CP: 19081-CEP: 81531-990, Centro Politécnico, Jardim das Américas, Curitiba, PR.

[9] MALAQUIAS, Neli G.L. *Uso dos Algoritmos Genéticos para a Otimização de Rotas de Distribuição* Universidade Federal de Uberlândia, Faculdade de Engenharia Eletrica, Pós-graduação em Engenharia Elétrica.