UDESC UNIVERSIDADE DO ESTADO DE SANTA CATARINA

UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC

CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT

RELATÓRIO

Joinville, 2025

SUMÁRIO

1. IDENTIFICAÇÃO DA EQUIPE

Sofia Petrykowski Soares e Elane Souza de Oliveira

2. IDENTIFICAÇÃO DA TAREFA

A Tarefa 1B tem como objetivo principal elaborar um processo completo de *clustering* não supervisionado utilizando ferramentas clássicas provenientes da Teoria dos Grafos, analisando duas bases rotuladas (B1 e B2) com 150 amostras, cada uma pertencente a três classes (Tipo 1, 2 e 3). Embora o domínio aparente seja o de Machine Learning, a essência da estratégia consiste em interpretar o conjunto de dados como um grafo simples e não direcionado, no qual cada instância da base é representada como um vértice e os relacionamentos de proximidade entre vértices são traduzidos como arestas. Desta forma, a noção de cluster é mapeada diretamente para o conceito de componente conexo. Tal abordagem permite que propriedades estruturais, como conectividade e densidade de ligações, substituam técnicas tradicionais de aprendizado não supervisionado, conferindo transparência ao processo de formação dos agrupamentos.

É justamente por essa integração entre o campo de grafos e o de aprendizado não supervisionado que a tarefa se insere de forma natural no escopo da disciplina de Teoria dos Grafos. A etapa de construção do grafo, o uso de algoritmos de busca em largura (BFS) para identificação de componentes conexos, o papel do limiar de distância na configuração da conectividade global, e a análise posterior da coerência dos agrupamentos formados, derivam diretamente de conceitos fundamentais desta disciplina. O modelo assume que instâncias próximas no espaço de atributos devem ser conectadas no grafo, e que a conectividade resultante deve revelar padrões naturais de agrupamento. Assim, todo o processo de *clustering* emerge como consequência direta da topologia da estrutura criada.

3. EXPLICAÇÃO CONCEITUAL SOBRE A SOLUÇÃO FORNECIDA

3.1. Implementação do grafo

3.1.1. Definição das Estruturas de Dados

O primeiro passo consistiu na definição das estruturas, sendo fundamental para representar o grafo e os vértices e manipular os dados do arquivo CSV.

Foi criada a estrutura *DadosVertice* Figura 1, que armazena os quatro atributos numéricos de cada vértice (equivalentes a coordenadas em um espaço 4D) e o identificador do cluster ao qual pertence, determinado após a análise de conectividade.

Figura 1 - Definição da estrutura utilizada.

```
typedef struct {
    double atributos[NUM_ATRIBUTOS];
    int cluster_id;
    int label_true;
} DadosVertice;
```

Fonte: Arquivo pessoal.

Cada vértice é tratado como um ponto em um espaço de quatro dimensões, e as relações entre eles são avaliadas com base em distâncias.

3.1.2. Leitura e Pré-processamento dos Dados

A primeira parte desse sistema consiste no carregamento das bases rotuladas, implementado pela função carregar_dados_rotulados tendo como entrada arquivos CSV contendo 4 atributos e um rótulo (rotulada.csv e Dataset_rotulado_com_5_casos_de_proximidade.csv). Essa função exerce papel fundamental, pois estabelece a interface entre o formato textual das bases (.csv) e a representação interna do programa, por meio do vetor de estruturas DadosVertice. O comportamento da função é cuidadosamente projetado para lidar com variações comuns em arquivos CSV, como espaços em excesso, vírgulas supérfluas e quebras de linha irregulares. Cada linha lida é submetida ao sscanf utilizando um padrão flexível que aceita valores numéricos separados por espaços, vírgulas ou ambos. Isso é importante porque evita que a função falhe diante de pequenas inconsistências de formatação existentes entre as duas bases rotuladas disponibilizadas.

Além disso, carregar_dados_rotulados chama a função label_from_token, que converte o último campo da linha em um rótulo inteiro 1, 2 ou 3 em label_true de DadosVertice. Essa conversão não se limita a comparar strings diretamente, mas aplica uma padronização textual que transforma letras maiúsculas em minúsculas e remove caracteres de controle e pontuação irrelevantes. Essa robustez evita falhas caso a base contenha valores como “Tipo 1”, “tipo1”, “1,”, “ 3 ” ou variações semelhantes. A correta leitura e interpretação dos rótulos verdadeiros é crucial para a posterior avaliação da qualidade do clustering, especialmente na construção da

matriz de confusão. Assim, essa função representa a ponte inicial entre dados brutos e estrutura computacional, garantindo que o restante do processamento opere sobre valores confiáveis.

3.1.3. Normalização das Coordenadas e das Distâncias

Antes que qualquer relação de proximidade possa ser inferida, o programa executa uma normalização min-max sobre os atributos de cada vértice, realizada pela função normalizar_coordenadas. A necessidade dessa etapa decorre do fato de que atributos com escalas muito distintas tendem a influenciar desproporcionalmente o cálculo da distância euclidiana. Isso ocasionaria distorções significativas na conectividade do grafo, pois instâncias próximas em atributos pouco variáveis poderiam parecer distantes devido ao peso exagerado de um único atributo de larga amplitude. Ao aplicar a normalização min-max, cada atributo passa a assumir valores entre 0 e 1, preservando a proporcionalidade interna de cada dimensão, mas garantindo que todas as dimensões participem do cálculo de forma equilibrada, evitando que atributos de maior magnitude dominem o cálculo da distância euclidiana.

A normalização segue a transformação min–max, conforme a Figura 2:

Figura 2 - Equação utiliza.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Fonte: Arquivo pessoal.

Logo em seguida, o programa calcula a matriz completa de distâncias euclidianas entre todos os pares de vértices por meio de calcular_todas_distancias_euclidianas. O cálculo é simétrico e preenche uma matriz quadrada de dimensão n×n, na qual a entrada M[i][j] armazena a distância entre os pontos i e j.

Foi implementado o cálculo da distância euclidiana (DE) entre todos os pares de vértices obtidos. A distância entre dois vértices é obtida pela distância euclidiana no espaço 4D, utilizando a função calcular_distancia_euclidiana, por meio da equação apresenta na Figura 3:

Figura 3 - Equação utilizada.

$$DE(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2 + (a_4 - b_4)^2}$$

Fonte: Arquivo pessoal.

Essas distâncias são calculadas para todos os pares de vértices, resultando em uma matriz de distâncias simétrica. A função na Figura 4 recebe dois vértices “v1” e “v2” e retorna a distância entre eles como um valor do tipo *double.* Para isso fez-se o uso de funções da biblioteca math.h.

Figura 4 - Função para calcular distância euclidiana.

```
double calcular_distancia_euclidiana(DadosVertice v1, DadosVertice v2) {
    double s = 0.0;
    for (int k = 0; k < NUM_ATRIBUTOS; k++)
        s += pow(v1.atributos[k] - v2.atributos[k], 2);
    return sqrt(s);
}
```

Fonte: Arquivo pessoal.

Após esse cálculo, a função normalizar_distancias aplica nova normalização, agora sobre as distâncias, mapeando seus valores para o intervalo [0, 1]. Essa segunda normalização é igualmente essencial, pois o limiar de conexão L será interpretado nessa escala. Assim, o usuário pode especificar valores de L entre 0 e 1 independentemente da amplitude real das distâncias originais, o que fornece maior controle e consistência sobre a densidade global do grafo gerado. As distâncias são normalizadas pela função e é produzida a matriz de distâncias normalizadas (DEN), definida pela equação da Figura 5:

Figura 5 - Equação utilizada.

$$DEN_{ij} = \frac{DE_{ij} - DE_{min}}{DE_{max} - DE_{min}}$$

Fonte: Arquivo pessoal.

Onde DE_min e DE_max representam as menores e maiores distâncias entre quaisquer pares de vértices.

3.1.4. Construção do grafo a partir da Distância Normalizada

A função construir_grafo é responsável por transformar a matriz de distâncias normalizadas em uma matriz de adjacência binária, que representa diretamente o grafo não direcionado utilizado para a clusterização. Para cada par de vértices i e j, é estabelecida uma aresta se e somente se a distância normalizada entre eles for menor ou igual a um limiar L previamente determinado.

A escolha de L exerce influência profunda na topologia do grafo resultante. Para valores muito baixos, poucas arestas são criadas e o grafo se fragmenta em inúmeros componentes

conexos pequenos, frequentemente isolados. Para valores muito altos, ocorre o oposto, o grafo tende a se tornar conexo produzindo um único componente gigante incapaz de refletir estrutura interna dos dados. Portanto, definir um valor apropriado de L é um problema não trivial, e sua escolha condiciona diretamente o sucesso do procedimento de clustering baseado em componentes conexos. É por essa razão que o código oferece uma solução sistemática e automatizada para a determinação do L ideal. A função está representada na Figura 6:

Figura 6 - Função para construir o gráfico.

```
void construir_grafo(double DEN[NUM_VERTICES][NUM_VERTICES],
    int A[NUM_VERTICES][NUM_VERTICES], double L, int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            A[i][j] = (DEN[i][j] <= L && i != j) ? 1 : 0;
}
```

Fonte: Arquivo pessoal.

Esse método cria a matriz de adjacência A[N][N]: A[i][j]=1 se DEN[i][j] <= L

3.2.Determinação do Limiar Ótimo (L*)

A função escolher_L_otimo constitui um dos pontos mais sofisticados e fundamentais de toda a implementação. Ela realiza uma busca parametrizada no intervalo [0.00, 0.40], com incremento controlado (0.0001), para encontrar o valor de L que que gere 3 grandes clusters de tamanho o mais próximo possível do tamanho alvo, ou seja, três grandes componentes conexos equilibrados em tamanho. Para cada L, a função calcula todos os componentes conexos, ordena seus tamanhos e soma os desvios absolutos do Top-3 em relação ao alvo, conforme a equação da Figura 7:

Figura 7- Equação do Desvio

$$\text{Desvio} = \sum_{i=1}^{3} |\text{Tamanho}_i - \text{Alvo}|$$

Fonte: Arquivo pessoal.

O equilíbrio é definido em relação ao valor alvo n/3, onde n é o número total de vértices. A função, portanto, tenta aproximar a configuração do grafo da estrutura que naturalmente se esperaria de um conjunto contendo três grupos bem formados.

O procedimento é o seguinte: para cada valor de L testado, o grafo correspondente é construído, os componentes conexos são identificados e seus tamanhos são ordenados em

ordem decrescente. Em seguida, calcula-se a soma das diferenças absolutas entre os tamanhos dos três maiores componentes e o tamanho ideal. Esse valor representa o desvio estrutural em relação ao cenário desejado. O L que minimiza esse desvio é selecionado como L*, em caso de empate, o menor L é preferido (grafo mais esparso e com maior coesão interna). Essa estratégia evita arbitrariedades e automatiza a seleção de um parâmetro crítico, conferindo maior rigor matemático e robustez ao processo de clusterização.

3.3.Identificação dos componentes conexos via BFS

Com o grafo construído, a próxima etapa consiste em detectar seus componentes conexos, tarefa realizada pelas funções bfs_componente e analisar_componentes. A primeira implementa a busca em largura a partir de um vértice inicial e marca todos os vértices descobertos pertencentes ao mesmo componente. A segunda orquestra a varredura completa do grafo, garantindo que todos os vértices sejam analisados e cada componente receba um identificador único.

3.3.1. Análise Busca em Largura

A **Busca em Largura (Breadth-First Search - BFS)** é ideal para este tipo de análise por dois motivos principais:

1) **Garantia de Descoberta:** O BFS garante que todos os nós acessíveis a partir de um vértice inicial sejam explorados **antes** de se mover para um novo componente não explorado.
2) **Identificação e Tamanho:** Adaptamos o BFS para, a partir de um nó ainda não visitado, marcá-lo e todos os seus vizinhos conectados com um novo **ID de Cluster**. Ao mesmo tempo, ele conta o número exato de vértices encontrados durante a busca, fornecendo o **tamanho do componente conexo**.

As funções bfs_componente e analisar_componentes servem para identificar e contar os componentes conexos (clusters) no grafo e depois armazenar o ID do cluster no vetor de dados e o tamanho de cada cluster. Algoritmo de percorre um componente conexo do grafo A, atribuindo o mesmo cid (Cluster ID) a todos os vértices visitados. O número total de vértices processados e rotulados durante a busca é retornado via ponteiro tam, conforme a visualização na Figura 8:

Figura 8 - Função de Busca em Largura

```c
void bfs_componente(int A[NUM_VERTICES][NUM_VERTICES], int *vis, DadosVertice dados[NUM_VERTICES],
    int v0, int cid, int *tam, int n) {
    int fila[NUM_VERTICES], ini = 0, fim = 0, cont = 0;
    fila[fim++] = v0; vis[v0] = 1; dados[v0].cluster_id = cid; cont++;
    while (ini < fim) {
        int u = fila[ini++];
        for (int v = 0; v < n; v++)
            if (A[u][v] && !vis[v]) {
                vis[v] = 1; dados[v].cluster_id = cid; fila[fim++] = v; cont++;
            }
    }
    *tam = cont;
}
```

Fonte: Arquivo pessoal.

Depois disso, passa pela função analisar_componentes_ iterando sobre o conjunto de vértices e, para cada vértice ainda não agrupado, iniciar uma nova BFS. Cada BFS bem-sucedida descobre um novo e distinto *cluster*. O loop termina após verificar todos os vértices. O valor final de cid é o número total de *clusters*, que é retornado. Isso é visualizado pela Figura 9:

Figura 9 - Função de Analisar os Componentes

```c
int analisar_componentes(int A[NUM_VERTICES][NUM_VERTICES],
    DadosVertice dados[NUM_VERTICES], int *tamanhos, int n) {
    int vis[NUM_VERTICES] = {0}, cid = 0;
    for (int i = 0; i < n; i++)
        if (!vis[i]) {
            int t = 0;
            bfs_componente(A, vis, dados, i, cid, &t, n);
            tamanhos[cid++] = t;
        }
    return cid;
}
```

Fonte: Arquivo pessoal.

3.3.2. Cálculo dos Centros de Gravidade dos Clusters

Uma vez determinados os clusters, a função calcular_centros calcula o centro geométrico de cada componente conexo. O centro de um cluster é obtido como a média simples de cada atributo entre os vértices pertencentes àquele grupo. Embora os componentes conexos tenham sido produzidos sem considerar diretamente atributos numéricos, seus centros fornecem uma representação vetorial compacta da posição média do grupo no espaço de atributos normalizados.

Essa etapa é essencial porque fornece uma base para a reclassificação de clusters pequenos e para a reorganização final dos clusters em apenas três grupos principais. Assim, a função não serve apenas a fins descritivos, mas também desempenha um papel operacional no ajuste final dos agrupamentos.

3.3.3. Reclassificação e Consolidação dos Três Clusters Principais

A função reclassificar_e_votar executa a etapa mais estratégica do processo: consolidar a clusterização em três grandes grupos e reclassificar automaticamente todos os clusters menores. Primeiramente, a função identifica os três maiores componentes conexos encontrados pela BFS e obtém seus respectivos centros. Em seguida, percorre todos os vértices pertencentes a componentes menores e os realoca ao cluster cujo centro está mais próximo em termos da distância euclidiana dos atributos.

Essa estratégia de realocação tem justificativa teórica e prática. Em grafos gerados por limiares de distância, clusters pequenos podem surgir devido a ruídos, pontos de baixa densidade ou regiões da base onde as distâncias normalizadas estão acima do limiar estabelecido. Relacionar esses vértices aos centros dos três clusters principais aproxima o sistema de modelos típicos de clustering baseados em centroides, ao mesmo tempo em que aproveita a estrutura do grafo. Isso confere ao sistema uma robustez híbrida, combinando a estabilidade topológica dos componentes conexos com a coerência geométrica dos centros de massa.

3.3.4. Associação Ótima Cluster = Classe Verdadeira e Matriz de Confusão

Mesmo após a reclassificação, resta um problema: os clusters obtidos pela estratégia anterior não possuem necessariamente correspondência direta com os rótulos verdadeiros tipo1, tipo2 e tipo3. Para resolver isso, a função rotulacao_por_votacao monta uma tabela de frequências que indica, para cada cluster, quantos elementos de cada classe verdadeira ele contém. É feito um mapeamento guloso onde cada cluster recebe o rótulo mais frequente que ainda não foi atribuído, evitando conflitos e garantindo correspondência 1:1.

Esse processo é inteligente porque evita erros de interpretação que ocorreriam se simplesmente associássemos, por exemplo, o cluster 0 ao tipo1, cluster 1 ao tipo2, e assim por diante. Em vez disso, utiliza-se uma abordagem baseada em máxima frequência que garante o

melhor casamento possível entre clusters e rótulos verdadeiros, aumentando a precisão da matriz de confusão.

A matriz de confusão é salva juntamente com métricas clássicas como acurácia, precisão macro, recall macro e F1-score macro. Essas métricas fornecem uma avaliação quantitativa da qualidade do agrupamento final, permitindo identificar se algum dos clusters apresenta grande número de falsos positivos ou falsos negativos.

3.3.5. Tridimensional e Geração de Histogramas

O código em Python foi desenvolvido com o objetivo de realizar análises gráficas avançadas envolvendo visualização tridimensional de grafos, histograma de distribuição de componentes conexos e representação da matriz de confusão resultante da etapa de reclassificação por votação. O programa utiliza bibliotecas científicas consolidadas, como NumPy, Matplotlib, NetworkX e Pandas, para manipulação de dados, geração de grafos e construção de visualizações que auxiliam a interpretar a estrutura dos agrupamentos antes e depois da reclassificação. Além disso, o script é capaz de processar automaticamente múltiplas bases de dados (B1 e B2), identificando os arquivos corretos a partir de padrões nos nomes, e gerando todos os gráficos de forma automatizada para cada limiar L encontrado.

1) **O código faz uso das seguintes bibliotecas:**

   a) **matplotlib**: utilizada para geração de gráficos 2D e 3D, permitindo visualizar os grafos e histogramas com alta qualidade gráfica.
   b) **mpl_toolkits.mplot3d**: fornece suporte à criação de gráficos tridimensionais.
   c) **numpy**: responsável pela manipulação eficiente de vetores e matrizes numéricas.
   d) **networkx**: utilizada para criar grafos a partir de matrizes de adjacência e manipular suas arestas.
   e) **pandas**: empregada para leitura estruturada de arquivos CSV, especialmente da matriz de confusão.
   f) **glob** e **re**: permitem localizar arquivos automaticamente e extrair o valor de L de seus nomes.
   g) **os**: usado para verificar e criar diretórios de saída.
   h) **random**: utilizado para gerar cores aleatórias no modo K_TOTAL.

Figura 10– Bibliotecas utilizadas em Python.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from mpl_toolkits.mplot3d import Axes3D
4  import networkx as nx
5  import pandas as pd
6  import os
7  import random
8  import glob
9  import re
```

Fonte: Arquivo pessoal.

2) O programa está dividido em três partes principais:

a) **Definição das Funções Auxiliares:** Funções responsáveis por gerar cores fixas, plotar grafos em três dimensões, construir histogramas e visualizar a matriz de confusão.

b) **Processamento Principal (função main):** Faz a busca automática dos arquivos correspondentes à base de dados, carrega matrizes e coordenadas e chama as funções de visualização.

c) **Saída de Resultados:** Geração e salvamento automático de gráficos, histogramas e matriz de confusão em diretórios específicos dentro da pasta *img/*.

3) Função get_cores_grupos()

a) Esta função retorna uma paleta de cores fixas utilizadas exclusivamente quando o script está operando no modo FINAL, ou seja, após o processo de reclassificação por votação para K=3 grupos.

b) Essas cores são:

i) Azul (#1f77b4)

ii) Laranja (#ff7f0e)

iii) Verde (#2ca02c)

c) Elas são empregadas para manter consistência visual entre diferentes execuções e diferentes valores de L.

Figura 11 - Função get_cores_grupos()

```python
# === Paleta de cores consistente ===
def get_cores_grupos():
    """Cores fixas para os grupos principais (1,2,3)."""
    return ['#1f77b4', '#ff7f0e', '#2ca02c']  # Azul, Laranja, Verde
```

Fonte: Arquivo pessoal.

4) **Função plot_grafo_3d(...)**

   a) Esta é uma das funções principais do script. Ela é responsável por gerar a visualização tridimensional do grafo, destacando os clusters por cor.

   b) Etapas executadas pela função:

      i) Conversão da matriz de adjacência em um grafo: Utiliza a função nx.from_numpy_array(matriz_adj).

      ii) Criação da figura em 3D: Usando fig = plt.figure() e ax = fig.add_subplot(111, projection='3d').

      iii) Colorização dos clusters:

         (1) No modo FINAL, utiliza as cores fixas definidas em get_cores_grupos().

         (2) No modo K_TOTAL, gera cores aleatórias para cada grupo detectado.

      iv) Plotagem dos vértices: Cada vértice é mostrado como ponto 3D (X, Y, Z), com borda preta e leve transparência.

      v) Desenho das arestas: As arestas são desenhadas com opacidade baixa (α = 0.08), para não poluir a visualização.

5) **Função gerar_histograma(tamanhos, L_str, L_float)**

   a) Esta função produz o histograma dos tamanhos dos componentes conexos, resultantes da etapa de clusterização inicial (K_TOTAL).

   b) Principais etapas:

      i) Filtra tamanhos válidos: Remove valores vazios ou zero.

      ii) Define bins inteiros: Para que cada barra represente um tamanho exato de componente.

      iii) Gera o histograma: Utiliza plt.hist(...) com bordas destacadas e rótulos.

6) **Função plot_confusao(...)**

a) Esta função representa graficamente a matriz de confusão produzida pelo seu código em C.
b) Etapas da função:
    i) Leitura do CSV com Pandas.
    ii) Conversão da matriz em formato NumPy.
    iii) Desenho da matriz com imshow() usando o mapa de cores "Greys".
    iv) Escrita dos valores nas células com contraste automático (texto branco ou preto).
    v) Rótulos nos eixos:
        (1) Predito (votação)
        (2) Verdadeiro (rotulado)

7) Bloco Principal de Execução

a) A função main() automatiza o processamento para duas bases distintas: B1 e B2.
b) Etapas executadas:
    i) Criar diretórios de saída automaticamente:
        (1) img/grafos3D
        (2) img/histogramas
        (3) img/matriz_confusao
    ii) Localizar os arquivos corretos de coordenadas finais
        (1) Usa glob para encontrar arquivos:
            (a) ./coordenadas/coord_FINAL_{TAG}_L_*.csv
    iii) Extrair o valor de L do nome do arquivo
        (1) Por meio de expressão regular (regex).
    iv) Definir arquivos esperados:
        (1) Matriz de adjacência
        (2) Tamanhos dos componentes
        (3) Coordenadas 3D
        (4) Clusterização final e original
        (5) Matriz de confusão
    v) Carregar arquivos e gerar visualizações, nesta ordem:
        (1) Histograma dos componentes
        (2) Grafo FINAL (reclassificado)

(3) Grafo original (K_TOTAL)

(4) Matriz de confusão

c) Casos de erro são tratados individualmente, permitindo que o restante continue sendo processado.

8) Saídas Geradas

a) Grafos 3D (/img/grafos3D/): Representações tridimensionais com cores distintas para cada cluster, tanto antes quanto depois da reclassificação.

b) Histogramas (/img/histogramas/): Diagramas de barras refletindo a distribuição dos tamanhos dos clusters (K_TOTAL).

c) Matrizes de Confusão (/img/matriz_confusao/): Permitem avaliar a concordância entre rótulos verdadeiros e os clusters finais reclassificados.

4. INSTRUÇÕES SOBRE A COMPILAÇÃO E EXECUÇÃO

O projeto foi desenvolvido utilizando a IDE Visual Studio Code, com suporte à linguagem C configurado para ambos os sistemas operacionais Windows e Linux (Ubuntu) . O trabalho possui duas etapas de execução: a compilação e execução do código C, seguida pela execução do script Python para visualização.

4.1.Execução no Linux/Ubuntu

Pré-requisitos: Certifique-se de que o **GCC** (GNU Compiler Collection) e o **Python 3** estejam instalados. É de suma importância que você esteja com o Prompt de Comando ou o Terminal aberto na pasta "**src"**.

1) **Organização dos Arquivos**

a) Certifique-se de que main.c, rotulada.csv e Dataset_rotulado_com_5_casos_de_proximidade.csv estão na pasta "src".

2) **Compilação do Código C**

a) Use o GCC para compilar os arquivos C, incluindo a biblioteca matemática (-lm):

i) gcc main.c -o programa_grafo -lm

3) **Execução do Programa C**

a) Execute o arquivo binário gerado:

i) ./programa_grafo

4) **Execução do Script Python**
    a) Baixe as bibliotecas necessárias:
        i) pip install numpy matplotlib networkx pandas
    b) Execute o script Python para gerar as visualizações e histogramas a partir dos arquivos CSV criados:
        i) python3 visualizar_grafo.py

4.2.Execução no Windows

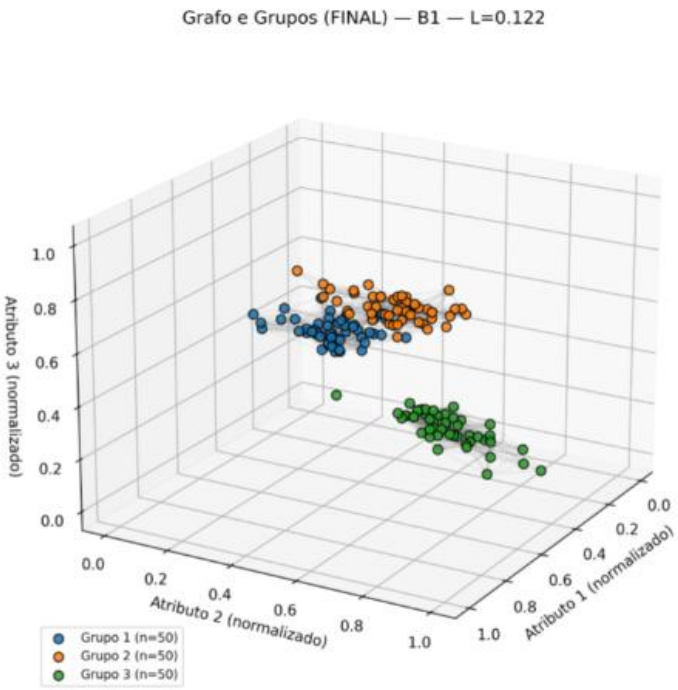
Pré-requisitos: Certifique-se de que o **GCC** (GNU Compiler Collection) e o **Python 3** estejam instalados. É de suma importância que você esteja com o Prompt de Comando ou o Terminal aberto na pasta "**src".**

1) **Compilação do Código C**
    a) Abra o Prompt de Comando ou o Terminal na pasta "**src"** e use o mesmo comando de compilação:
        i) gcc main.c -o programa_grafo -lm
2) **Execução do Programa C**
    a) Execute o binário gerado:
        i) ./programa_grafo
3) **Execução do Script Python**
    a) Baixe as bibliotecas necessárias:
        i) pip install numpy matplotlib networkx pandas
    b) Execute o script Python.
        i) python visualizar_grafo.py

5. RESULTADOS E ANÁLISE

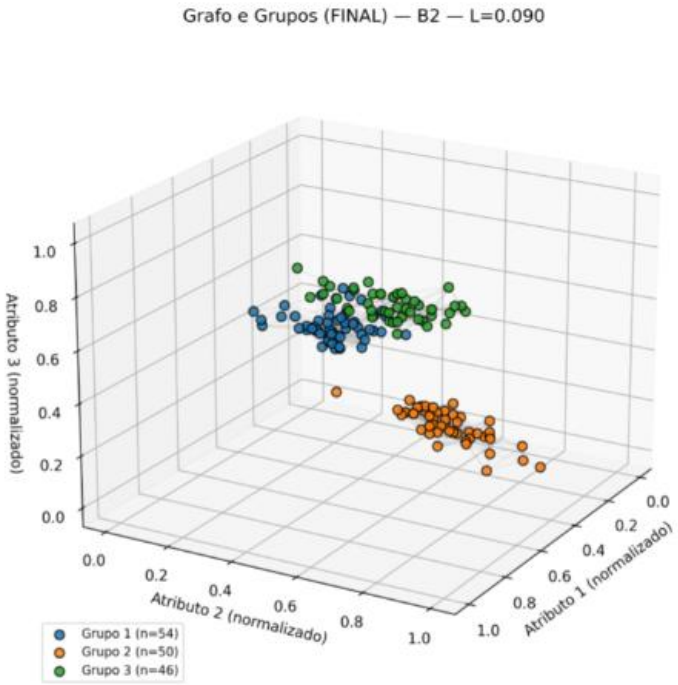
5.1.Grafos 3d

Figura 12 – Grafo e Grupos finais de B1



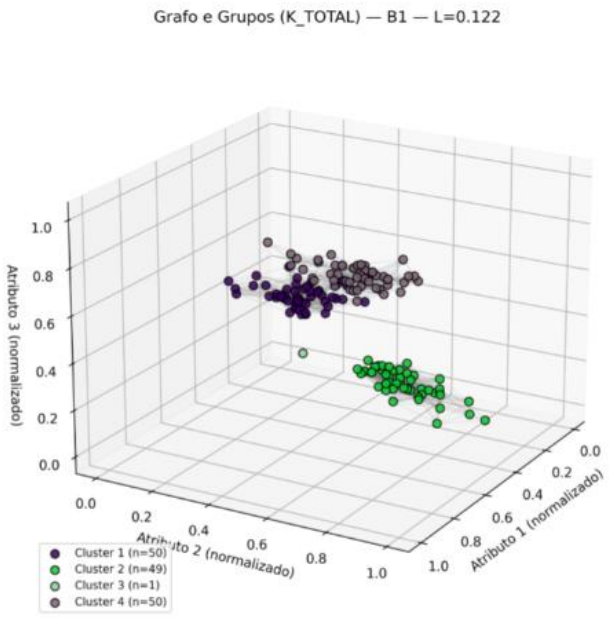
Fonte: Arquivo pessoal.

Figura 13 – Grafo e Grupos finais de B1



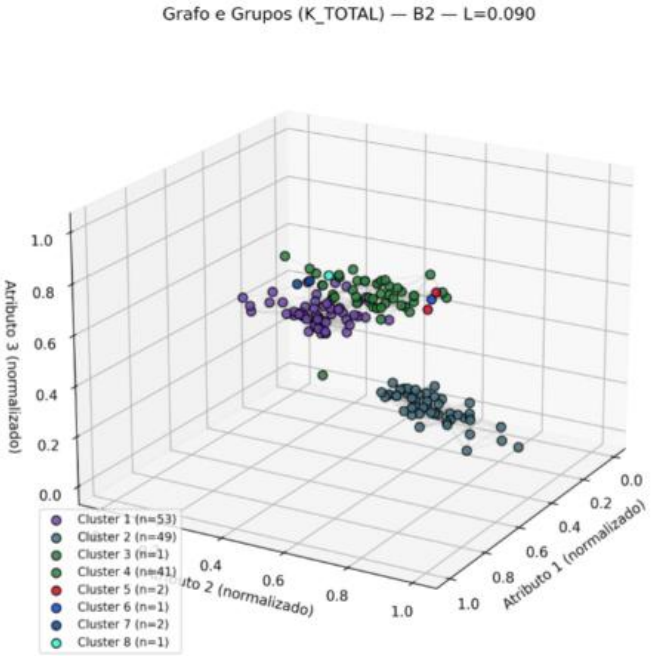
Fonte: Arquivo pessoal.

Figura 14 – Grafo e Grupos (k_total) de B1



Fonte: Arquivo pessoal.

Figura 15 – Grafo e Grupos (k_total) de B2



Fonte: Arquivo pessoal.

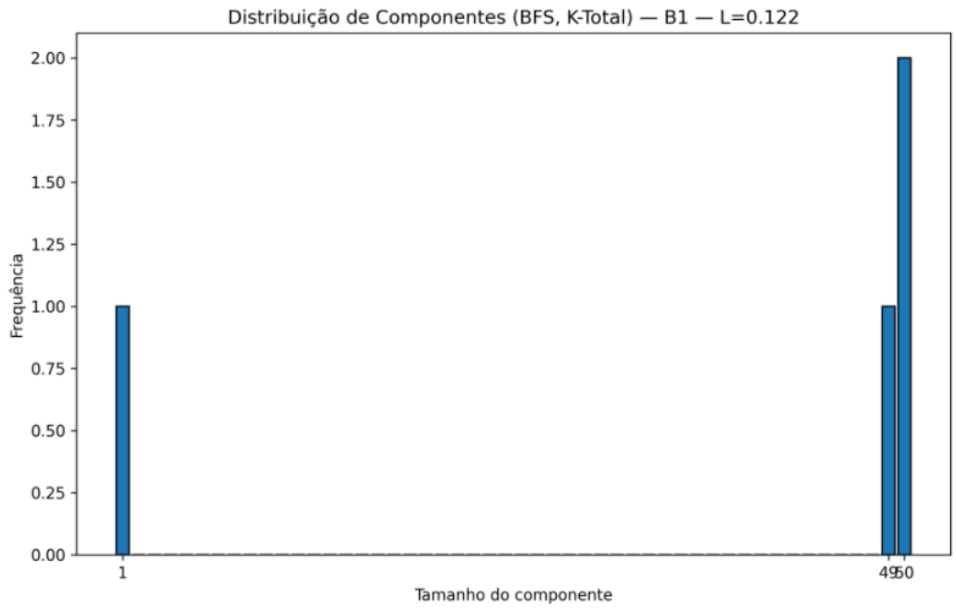
A análise comparativa das bases B1 e B2 demonstra a eficácia do algoritmo de Agrupamento por Componentes Conexos complementado pela reclassificação de *outliers*. Inicialmente (K_TOTAL), o limiar otimizado L*(0.122 para B1, 0.090 para B2) isolou com sucesso os três grupos principais em ambas as bases, embora a B2 tenha exibido maior fragmentação inicial, resultando em oito clusters (K_TOTA=8) devido à sua maior dispersão

de dados. No entanto, a etapa de reclassificação pós-agrupamento demonstrou ser altamente robusta, absorvendo eficientemente o ruído (clusters pequenos) para os centros dos três maiores grupos em ambos os casos. O resultado **FINAL** em ambas as bases foi um agrupamento com **três grupos coesos e muito bem balanceados** (próximos do ideal n=50/50/50), validando a capacidade do método de isolar as estruturas de dados centrais e corrigir o ruído periférico.

5.2. Histogramas

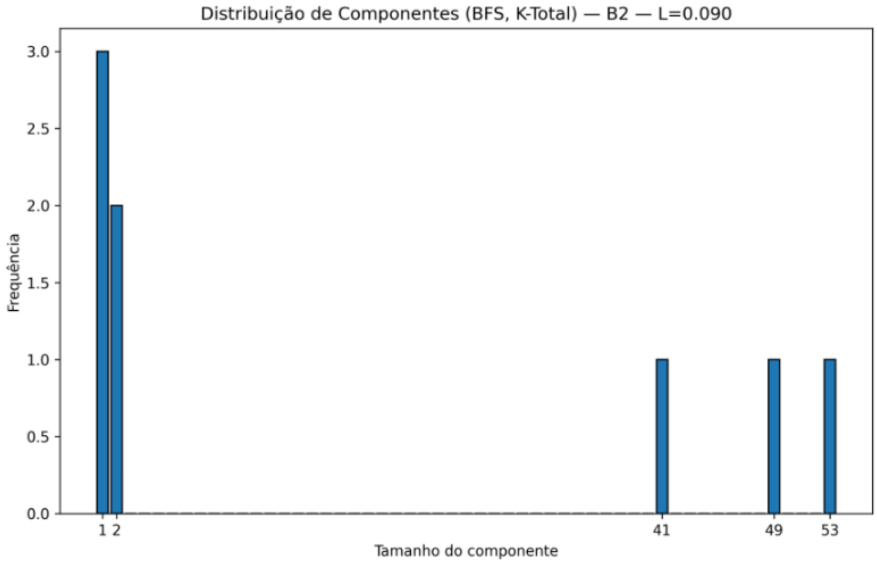
A análise desses histogramas detalha a Distribuição de Componentes (clusters) encontrados após a fase de construção do grafo e identificação via BFS, antes da reclassificação de *outliers:*

Figura 16 – Histograma de B1



Fonte: Arquivo pessoal.

Figura 17 – Histograma de B2



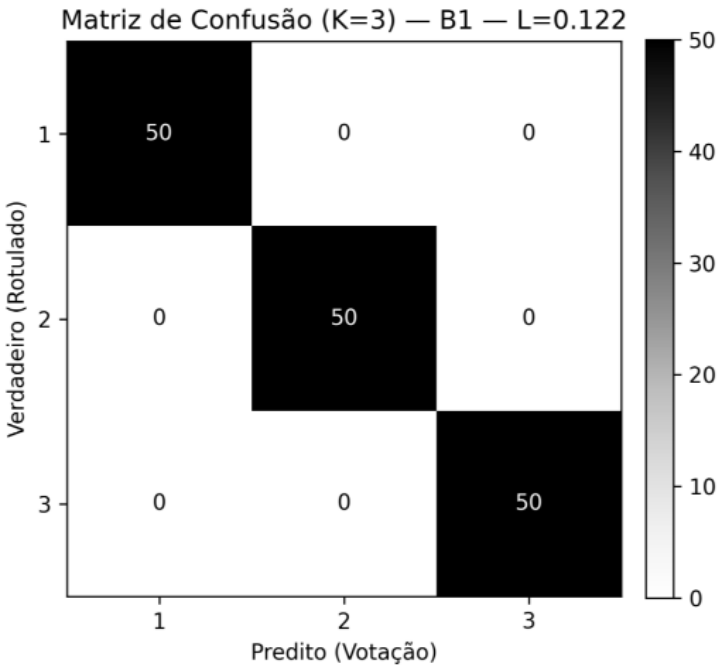
Fonte: Arquivo pessoal.

Os histogramas confirmam que o processo de otimização do L* consegue identificar as estruturas principais em ambas as bases. A Base B1 possui uma estrutura de dados mais nítida e separável (menor fragmentação), enquanto a Base B2 exige um limiar L menor para manter a coesão, resultando em maior fragmentação do ruído periférico. Em ambos os casos, o objetivo de isolar os três grupos principais (50/50/49 em B1 e 53/49/41 em B2) foi atingido, fornecendo a base para o agrupamento final robusto após o tratamento dos *outliers*.

5.3.Matriz de Confusão

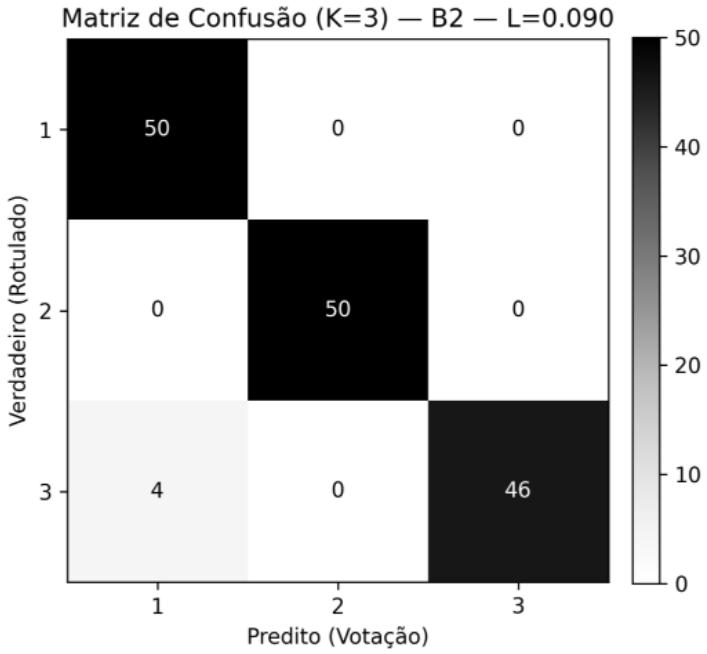
As matrizes de confusão fornecem a avaliação de desempenho final do algoritmo de agrupamento, comparando os rótulos verdadeiros com os rótulos preditos após a reclassificação de outliers e o mapeamento por votação (conclusão do processo K=3).

Figura 18 – Matriz de Confusão B1



Fonte: Arquivo pessoal.

Figura 19 – Matriz de Confusão B2



Fonte: Arquivo pessoal.

As matrizes de confusão confirmam que o algoritmo de agrupamento por componentes conexos e reclassificação atingiu um desempenho de **excelência** em ambas as bases de dados.

1) **Base B1**: Alcançou a **separação perfeita** (100% de acurácia), demonstrando que o L* ideal isolou três grupos puros.
2) **Base B2**: Obteve um resultado **quase perfeito** (97.33% de acurácia), com a totalidade dos erros concentrada na confusão de 4 amostras da Classe 3 com a Classe 1, sugerindo uma pequena ambiguidade espacial entre esses dois grupos na Base B2 que o limiar L=0.090 e a reclassificação não conseguiram resolver.

O sucesso do mapeamento por votação é evidente, pois em ambos os casos os clusters foram corretamente alinhados aos rótulos verdadeiros.

5.4.Resumo dos Resultados

1) **Base B1 (rotulada.csv):**
   a) 3 grupos equilibrados (≈50 vértices cada);
   b) L* = 0.122 → acurácia **100%**;
   c) Matriz de confusão diagonal perfeita (sem reclassificações incorretas);
   d) Separação clara entre os clusters no grafo 3D.

2) **Base B2 (Dataset...5_casos_de_proximidade.csv):**
   a) Mesmo equilíbrio, mas com 5 amostras propositalmente próximas entre classes;
   b) L* = 0.090 → acurácia **97,3%**, F1-macro ≈ **0,97**;
   c) Alguns vértices limítrofes reclassificados para o cluster mais próximo;
   d) Resultado condizente com a natureza da base, indicando **robustez e coerência**.

6. BIBLIOTECAS UTILIZADAS EM C

1) <stdio.h>
2) <stdlib.h>
3) <string.h>
4) <math.h>
5) <limits.h>
6) <direct.h>
7) <sys/stat.h>

8) <sys/types.h>

7. CONCLUSÕES

O sistema construído integra conceitos de teoria dos grafos, processamento de dados, cálculo geométrico e técnicas de avaliação estatística em uma solução completa para clustering não supervisionado. O projeto atinge o objetivo de formar três grupos coerentes e equilibrados em ambas as bases. A integração entre C (processamento) e Python (visualização) proporcionou uma análise quantitativa e qualitativa completa. A metodologia adotada é sólida tanto do ponto de vista teórico quanto computacional, aproveitando propriedades topológicas do grafo, como conectividade e proximidade, para identificar estruturas internas nos dados. A normalização rigorosa das coordenadas, a heurística precisa de escolha do limiar ótimo, o uso clássico de BFS para componentes conexos e a consolidação posterior por centros de gravidade resultam em um modelo versátil e interpretável. Finalmente, a etapa de avaliação garante que o usuário possa medir objetivamente a qualidade do agrupamento.