UNIVERSIDADE FEDERAL DO CARIRI

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALAN GABRIEL SILVA OLIVEIRA

CORE ESTRITO DE UM JOGO HEDÔNICO DE APRECIAÇÃO DE AMIGOS COMO UMA SOLUÇÃO DE *CLUSTERING*

JUAZEIRO DO NORTE - CE

2026

ALAN GABRIEL SILVA OLIVEIRA

CORE ESTRITO DE UM JOGO HEDÔNICO DE APRECIAÇÃO DE AMIGOS COMO UMA SOLUÇÃO DE *CLUSTERING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Federal do Cariri, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Nelson Carvalho Sandes

JUAZEIRO DO NORTE - CE

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Cariri
Sistema de Bibliotecas

048c Oliveira, Alan Gabriel Silva.
Core estrito de um jogo hedônico de apreciação de amigos como uma solução de *clustering* / Alan Gabriel Silva Oliveira. – 2026.
65 f. : il. color.

Monografia (Graduação em Ciência da Computação) – Centro de Ciências e Tecnologia, Departamento de Ciência da Computação, Universidade Federal do Cariri, Juazeiro do Norte, CE, 2026.

Orientador: Prof. Dr. Nelson Carvalho Sandes.

1. Ciência da computação. 2. Teoria dos jogos. 3. Jogos hedônicos. 4. Agrupamento de dados (Computação). 5. Algoritmos. I. Sandes, Nelson Carvalho (Orient.). II. Universidade Federal do Cariri. III. Título.

CDD 006.312

Bibliotecário: João Bosco Dumont do Nascimento – CRB 3/1355

ALAN GABRIEL SILVA OLIVEIRA

CORE ESTRITO DE UM JOGO HEDÔNICO DE APRECIAÇÃO DE AMIGOS COMO UMA SOLUÇÃO DE *CLUSTERING*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Federal do Cariri, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 30/03/2026.

BANCA EXAMINADORA

Prof. Dr. Nelson Carvalho Sandes (Orientador)
Universidade Federal do Cariri (UFCA)

Prof. Dr. Carlos Vinícius Gomes Costa Lima
Universidade Federal do Cariri (UFCA)

Prof. Dra. Luana Batista da Cruz
Universidade Federal do Cariri (UFCA)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

A Deus, por me permitir chegar até este momento e por iluminar meus caminhos durante toda esta jornada.

Ao Prof. Dr. Nelson Carvalho Sandes, pela orientação atenciosa, paciência e por todo o conhecimento compartilhado ao longo do desenvolvimento deste trabalho.

Ao Prof. Dr. Carlos Julian Menezes Araújo, coordenador do curso de Ciência da Computação, pelo apoio e dedicação à formação dos alunos.

Aos professores Prof. Dr. Carlos Vinícius Gomes Costa Lima e Prof.ª Dr.ª Luana Batista da Cruz, por aceitarem o convite para integrar a banca examinadora e pelas valiosas observações e contribuições realizadas neste trabalho.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Aos meus pais e minha irmã, que, mesmo nos momentos de minha ausência dedicados ao estudo, sempre me ensinaram que o futuro se constrói com a constância e o esforço no presente.

Aos amigos do *Go Platypus*, que conquistei durante o curso, pela convivência, apoio e companheirismo ao longo dessa caminhada.

Agradeço também a todos os professores, não apenas pelo conhecimento transmitido, mas por revelarem, em cada ensinamento, a importância do caráter, da sensibilidade e da verdadeira essência da educação no processo de formação profissional.

"A ciência é muito mais do que um corpo de conhecimento. É uma maneira de pensar."

(Carl Sagan)

RESUMO

O agrupamento de dados (*clustering*) é um conjunto de técnicas que visam analisar um conjunto de dados e dividi-lo em grupos, de modo que os elementos pertencentes a um mesmo grupo sejam semelhantes entre si, enquanto elementos de grupos distintos apresentem diferenças significativas. Uma das formas de se obter um agrupamento de dados é aplicar uma abordagem que gera uma partição do conjunto de dados. Entre as diversas abordagens existentes, algumas são baseadas em conceitos da teoria dos jogos cooperativos, área que estuda interações estratégicas entre agentes racionais e a formação de coalizões entre eles. Em particular, os jogos hedônicos são um tipo de jogo cooperativo em que jogadores tem preferências entre as coalizões que eles podem participar. Diversos conceitos da teoria dos jogos cooperativos já foram aplicados em algoritmos de agrupamento, como o valor de Shapley e, no contexto de jogos hedônicos, o uso de coalizões Nash estáveis. No entanto, no cenário de jogos hedônicos, ainda observa-se uma lacuna na utilização do conceito de núcleo (*core*), um dos mais robustos e fundamentais da teoria dos jogos cooperativos. Diante disso, este trabalho propõe dois novos algoritmos de *clustering*, o CEAC e o CEAC++, que identificam as componentes fortemente conexas do grafo associado ao jogo, diferindo no modo de selecionar os amigos de cada jogador. Os algoritmos propostos foram comparados com métodos clássicos, como o *k-means*, o agrupamento hierárquico e o *DBSCAN*. Os resultados experimentais indicaram diferenças estatisticamente significativas segundo o teste de Friedman, com diferenças entre o CEAC++ e os métodos *k-means* e agrupamento hierárquico com ligações *complete*, *single* e *average*, segundo o teste *post hoc* de Nemenyi, demonstrando o potencial da abordagem proposta.

Palavras-chave: Teoria dos jogos. Jogos hedônicos. Agrupamento de dados. Coalizão. Grupos.

ABSTRACT

Data clustering is a set of techniques aimed at analyzing a dataset and dividing it into groups such that the elements within the same group are similar to each other, while elements from different groups exhibit significant differences. One way to obtain a data clustering is by applying an approach that generates a partition of the dataset. Among the various existing approaches, some are based on concepts from cooperative game theory, a field that studies strategic interactions among rational agents and the formation of coalitions among them. In particular, hedonic games are a type of cooperative game in which players have preferences over the coalitions they may join. Several concepts from cooperative game theory have already been applied to clustering algorithms, such as the Shapley value and, in the context of hedonic games, the use of Nash-stable coalitions. However, in the context of hedonic games, there remains a gap regarding the use of the core, one of the most fundamental and robust concepts in cooperative game theory. Therefore, this work proposes two new clustering algorithms, the CEAC and CEAC++, which identify the strongly connected components of the graph associated with the game, differing in the way each player's friends are selected. The proposed algorithms were compared with classical methods, such as k-means, hierarchical clustering, and DBSCAN. The experimental results indicated statistically significant differences according to the Friedman test, with differences observed between CEAC++ and the k-means method as well as hierarchical clustering with complete, single, and average linkage, according to the Nemenyi post hoc test, demonstrating the potential of the proposed approach.

Keywords: Game theory. Hedonic games. Clustering. Coalition. Groups.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ARI	*Adjusted Rand Index*
CEAC	*Core* Estrito Aplicado à *Clustering*
CEAC++	*Core* Estrito Aplicado à *Clustering* ++
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*
DFS	*Depth-First Search*
TU	*Transferable Utility*
DRAC	*Density-Restricted Agglomerative Clustering*
LIFO	*Last In, First Out*
NTU	*Non-Transferable Utility*
RI	*Rand Index*
SCC	*Strongly Connected Component*

LISTA DE SÍMBOLOS

d	Distância
ε	Raio
G	Grafo
V	Conjunto de vértices
E	Conjunto de arestas
D	Grafo direcionado (digrafo)
D^T	Digrafo transposto de D
$\succeq$	Relação de preferência
C	Coalizão de jogadores
N	Conjunto de jogadores
CS	Estrutura de coalizão (partição de N)
J	Jogo
$P_i(j)$	Apreciação do jogador j pelo jogador i
$U(i)$	Utilidade do jogador i
A_i	Conjunto de amigos do jogador i
I_i	Conjunto de inimigos do jogador i
$\mathscr{N}_i$	Conjunto de todas as coalizões que contém o jogador i

SUMÁRIO

1 INTRODUÇÃO

Clustering é um conjunto de técnicas cujo objetivo é organizar elementos em grupos de modo que haja homogeneidade interna e heterogeneidade externa (TAN *et al.*, 2009). Nesta visão, um bom agrupamento tende a agrupar elementos similares no mesmo grupo, enquanto elementos considerados dissimilares são colocados em grupos diferentes. Essa área remonta à década de 1930, com trabalhos pioneiros de Driver e Kroeber (1932) e Tryon (1939). Trata-se de um campo amplo, com aplicações em diversas áreas do conhecimento, como ciência da computação, psicologia, biologia, antropologia e inteligência artificial.

Diversas abordagens foram propostas ao longo do tempo para identificar *clusters* em conjuntos de dados. Entre as mais conhecidas destacam-se o algoritmo *k-means*, o agrupamento hierárquico e o *Density-Based Spatial Clustering of Applications with Noise* (*DBSCAN*), amplamente utilizados em diferentes domínios por sua eficiência e simplicidade de implementação (TAN *et al.*, 2009).

Dependendo da visão do algoritmo adotado, o resultado de um problema de *clustering* pode ser dado por uma partição do conjunto de dados. Esta visão de agrupamento é bastante similar a problemas estudados na teoria dos jogos cooperativos. A teoria dos jogos cooperativos, originada com o trabalho seminal de Neumann e Morgenstern (1944), dedica-se ao estudo de coalizões, isto é, conjuntos de jogadores que decidem cooperar para maximizar ganhos coletivos ou individuais. Diferentemente da teoria dos jogos não cooperativos, que analisa decisões estratégicas de agentes agindo isoladamente, a vertente cooperativa estuda quais coalizões vão emergir a partir da interação entre jogadores e como recompensas obtidas por cada coalizão podem ser distribuídas entre os jogadores.

O resultado do estudo de formação de coalizões, de forma similar ao resultado do problema de *clustering*, é dado por uma partição do conjunto de jogadores, em que cada conjunto pertencente à partição é considerado uma coalizão. Nesse contexto, ao observar a similaridade entre os dois problemas, alguns trabalhos da literatura abordam o problema de *clustering* a partir de conceitos matemáticos da teoria dos jogos cooperativos. Nessa perspectiva, os pontos das bases de dados são considerados jogadores de um jogo cooperativo. Alguns desses trabalhos são apresentados a seguir.

1.1 Trabalhos relacionados

Uma das formas que a teoria dos jogos cooperativos usa para distribuir os valores das coalizões entre os jogadores é o valor de Shapley (CHALKIADAKIS *et al.*, 2011). Uma característica interessante do valor de Shapley é que os jogadores recebem um valor proporcional à contribuição deles nas possíveis coalizões do jogo. Ou seja, quanto mais importante for um jogador para as coalizões, maior será o seu valor de Shapley. Garg *et al.* (2012) mapearam o problema de *clustering* em um jogo cooperativo. Nesse trabalho, os autores mostraram como calcular o valor de Shapley do jogo em tempo polinomial e observaram que os pontos com maior valor de Shapley pertencem às regiões de maior densidade. Os autores usaram esta informação para criar grupos a partir dessas regiões densas. Outra aplicação deste algoritmo é usar os pontos com maiores valores de Shapley como centróides iniciais no algoritmo *k-means*.

Dhamal *et al.* (2012) também modelaram o problema de *clustering* em um jogo cooperativo. Neste jogo formulado, eles mostraram que o valor de Shapley coincide com outros conceitos de solução da teoria dos jogos cooperativos. Eles aproveitam o valor de Shapley para criar outro algoritmo chamado *Density-Restricted Agglomerative Clustering* (*DRAC*), baseado em densidade, para encontrar agrupamentos nas bases de dados.

Apesar de esses autores terem se aproveitado do problema de distribuição da utilidade das coalizões entre os jogadores, também é possível atacar o problema de *clustering* a partir do problema de formação de coalizões. Jogos hedônicos são uma classe de jogos cooperativos em que os jogadores possuem preferências entre as possíveis coalizões de um jogo. Por conta da característica deste jogo, o principal problema estudado nele é a formação das coalizões entre os jogadores (BOGOMOLNAIA; JACKSON, 2002).

Coelho e Sandes (2021) modelaram o problema de *clustering* como um jogo hedônico em que os pontos das bases de dados são considerados os jogadores do jogo. Os autores propuseram um algoritmo que sempre converge para uma partição que é Nash estável. Uma partição é considerada Nash estável quando todos os jogadores preferem permanecer em suas próprias coalizões a se unir a outras coalizões da partição. A mesma ideia foi utilizada no problema de *Clustering Ensembles* (SANDES; COELHO, 2018).

Nesse sentido, esses trabalhos exploram conceitos como o valor de Shapley, relacionado à distribuição de utilidade entre os jogadores, bem como a formação de coalizões baseada em noções de estabilidade, como a estabilidade de Nash. Este trabalho diferencia-se dos demais por propor a partição dos dados com base no conceito de *core*, para o qual, até o momento,

não foram encontrados estudos na literatura que o utilizem nesse contexto. Assim, a subseção seguinte apresenta a justificativa para a adoção dessa abordagem.

1.2 Justificativa da pesquisa

Um conceito importante no contexto de jogos hedônicos é quando uma partição de jogadores faz parte do *core* estrito de um jogo. Dado que uma partição de jogadores é definida por um conjunto de coalizões cujas interseções são vazias e cuja união corresponde ao conjunto de todos os jogadores, define-se informalmente que uma partição está no *core* estrito quando todos os jogadores preferem permanecer em suas coalizões a se unir a qualquer outra coalizão que seja subconjunto do conjunto de jogadores. Esta definição é imprecisa, mas fornece uma intuição do que seria uma partição pertencente ao *core* estrito.

Dimitrov *et al.* (2006) propôs um jogo hedônico de apreciação de amigos. Este jogo pode ser modelado através de um grafo direcionado em que os vértices do grafo representam os jogadores e a existência de uma aresta de um jogador a para outro qualquer b indica que a considera b como amigo. Neste contexto, os autores mostram que uma partição que pertence ao *core* estrito do jogo pode ser obtida a partir da geração dos componentes fortemente conexos do grafo. Levando em consideração este resultado, o objetivo deste trabalho é modelar o problema de *clustering* em um jogo hedônico de apreciação de amigos e verificar se uma partição que está no *core* estrito representa uma boa solução de *clustering*.

Também existem abordagens baseadas na teoria dos grafos para abordar o problema de *clustering*. Um trabalho similar ao proposto nesta pesquisa é descrito por Shekhar *et al.* (2018). Neste trabalho, os autores identificam regiões densas a partir de pontos que eles classificam como núcleos. Um grafo é formado de tal forma que os vértices são os núcleos e as arestas são definidas de um núcleo a para outro núcleo b se b é k-vizinho mais próximo de a. Os autores geram os componentes fortemente conexos deste grafo para representar os grupos de um agrupamento de dados. Dado que nem todo ponto é considerado um núcleo, existem pontos que não fazem parte deste agrupamento inicial. Por conta disso, os autores associam cada ponto sem grupo ao grupo mais próximo dele. Pontos que não estão suficientemente perto de qualquer grupo são considerados *outliers*.

É importante mencionar que a proposta desta pesquisa se diferencia da abordagem de Shekhar *et al.* (2018), pois todos os pontos das bases de dados serão considerados vértices do grafo para usar os componentes fortemente conexos como a solução de *clustering*. A

fundamentação desta escolha é dada pelo trabalho de Dimitrov *et al.* (2006), que informa que os componentes fortemente conexos do grafo devem gerar uma partição que pertence ao *core* estrito do jogo hedônico de apreciação de amigos.

1.3 Questões investigadas na pesquisa

Em relação à similaridade entre o problema de formação de coalizões e o problema de *clustering*, vale a pena refletir que a interpretação deles é diferente. A teoria dos jogos foca no estudo da estabilidade das partições geradas no jogo. Ou seja, situações em que os jogadores não desejam desviar de sua coalizão na partição obtida. Para isso, a teoria dos jogos lança mão de vários conceitos como o do *core* estrito e das partições Nash estáveis. Por outro lado, o problema de *clustering* procura partições nos dados de tal forma que pontos considerados similares estejam no mesmo grupo, enquanto pontos dissimilares são alocados em grupos diferentes. Considerando esta diferença de interpretação dos problemas e levando em consideração que o objetivo do trabalho é modelar um jogo em que os pontos das bases de dados são considerados jogadores de um jogo hedônico, as principais questões respondidas são:

1. *Existe alguma relação entre o conceito de estabilidade da teoria dos jogos cooperativos e bons agrupamentos de dados?*
2. *A partição pertencente ao core em um jogo hedônico de apreciação de amigos é relevante ao se levar em consideração o problema de clustering?*
3. *Como modelar a relação de amizade para que o core do jogo seja relevante do ponto de vista do problema de clustering?*
4. *Qual será a qualidade dos agrupamentos gerados usando esta abordagem em comparação com outras abordagens existentes?*

Com o intuito de definir formalmente os conceitos necessários para o desenvolvimento desta pesquisa, este trabalho se divide nos seguintes capítulos: O Capítulo 2 detalha os objetivos deste estudo. O Capítulo 3 apresenta a fundamentação teórica e cobre as definições de *clustering* e teoria dos grafos. Ainda nesta seção, são definidos formalmente jogos hedônicos e conceitos de solução do jogo, como *core*, *core* estrito e partições Nash estáveis. O Capítulo 4 aborda a metodologia adotada no trabalho. O Capítulo 5 apresenta e discute os resultados. Por fim, o Capítulo 6 apresenta as conclusões do trabalho e discute possíveis direções para pesquisas futuras.

2 OBJETIVOS

Os objetivos do presente projeto foram organizados em geral e específicos, conforme descrito a seguir.

2.1 Objetivo Geral

- Mapear o problema de *clustering* em um jogo hedônico de apreciação de amigos, explorando seus fundamentos teóricos e aplicabilidade prática.

2.2 Objetivos Específicos

- Realizar um levantamento bibliográfico sobre algoritmos de *clustering* baseados em teoria dos jogos e em grafos;
- Definir formalmente o conjunto A_i associado a cada ponto i no contexto do problema de *clustering*;
- Desenvolver um algoritmo para a construção do grafo que representa o jogo de apreciação de amigos;
- Implementar um algoritmo para a identificação de componentes fortemente conexas no grafo gerado;
- Aplicar a abordagem proposta a diferentes conjuntos de dados (*datasets*) artificiais e reais disponíveis na literatura;
- Comparar a abordagem proposta com algoritmos clássicos da literatura, utilizando o teste estatístico de Friedman para avaliar o desempenho relativo.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar os fundamentos teóricos que embasam o desenvolvimento deste trabalho. Para construir uma base sólida, são introduzidos três eixos principais de conceitos.

A Subseção 3.1 fala sobre *clustering*, apresenta os principais conceitos, métodos e modelos utilizados para identificar estruturas em conjuntos de dados. Além disso, são descritos algoritmos clássicos da literatura que serão utilizados como base de comparação com os algoritmos propostos neste trabalho.

A Subseção 3.2 fala sobre grafos, e fornece uma representação matemática adequada para modelar relações de proximidade e similaridade entre elementos, permitindo interpretar naturalmente *clusters* como subgrafos ou componentes conexas. O objetivo desta seção é apresentar os conceitos fundamentais necessários para compreender a estrutura do digrafo utilizado e os métodos para encontrar o *core* estrito de um jogo hedônico.

A Subseção 3.3 explora a teoria dos jogos cooperativos, com foco em jogos hedônicos, que permite analisar a formação de grupos de maneira estratégica, considerando as preferências individuais dos jogadores (ou pontos do *dataset*) e a estabilidade das coalizões formadas. Nessa seção é apresentado o conceito de *core* estrito, que possui forte relação com o conceito de componentes fortemente conexas em grafos, sendo esse o princípio fundamental do método proposto neste trabalho.

Juntos, esses três blocos teóricos fornecem a base necessária para o desenvolvimento do trabalho.

3.1 *Clustering*

O problema de agrupamento de dados, também conhecido como *clustering*, refere-se à técnicas e algoritmos que têm como objetivo particionar um conjunto de dados em subconjuntos (ou *clusters*) que são significativos, úteis ou ambos. Sua relevância reside na capacidade de identificar padrões e estruturas ocultas em dados não rotulados, auxiliando na compreensão e interpretação de fenômenos complexos (TAN *et al.*, 2009).

Para compreender os conceitos de *clustering* necessários para este trabalho, esta subseção está organizada da seguinte forma: a Seção 3.1.1 apresenta definições básicas relacionadas ao *clustering*, incluindo conceitos como grupos e medidas de distância. O objetivo

desta subseção é fornecer ao leitor os fundamentos necessários para compreender noções como grupos, agrupamentos, similaridade entre pontos e entre grupos, entre outros conceitos essenciais da área. As Subseções 3.1.2, 3.1.3 e 3.1.4 descrevem o funcionamento de algoritmos clássicos de *clustering*. O objetivo dessas subseções é apresentar métodos que exploram os conceitos definidos na Subseção 3.1.1, além de contextualizar os algoritmos que serão utilizados como base de comparação para avaliar a qualidade do método proposto.

3.1.1 Conceitos básicos de clustering

O objetivo da análise de *clustering* é formar *clusters* internamente homogêneos e externamente heterogêneos. Em outras palavras, quanto mais semelhantes entre si forem os elementos de um mesmo grupo e mais distintos forem em relação aos elementos de outros grupos, melhor será o resultado do *clustering*.

Uma característica central dessa técnica é que a formação dos *clusters* baseia-se exclusivamente nas informações presentes nos próprios dados, sem a utilização de rótulos prévios, sendo portanto uma abordagem de aprendizado não supervisionado.

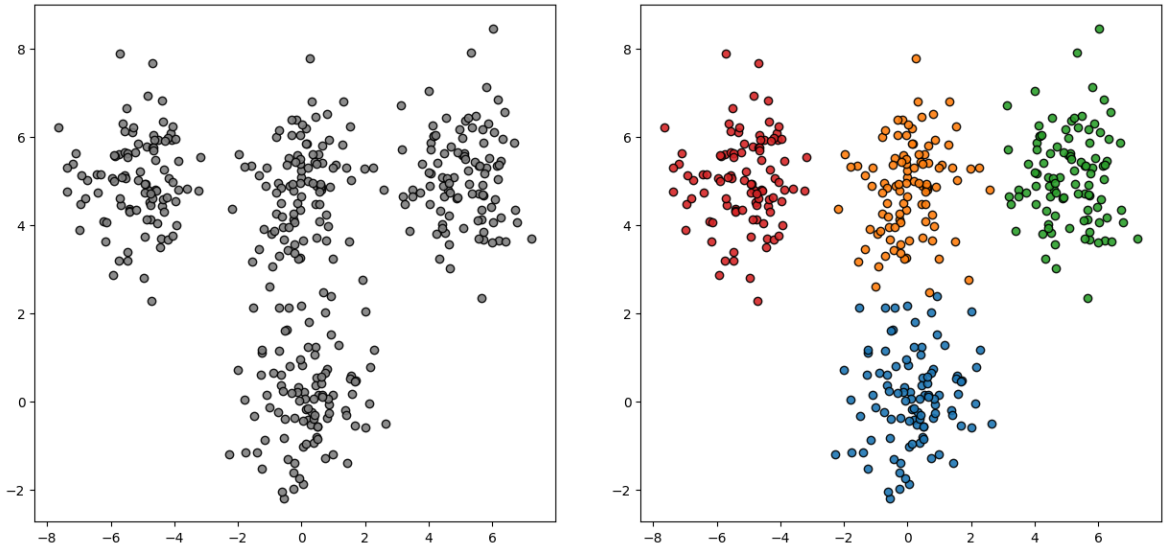
A Figura 1 ilustra um conjunto de dados e apresenta uma possível forma de agrupá-los. Nela, observa-se que os *clusters* formados contêm pontos próximos entre si, indicando um maior grau de similaridade.

Para compreender melhor o processo de agrupamento, é necessário definir alguns conceitos fundamentais, como grupos, similaridade e dissimilaridade entre pontos. As definições apresentadas a seguir são baseadas em Tan *et al.* (2009).

A primeira definição é a de grupo, que não é única e pode variar de acordo com o objetivo da análise de *clustering*. Algumas das principais definições para grupos encontradas na literatura são apresentadas a seguir.

- **Grupos bem separados**: cada objeto está mais próximo dos elementos de seu próprio grupo do que de qualquer elemento pertencente a outro grupo.
- **Grupos baseados em protótipos**: cada grupo é representado por um protótipo, como a média ou a moda de seus elementos. Os objetos são atribuídos ao grupo cujo protótipo está mais próximo.
- **Grupos baseados em grafos**: os objetos são representados como nós de um grafo, enquanto as relações de similaridade são representadas por arestas. Um grupo é formado por nós que apresentam fortes conexões entre si e conexões fracas com nós de outros

Figura 1 – Um conjunto de dados e uma possível forma de agrupá-los.



Fonte: Elaborado pelo autor.

grupos.

- **Grupos baseados em densidade**: um grupo corresponde a uma região do espaço de dados onde os pontos estão mais concentrados, sendo delimitada por regiões de baixa densidade.

Para identificar essas relações entre os objetos, é necessário definir medidas de similaridade ou dissimilaridade. Dado dois pontos p e q, pode-se estabelecer uma medida que quantifique o grau de proximidade entre eles. A escolha dessa medida depende da natureza dos dados e da aplicação considerada.

Assumindo que cada ponto possua m atributos e considerando o espaço euclidiano, algumas das medidas de dissimilaridade mais utilizadas são a distância euclidiana (Equação 3.1) e a distância de Manhattan (Equação 3.2).

$$d(p,q) = \sqrt{\sum_{i=1}^{m} (p_i - q_i)^2} \tag{3.1}$$

$$d(p,q) = \sum_{i=1}^{m} |p_i - q_i| \tag{3.2}$$

Em aplicações relacionadas ao agrupamento de textos ou dados de alta dimensionalidade, uma medida frequentemente mais adequada é a similaridade do cosseno (Equação 3.3), que considera o ângulo entre os vetores de características.

$$\text{sim}(p,q) = \frac{\sum_{i=1}^{m} p_i q_i}{\sqrt{\sum_{i=1}^{m} p_i^2} \cdot \sqrt{\sum_{i=1}^{m} q_i^2}} \tag{3.3}$$

Um conjunto de grupos formados a partir de um *dataset* é chamado de agrupamento. Na literatura, diferentes tipos de agrupamentos podem ser definidos de acordo com a estrutura dos grupos e a forma como os objetos são atribuídos a eles.

- **Agrupamento hierárquico e particional**: quando os grupos são organizados em uma estrutura de árvore, o agrupamento é chamado de hierárquico. Nesse caso, cada nó representa um grupo e seus subgrupos correspondem aos nós filhos. Quando os dados são diretamente divididos em grupos sem a existência de subgrupos, o agrupamento é chamado de particional, no qual cada elemento pertence a apenas um grupo.
- **Agrupamento exclusivo e difuso**: se cada objeto é atribuído a apenas um grupo, o agrupamento é chamado de exclusivo. Caso cada objeto possa pertencer simultaneamente a vários grupos, o agrupamento é chamado de difuso, no qual cada objeto possui diferentes graus de pertencimento aos grupos, geralmente representados por valores entre 0 e 1.
- **Agrupamento completo e incompleto**: quando todos os objetos são obrigatoriamente atribuídos a algum grupo, o agrupamento é chamado de completo. Caso alguns objetos não possam ser atribuídos a nenhum grupo, o agrupamento é considerado incompleto, e esses objetos são chamados de *outliers* ou pontos atípicos.

Essa variedade de abordagens evidencia a flexibilidade da análise de *clustering*, tornando-a uma ferramenta importante para explorar e organizar dados em diversas áreas do conhecimento. Entre os algoritmos mais conhecidos dessa área destacam-se o *k-means*, o agrupamento hierárquico e o *DBSCAN*, que serão apresentados nas seções seguintes.

3.1.2 k-means

O algoritmo *k-means* (MCQUEEN, 1967) é um dos métodos de agrupamento mais populares. Ele organiza os dados em torno de protótipos chamados centroides, de forma que os pontos sejam divididos em k grupos distintos. O objetivo é minimizar a distância entre os pontos e o centroide do grupo ao qual pertencem.

O funcionamento do algoritmo pode ser descrito em quatro etapas principais:

1. Inicialmente, escolhem-se k centroides distintos.
2. Para cada ponto do conjunto de dados, identifica-se o centroide mais próximo, com base em uma das medidas de distância apresentadas anteriormente. O ponto é então atribuído ao grupo correspondente a esse centroide.
3. Uma vez atribuídos os pontos, recalculam-se os centroides como sendo a média dos

elementos de cada grupo.

4. As etapas 2 e 3 são repetidas até que os centroides se estabilizem, ou seja, não sofram alterações significativas entre as iterações.

De maneira resumida, o *k-means* busca uma partição dos dados em k grupos, reduzindo a variabilidade dentro dos grupos em relação aos seus centroides.

O Algoritmo 1 ilustra esse processo (TAN *et al.*, 2009), enquanto a Figura 2 apresenta a execução do *k-means* em um *dataset*. Nessa figura, é possível observar que os centroides são gradualmente atualizados até convergirem para uma solução estável, que representa o agrupamento final dos dados.

Algoritmo 1: Algoritmo *k-means* simples

```
Selecione k pontos como centroides iniciais;
repeat
    Atribua cada ponto ao centroide mais próximo;
    Recalcule o centroide de cada grupo;
until Os centroides se estabilizem;
```

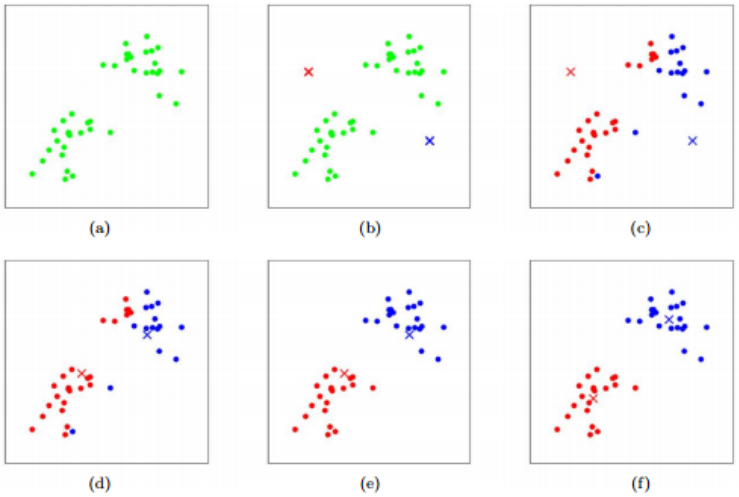
A complexidade do *k-means* é dada por $O(T \cdot k \cdot m \cdot n)$, em que T representa o número de iterações até a convergência, k é o número de grupos, m é a quantidade de pontos e n é a dimensionalidade dos dados. Na prática, o número de iterações tende a ser pequeno, o que torna o algoritmo eficiente quando $k \ll m$.

Entre as principais vantagens do *k-means*, estão a simplicidade e a eficiência, especialmente em grandes conjuntos de dados, quando os grupos possuem formato aproximadamente esférico (globular). Contudo, sua limitação aparece em cenários nos quais os *clusters* apresentam formas não esféricas ou densidades muito diferentes, além da dificuldade de escolher os k centroides iniciais. A Figura 3 apresenta um exemplo em que o *k-means* falha em encontrar um bom agrupamento. Nesse caso, embora os *clusters* possuam formato aproximadamente esférico, seus tamanhos são bastante distintos, o que dificulta a identificação correta dos grupos pelo algoritmo. Como resultado, o *k-means* tende a particionar os dados de maneira inadequada, produzindo agrupamentos que não refletem corretamente a estrutura original dos dados.

3.1.3 Agrupamento hierárquico

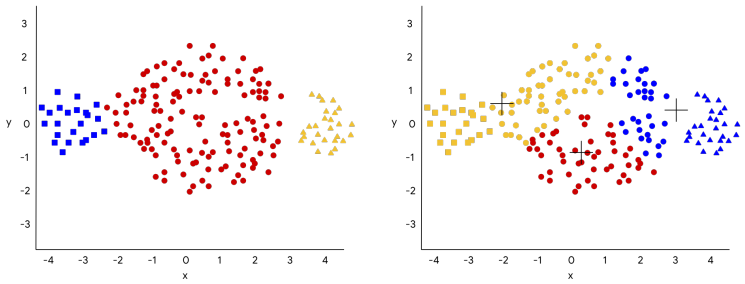
O agrupamento hierárquico (SOKAL *et al.*, 1958) organiza os dados por meio da construção de uma estrutura em árvore chamada dendrograma. Essa estrutura representa

Figura 2 – Exemplo da execução do *k-means*.



Fonte: (PIECH, 2013).

Figura 3 – Exemplo em que o *k-means* não apresenta bom desempenho.



Fonte: (GOOGLE, 2025).

graficamente as relações de proximidade entre os elementos do conjunto de dados, mostrando como eles são agrupados de maneira progressiva ao longo do processo.

No dendrograma, cada folha da árvore representa um objeto individual do conjunto de dados. À medida que o algoritmo avança, objetos ou grupos de objetos que apresentam maior similaridade são combinados, formando novos nós na árvore. Esses nós representam grupos maiores que contêm os elementos previamente unidos.

Dessa forma, o dendrograma fornece uma visualização da organização hierárquica dos dados, evidenciando como os grupos são formados a partir das relações de similaridade entre os elementos.

Existem duas abordagens principais para realizar o agrupamento hierárquico:

- **Método divisivo**: todos os pontos do conjunto de dados iniciam em um único grupo e, de forma iterativa, esse grupo é dividido em subgrupos menores até que cada ponto forme um grupo individual ou até que algum critério de parada seja atingido.
- **Método aglomerativo**: cada ponto do conjunto de dados inicia em seu próprio grupo e,

a cada iteração, os dois grupos mais próximos são fundidos, formando grupos progressivamente maiores até que todos os pontos pertençam a um único grupo ou até que algum critério de parada seja satisfeito.

Neste trabalho, será considerada a abordagem aglomerativa. Nesta abordagem, a cada etapa do algoritmo, é necessário identificar quais são os dois grupos mais próximos para que sejam fundidos. Para isso, torna-se necessário definir uma medida de distância entre grupos, também conhecida como *linkage*. Diferentes critérios podem ser utilizados para calcular essa distância, como os métodos *single linkage*, *complete linkage*, *average linkage* e o *ward linkage*, cada um deles definindo a proximidade entre grupos de maneira distinta (TAN *et al.*, 2009).

No *single linkage*, também conhecido como método do vizinho mais próximo, a distância entre dois grupos é definida como a menor distância entre qualquer par de pontos pertencentes a esses grupos. Dessa forma, dois grupos podem ser considerados próximos caso exista ao menos um par de pontos, um em cada grupo, com pequena distância entre si. Esse método tende a formar grupos mais alongados.

No *complete linkage*, ou método do vizinho mais distante, a distância entre dois grupos é definida como a maior distância entre os pares de pontos pertencentes aos grupos. Esse critério tende a gerar grupos mais compactos, pois a fusão de grupos ocorre apenas quando todos os seus pontos estão relativamente próximos.

O *average linkage* define a distância entre dois grupos como a média das distâncias entre todos os pares de pontos pertencentes a esses grupos. Esse método busca um equilíbrio entre os critérios utilizados pelo *single linkage* e pelo *complete linkage*, sendo menos sensível a valores extremos.

Por fim, o *ward linkage* difere dos anteriores por não considerar diretamente a distância entre pares de pontos. Em vez disso, ele seleciona para fusão os grupos cuja união provoca o menor aumento na variância interna dos *clusters*, geralmente medida pela soma dos erros quadráticos. Como consequência, esse método tende a produzir grupos mais compactos e aproximadamente esféricos.

O Algoritmo 2 apresenta uma versão simplificada do agrupamento aglomerativo (TAN *et al.*, 2009).

O resultado desse algoritmos é um dendrograma que permite visualizar de forma intuitiva o processo de fusão ou divisão dos grupos ao longo do algoritmo. A partir dessa representação, os *clusters* podem ser obtidos selecionando-se um determinado nível da árvore, o

Algoritmo 2: Algoritmo de agrupamento hierárquico aglomerativo básico

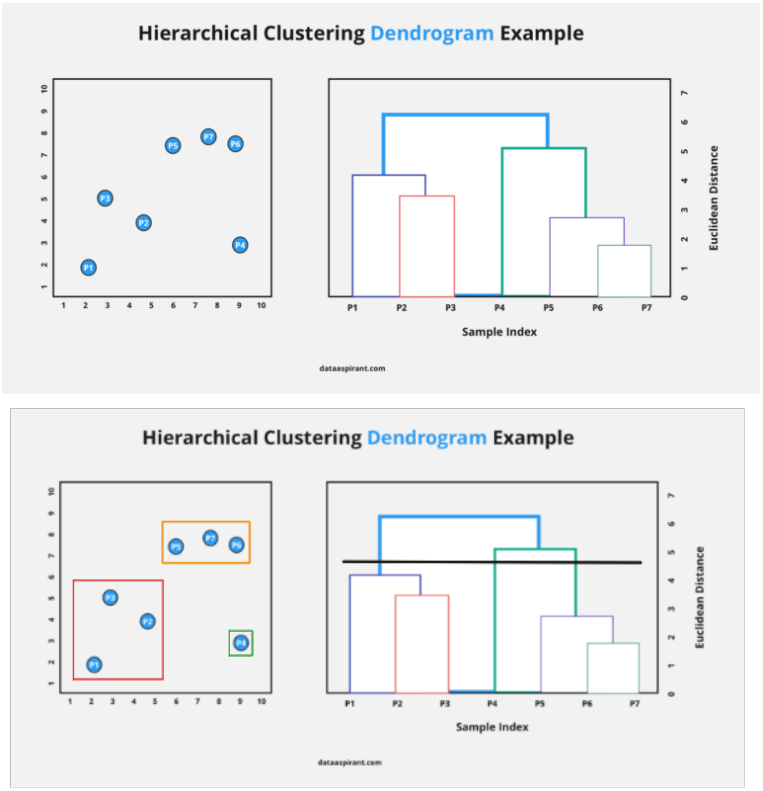
Calcule a matriz de proximidade, contendo as distâncias entre todos os pares de grupos;
repeat
    Funda os dois grupos mais próximos;
    Atualize a matriz de proximidade para refletir a nova configuração;
until *Reste apenas um grupo*;

que equivale a realizar um corte horizontal no dendrograma.

Esse corte define quais uniões entre grupos serão consideradas no agrupamento final: os ramos da árvore que permanecem desconectados acima desse nível correspondem aos *clusters* formados. Dessa forma, diferentes níveis de corte resultam em diferentes quantidades de grupos, permitindo analisar os dados em variados graus de granularidade.

Na prática, a escolha do ponto de corte pode ser guiada por conhecimento prévio do domínio do problema ou por testes empíricos que busquem identificar a estrutura mais adequada dos dados. A Figura 4 apresenta o dendrograma obtido para um conjunto de dados e um exemplo de corte que permite identificar os respectivos *clusters*.

Figura 4 – Corte em um dendrograma ilustrando a formação dos *clusters*.



Fonte: Adaptado de Sultana (2020).

A complexidade computacional desse algoritmo é $O(m^2 \log m)$, o que restringe sua

aplicação a conjuntos de dados de tamanho moderado, uma vez que o custo cresce rapidamente para grandes volumes de informações.

A principal vantagem do método está em sua capacidade de revelar relações hierárquicas naturais entre os dados, como ocorre em estudos de taxonomia biológica. Sua limitação, entretanto, está no custo computacional elevado, que o torna menos adequado em cenários de *big data*.

3.1.4 DBSCAN

O algoritmo *DBSCAN* (ESTER *et al.*, 1996) é um método de agrupamento baseado na densidade dos pontos. Ele identifica regiões densas de pontos como *clusters*, separadas por regiões de baixa densidade, e trata pontos isolados como ruído (*outliers*).

O funcionamento do *DBSCAN* pode ser descrito em quatro etapas principais:

1. Define-se um raio ε e um número mínimo de pontos *minPts* necessários para formar um núcleo de *cluster*.
2. Cada ponto que possui pelo menos *minPts* vizinhos dentro do raio ε é considerado um ponto núcleo.
3. Pontos vizinhos dos núcleos são adicionados ao *cluster* correspondente, repetindo o processo até que todos os pontos densamente conectados sejam incluídos no *cluster*.
4. Pontos que não pertencem a nenhum núcleo são rotulados como *outliers*.

O Algoritmo 3 formaliza a ideia do *DBSCAN* (TAN *et al.*, 2009).

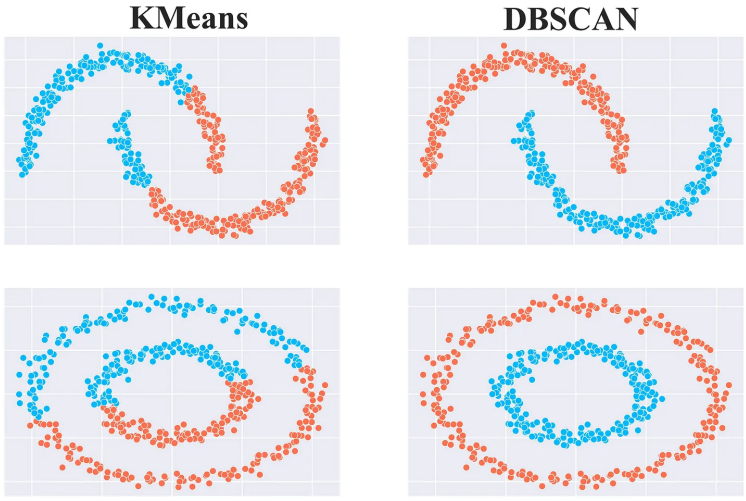
Algoritmo 3: Algoritmo *DBSCAN*

Rotule todos os pontos como núcleo, vizinho ou *outlier*;
Elimine os *outliers*;
Conecte todos os pontos núcleo que estejam dentro do raio ε;
Atribua a cada ponto vizinho ao *cluster* mais próximo;

O *DBSCAN* é especialmente útil para identificar *clusters* de formas arbitrárias e lidar naturalmente com ruído. A Figura 5 mostra a diferença entre o *k-means* e o *DBSCAN*, mostrando como o *DBSCAN* é útil para encontrar *clusters* em formato irregulares, ou seja, *clusters* não esféricos.

Entre suas limitações, destaca-se o desempenho quando os *clusters* possuem densidades muito diferentes, pois a escolha de ε e *minPts* pode não ser adequada para todos os grupos. A complexidade computacional do algoritmo é $O(m^2)$, podendo ser reduzida para $O(m \log m)$

Figura 5 – Comparação entre os agrupamentos do *DBSCAN* e do *k-means*.



Fonte: (SCIENCE, 2023).

com o uso de estruturas espaciais eficientes, como árvores *k*-d (BENTLEY, 1975).

Uma maneira bastante utilizada de modelar relações de similaridade entre pontos em problemas de *clustering* consiste em representar o conjunto de dados como um grafo. Nesse contexto, cada ponto é associado a um vértice, enquanto as relações de proximidade ou similaridade entre os pontos são representadas por arestas. A partir dessa representação, conceitos da teoria dos grafos, como conectividade e componentes fortemente conexos, podem ser explorados para identificar estruturas de agrupamento nos dados.

Dessa forma, a próxima seção apresenta alguns conceitos fundamentais da teoria dos grafos que serão utilizados ao longo deste trabalho.

3.2 Grafos

A teoria dos grafos é um ramo da matemática dedicado ao estudo das relações entre objetos por meio de estruturas chamadas grafos. Sua origem remonta ao século XVIII, com os trabalhos de Leonhard Euler sobre o problema das sete pontes de Königsberg (EULER, 1741). Ao longo do tempo, essa área evoluiu significativamente e passou a desempenhar um papel fundamental em diversos campos do conhecimento, incluindo matemática, ciência da computação, engenharia, química e biologia.

Esta seção tem como objetivo apresentar os principais conceitos de teoria dos grafos necessários para a compreensão do método proposto neste trabalho, com ênfase na construção do conceito de componentes fortemente conexas, que constitui uma de suas bases teóricas. Para isso, a Subseção 3.2.1 introduz definições básicas relacionadas a grafos. Em seguida, a Subseção

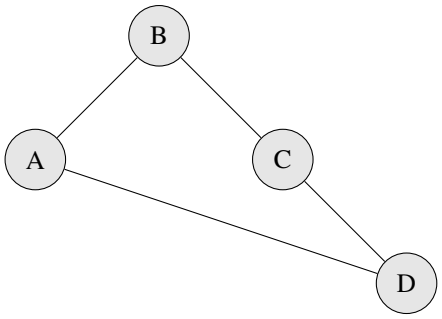
3.2.2 apresenta os grafos direcionados, também chamados de digrafos, bem como conceitos associados, incluindo o de componentes fortemente conexas. Por fim, as Subseções 3.2.3 e 3.2.4 descrevem dois algoritmos clássicos para a identificação de componentes fortemente conexas na literatura: o algoritmo de Tarjan e o algoritmo de Kosaraju-Sharir.

3.2.1 Definições gerais de grafos

Um grafo é uma estrutura matemática utilizada para representar relações entre pares de objetos. Formalmente, pode-se definir um grafo G representado pelo par (V,E), onde V é o conjunto de vértices (ou nós) e E é o conjunto de arestas.

Cada aresta $e \in E$ conecta dois vértices $u,v \in V$, chamados de extremidades de e, e pode ser representada pelo par $e = (u,v)$ ou $e = uv$. Dois vértices são ditos adjacentes quando existe uma aresta conectando-os, ou seja, $(u,v) \in E$. A Figura 6 mostra um exemplo de um grafo.

Figura 6 – Grafo G.



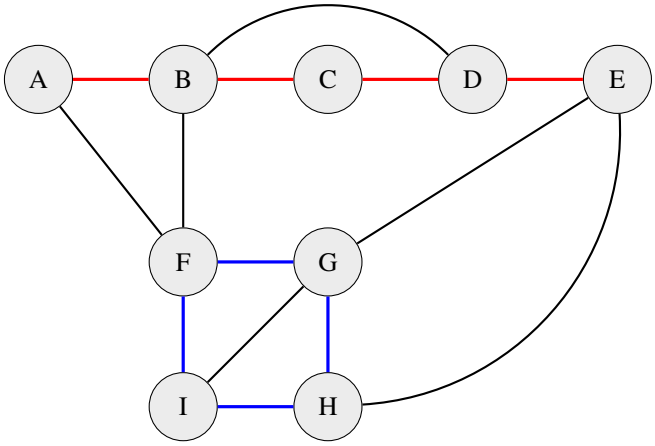
Fonte: Elaborado pelo autor.

A partir dessa definição, pode se caracterizar as propriedades dos vértices em relação às arestas que os conectam, como o número de conexões que cada vértice possui. Neste cenário, o *grau* de um vértice $v \in V$, denotado por $d(v)$, corresponde ao número de arestas incidentes em v, formalmente definido como $d(v) = |\{u \in V : (u,v) \in E\}|$. Outro conceito que está relacionado ao grau de um vértice é a vizinhança de um vértice $v \in V$ que é obtida pelo conjunto de todos os vértices adjacentes a ele: $N(v) = \{u \in V : (u,v) \in E\}$.

Com base nessas definições, pode-se introduzir os conceitos de passeio, caminho e ciclo, que descrevem sequências de vértices e arestas percorrendo o grafo. Uma sequência alternada de vértices e arestas $W = (v_1, e_1, v_2, e_2, ..., v_{k-1}, e_{k-1}, v_k)$, onde $e_i = (v_i, v_{i+1}) \in E$, é chamada de passeio em G. Os vértices v_1 e v_k são as extremidades do passeio W, e todos os demais vértices de W são chamados de vértices internos. Se $v_1 = v_k$, o passeio é denominado

passeio fechado. Um caminho simples C é um passeio que não repete vértices. Um ciclo é um caminho C cujas extremidades coincidem, isto é, $v_1 = v_k$. A Figura 7 ilustra os conceitos de caminho e ciclo. Nessa figura é possível ver o caminho simples A, B, C, D, E e o ciclo F, G, H, I, F.

Figura 7 – Grafo G com um caminho simples (em vermelho) e um ciclo (em azul).



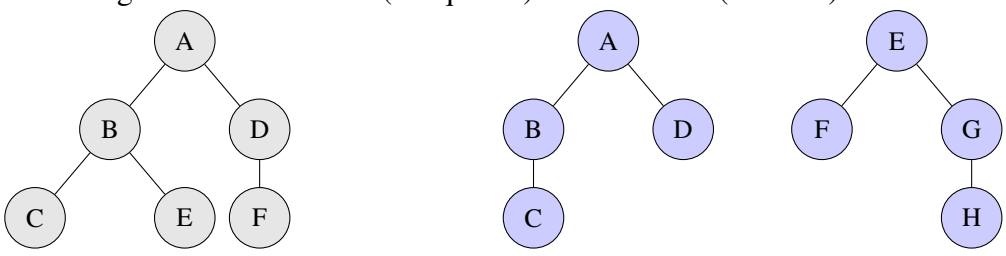
Fonte: Elaborado pelo autor.

Dados $u, v \in V$, u alcança v se existe um caminho entre eles. Um grafo G é dito conexo se, para quaisquer $u, v \in V$, existe um caminho entre u e v. Caso contrário, G é desconexo.

Pode-se também relacionar essas ideias à noção de subgrafos. Seja $G = (V, E)$. Um grafo $G' = (V', E')$ é um subgrafo de G se $V' \subseteq V$ e $E' \subseteq E$, com todas as arestas de E' incidindo apenas sobre vértices de V'. Uma componente conexa de G é um subgrafo conexo maximal, ou seja, não existe outro subgrafo conexo de G que contenha propriamente esse subgrafo.

Finalmente, conceitos especiais de grafos conexos sem ciclos podem ser definidos: um grafo conexo que não possui ciclos é chamado de árvore. Se o grafo não contiver ciclos e for desconexo, ele é chamado de floresta. Em outras palavras, uma floresta pode ser vista como um conjunto de árvores disjuntas. A Figura 8 ilustra os conceitos de árvore e floresta, onde é possível ver que a diferença entre eles está na conectividade.

Figura 8 – Uma árvore (a esquerda) e uma floresta (a direita).



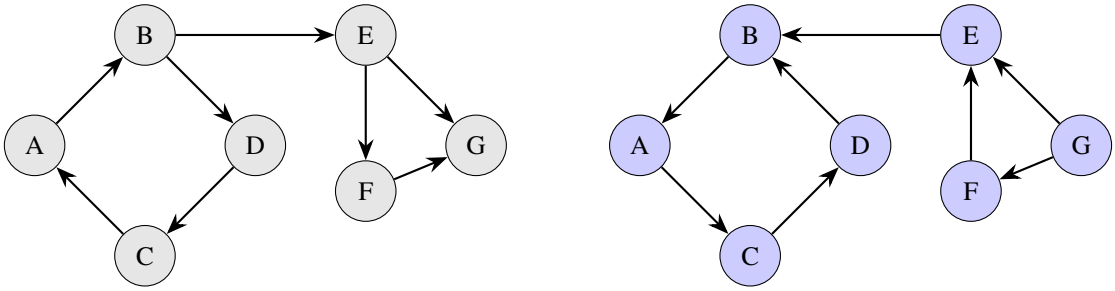
Fonte: Elaborado pelo autor.

3.2.2 Grafos direcionados

A definição apresentada anteriormente refere-se a grafos não direcionados (ou simplesmente, grafos). No entanto, em muitas aplicações práticas, como o método proposto neste trabalho, é necessário considerar relações assimétricas entre os objetos, o que nos leva ao conceito de grafos direcionados. Pode-se definir um grafo direcionado (ou digrafo) D pelo par (V,E), onde V é o conjunto de vértices e $E \subseteq V \times V$ é um conjunto de pares ordenados de vértices, chamados de arestas direcionadas ou arcos. A principal diferença em relação aos grafos não direcionados está no fato de que, em um digrafo, a ordem dos vértices importa: o arco (u,v) é distinto do arco (v,u). Assim, cada arco $e = uv$ possui apenas uma direção, indo de u para v.

Com base nessa definição, pode-se introduzir algumas variações e conceitos derivados de digrafos. Seja $D = (V,E)$ um digrafo. O grafo transposto de D é o digrafo $D^T = (V,E^T)$ tal que $uv \in E^T \iff vu \in E$. Em outras palavras, D^T possui os mesmos vértices de D, mas todas as arestas têm suas direções invertidas. Um exemplo pode ser encontrado na Figura 9, em que é possível ver que os vértices são idênticos, mas as arestas têm suas direções invertidas.

Figura 9 – Digrafo D e o digrafo D^T, transposto a partir de D.



Fonte: Elaborado pelo autor.

Outra construção importante é o grafo subjacente. Dado um digrafo D, o grafo subjacente S de D é o grafo não direcionado resultante da remoção das direções das arestas de D.

A partir dessas construções, pode-se estender o conceito de conectividade para digrafos. Seja $D = (V,E)$ um digrafo. Se, para todo par de vértices $u,v \in V$, existir um caminho em D de u para v e também de v para u, então D é dito fortemente conexo. Se o grafo subjacente S de D é conexo, então D é dito fracamente conexo. Caso contrário, D é desconexo. Note que todo digrafo fortemente conexo é necessariamente fracamente conexo, uma vez que a existência de caminhos em ambas as direções entre dois vértices garante também a conectividade no grafo subjacente. Entretanto, a recíproca não é verdadeira: um digrafo pode ser fracamente conexo sem ser fortemente conexo.

Por fim, pode-se definir um conceito central no estudo de digrafos, e que será fundamental para esse trabalho: as componentes fortemente conexas. Uma componente fortemente conexa (do inglês, *Strongly Connected Component* (*SCC*)) de um digrafo D é um subdigrafo maximal de D que seja fortemente conexo, isto é, não existe outro subdigrafo fortemente conexo de D que o contenha.

Existem diversos algoritmos para encontrar as componentes fortemente conexas de digrafos, com destaque para o algoritmo de Tarjan, (TARJAN, 1972); e o algoritmo de Kosaraju-Sharir, (SHARIR, 1981). Para introduzi-los, é necessário revisar o algoritmo de busca em profundidade (do inglês, *Depth-First Search* (*DFS*)), cujo objetivo é explorar os vértices de um grafo, avançando o máximo possível em cada caminho antes de retroceder (CORMEN *et al.*, 2022). Esse algoritmo utiliza uma pilha, estrutura de dados do tipo *Last In, First Out* (*LIFO*). Os Algoritmos 4 e 5 mostram o funcionamento da busca (CORMEN *et al.*, 2022).

Algoritmo 4: Busca em Profundidade *DFS*.

Input: Grafo $G = (V, E)$ representado por listas de adjacências.

Output: Tempos de descoberta $d[v]$ e término $f[v]$ de cada vértice.

```
foreach v ∈ V do
    cor[v] ← branco;
    π[v] ← nulo;
end
tempo ← 0;
foreach v ∈ V do
    if cor[v] = branco then
        DFS-Visita(G, v);
    end
end
```

Algoritmo 5: DFS-Visita

Input: Grafo $G = (V, E)$ e vértice $v \in V$.

cor[v] $\leftarrow$ cinza;

tempo $\leftarrow$ tempo + 1;

$d[v] \leftarrow$ tempo;

foreach $u \in Adj[v]$ **do**

  if *cor[u] = branco* **then**

    $\pi[u] \leftarrow v$;

    DFS-Visita(G, u);

  end

end

cor[v] $\leftarrow$ preto;

tempo $\leftarrow$ tempo + 1;

$f[v] \leftarrow$ tempo;

3.2.3 Algoritmo de Tarjan

O algoritmo de Tarjan recebe como entrada um digrafo $D = (V, E)$ e retorna uma partição de V em subconjuntos disjuntos, correspondentes às componentes fortemente conexas de D.

A ideia central do algoritmo consiste em executar uma *DFS* no grafo, produzindo uma floresta de busca. Durante a execução, é utilizada uma pilha para controlar os vértices que ainda pertencem ao caminho de recursão ativo. Diferentemente da *DFS* clássica, os vértices não são necessariamente removidos da pilha ao término da chamada recursiva, mas apenas quando sua componente fortemente conexa é identificada.

Para cada vértice $v \in V$, o algoritmo mantém dois valores associados:

- **índice** ($index[v]$): representa a ordem em que o vértice v foi descoberto durante a execução da *DFS*;
- **menor alcançável** ($lowlink[v]$): corresponde ao menor índice de qualquer vértice que pode ser alcançado a partir de v, considerando o próprio vértice, seus descendentes na árvore de busca e vértices que ainda permanecem ativos na pilha.

A invariante fundamental do algoritmo estabelece que um vértice permanece na pilha após ser visitado se, e somente se, ainda existir um caminho no grafo que leve desse vértice até

algum outro vértice que esteja anteriormente presente na pilha.

Quando um vértice v satisfaz $lowlink[v] = index[v]$, isso significa que v é o vértice de menor índice de uma componente fortemente conexa. Nesse momento, o algoritmo remove vértices da pilha até encontrar v, formando assim uma nova componente fortemente conexa. Uma descrição mais detalhada pode ser vista nos Algoritmos 6 e 7.

Algoritmo 6: Algoritmo de Tarjan para encontrar componentes fortemente conexas

Input: Um grafo $G = (V, E)$

Output: O conjunto das componentes fortemente conexas de G

index $\leftarrow 0$;

Seja S uma pilha vazia;

foreach $v \in V$ **do**

  if *v.index não definido* **then**

    TARJANVISITA(v);

  end

end

Algoritmo 7: TARJANVISITA(v)

Input: Um grafo $G = (V, E)$ e um vértice v

$v.index \leftarrow$ index;

$v.lowlink \leftarrow$ index;

index $\leftarrow$ index + 1;

Empilhe v em S;

$v.emPilha \leftarrow$ verdadeiro;

foreach $(v, w) \in E$ **do**

  if *w.index não definido* **then**

    TARJANVISITA(w);

    $v.lowlink \leftarrow \min(v.lowlink, w.lowlink)$;

  end

  else if *w.emPilha* **then**

    $v.lowlink \leftarrow \min(v.lowlink, w.index)$;

  end

end

if $v.lowlink = v.index$ **then**

  Inicie uma nova componente C;

  repeat

    $w \leftarrow$ desempilhe S;

    $w.emPilha \leftarrow$ falso;

    adicione w a C;

  until *até que* $w = v$;

  Adicione C ao conjunto de *SCC*s;

end

Esse algoritmo permite encontrar as *SCC*s de um digrafo em tempo $O(|V| + |E|)$. A seguir, apresenta-se uma abordagem alternativa que também busca resolver o mesmo problema abordado pelo algoritmo de Tarjan.

3.2.4 Algoritmo de Kosaraju-Sharir

O algoritmo de Kosaraju-Sharir identifica as *SCC*s de D seguindo os seguintes passos:

1. Executar uma busca em profundidade (DFS) em D para cada vértice não visitado, registrando os tempos de término de cada vértice.
2. Construir o grafo transposto D^T.
3. Executar DFS em D^T, processando os vértices na ordem decrescente de tempo de término obtida na DFS de D.
4. Cada árvore resultante da DFS em D^T corresponde a uma componente fortemente conexa de D.

A ideia central do algoritmo é que a *DFS* em D^T, realizada na ordem de término inversa, garante que cada chamada recursiva percorra todos e apenas os vértices de uma *SCC*, formando assim a partição do grafo em suas componentes fortemente conexas. A ideia mais formalizada pode ser vista no Algoritmo 8.

Algoritmo 8: Algoritmo de Kosaraju-Sharir para componentes fortemente conexas

DFS(D);
Calcular D^T;
Chamar DFS(D^T), mas no laço principal, selecionar os vértices em ordem crescente de tempo i obtido na chamada de DFS(D) (linha 1);
return Cada árvore da floresta como uma componente fortemente conexa;

Em termos de complexidade, esse algoritmo possui o mesmo custo assintótico do algoritmo de Tarjan, isto é, $O(|V| + |E|)$.

A partir dessa fundamentação, a próxima seção abordará a teoria dos jogos cooperativa, com foco na classe de jogos hedônicos, estabelecendo relações entre seus princípios e os conceitos de grafos apresentados anteriormente.

3.3 Teoria dos jogos cooperativos

A teoria dos jogos cooperativos é um ramo da teoria dos jogos que estuda a formação de grupos, ou coalizões, entre jogadores. Cada coalizão é composta por um subconjunto de jogadores e possui uma utilidade associada. Os jogos cooperativos podem ser classificados em duas categorias principais: jogos de utilidade transferível (do inglês, *Transferable Utility* (*TU*)) e jogos de utilidade não transferível (do inglês, *Non-Transferable Utility* (*NTU*)). Neste trabalho, o foco recai sobre uma subclasse de jogos *NTU* denominada jogos hedônicos, nos quais as preferências de cada jogador dependem apenas da composição da coalizão à qual pertence.

As definições de jogos cooperativos apresentadas a seguir baseiam-se em Chalkia-

dakis *et al.* (2011) e servirão como fundamento para o mapeamento do problema de teoria dos jogos para um problema de *clustering*. Para isso, a Seção 3.3.1 apresenta os conceitos iniciais de jogos cooperativos, com ênfase em jogos hedônicos. Em seguida, a Seção 3.3.2 discute os principais conceitos de estabilidade em jogos hedônicos. Por fim, a Seção 3.3.3 apresenta uma subclasse específica desses jogos, os jogos aditivamente separáveis. Essa seção é particularmente importante, pois introduz o jogo de apreciação de amigos, um tipo de jogo aditivamente separável que constitui a base do método proposto neste trabalho.

3.3.1 Conceitos básicos de jogos cooperativos

Formalmente, um jogo cooperativo de utilidade transferível (*TU*) é definido pelo par (N, v), em que N é um conjunto não vazio de jogadores e $v : 2^N \to \mathbb{R}$ é uma função que atribui a cada coalizão $C \subseteq N$ um valor real $v(C)$. O conjunto C é chamado de coalizão e $v(C)$ representa o valor dessa coalizão. Uma estrutura de coalizão (ou em inglês, *Coalition Structure*) é dada por $CS = \{C_1, C_2, \ldots, C_k\}$, uma partição do conjunto N, tal que $\bigcup_{i=1}^{k} C_i = N$ e $C_i \cap C_j = \emptyset$, para todo $i \neq j$. Usa-se CS_i para denotar a coalizão em CS que contém o jogador i. O resultado de um jogo cooperativo é representado pelo par (CS, x), em que CS é a estrutura de coalizão e $x = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ é o vetor de utilidades, sendo $\mathbf{x}_i$ o valor obtido pelo jogador i.

Nos jogos *NTU*, diferentemente dos jogos *TU*, a utilidade não pode ser livremente redistribuída entre os membros da coalizão. Uma subclasse relevante de *NTU*s é a dos jogos hedônicos, em que os jogadores se importam unicamente com a identidade dos demais integrantes de sua coalizão. A definição a seguir é necessária para formalizar o conceito de um jogo hedônico: seja $N = \{1, \ldots, n\}$ o conjunto de jogadores. Para cada jogador $i \in N$, seja $\mathcal{N}_i = \{C \subseteq N \mid i \in C\}$ o conjunto de todas as coalizões que contêm i. Uma relação de preferência de i é uma relação binária $\succeq_i \subseteq \mathcal{N}_i \times \mathcal{N}_i$ que satisfaz as seguintes propriedades:

- **Completude**: para todos $C_1, C_2 \in \mathcal{N}_i$, temos $C_1 \succeq_i C_2$ ou $C_2 \succeq_i C_1$;
- **Reflexividade**: para todo $C \in \mathcal{N}_i$, temos $C \succeq_i C$;
- **Transitividade**: para todos $C_1, C_2, C_3 \in \mathcal{N}_i$, se $C_1 \succeq_i C_2$ e $C_2 \succeq_i C_3$, então $C_1 \succeq_i C_3$.

A notação $C_i \succeq C_j$ indica que a coalizão C_i é pelo menos tão preferida quanto C_j. De forma análoga, $C_i \succ C_j$ significa que C_i é estritamente preferida em relação a C_j, enquanto $C_i \sim C_j$ expressa que o jogador é indiferente entre as coalizões C_i e C_j. Com base nessa notação, pode-se agora definir formalmente um jogo hedônico que é representado por $J = (N, \succeq_1, \ldots, \succeq_n)$, onde $N = \{1, \ldots, n\}$ é o conjunto de jogadores, e para cada jogador $i \in N$, $\succeq_i$ é uma relação de

preferência em $\mathscr{N}_i$.

O resultado de um jogo hedônico é uma estrutura de coalizão *CS*, isto é, uma partição de N em subconjuntos não vazios. A teoria dos jogos cooperativos utiliza conceitos de solução para analisar as características das partições obtidas nos jogos. Muitos dos conceitos de solução focam na noção de estabilidade. Ou seja, analisam estruturas de coalizão em que os jogadores, de alguma forma, não desejam desviar de suas coalizões.

3.3.2 Conceitos de estabilidade

Uma das principais preocupações na teoria dos jogos cooperativos é compreender sob quais condições as coalizões formadas permanecem estáveis. Em outras palavras, busca-se identificar estruturas de coalizão nas quais nenhum jogador, ou grupo de jogadores, tenha incentivo de alterar sua posição atual. Essa análise é fundamental, pois a estabilidade reflete a viabilidade de uma determinada configuração social, econômica ou estratégica dentro do jogo.

Diversos conceitos de estabilidade são propostos na literatura. A seguir, são apresentados alguns dos mais relevantes no contexto dos jogos hedônicos:

- Seja um jogo hedônico $J = (N, \succeq_1, \ldots, \succeq_n)$, e uma estrutura de coalizão *CS*. Dizemos que *CS* é individualmente racional se, para todo $i \in N$, $CS_i \succeq_i \{i\}$. Ou seja, cada jogador prefere sua coalizão atual pelo menos tanto quanto ficar sozinho.
- Seja um jogo hedônico $J = (N, \succeq_1, \ldots, \succeq_n)$, e uma estrutura de coalizão *CS*. Dizemos que *CS* é Nash estável se, para todo $i \in N$ e toda coalizão $C \in CS \cup \{\emptyset\}$, $CS_i \succeq_i C \cup \{i\}$. Isto é, nenhum jogador tem incentivo a sair de sua coalizão atual para entrar em outra que pertence à *CS* ou ficar sozinho.
- Seja um jogo hedônico $J = (N, \succeq_1, \ldots, \succeq_n)$ e *CS* uma estrutura de coalizão. Dizemos que uma coalizão $C \subseteq N$ constitui um desvio de *CS* se, para todo jogador $i \in C$, tem-se $C \succ_i CS_i$, isto é, todos os jogadores de C preferem estritamente estar juntos em C do que na coalizão que lhes foi atribuída em *CS*. O *core* de um jogo hedônico é o conjunto de todas as estruturas de coalizão *CS* que não admitem desvio.

Com o objetivo de exemplificar estas definições, suponha o jogo formado pelos jogadores $N = \{1,2,3\}$ e o seguinte conjunto de preferências:

$$\begin{cases} 1 : \{1,3\} \succ_1 \{1,2,3\} \succ_1 \{1\} \succ_1 \{1,2\} \\ 2 : \{1,2\} \succ_2 \{1,2,3\} \succ_2 \{2\} \succ_2 \{2,3\} \\ 3 : \{1,3\} \succ_3 \{1,2,3\} \succ_3 \{3\} \succ_3 \{2,3\} \end{cases}$$

Neste cenário, é fácil verificar que a estrutura de coalizão $CS = \{\{1,3\},\{2\}\}$ pertence ao *core*. Dado que os jogadores 1 e 3 estão alocados na coalizão de maior preferência deles, as coalizões $\{1\}$, $\{3\}$, $\{1,2\}$, $\{2,3\}$ e $\{1,2,3\}$ não são desvio de CS. Por outro lado, esta estrutura de coalizão não é Nash estável, pois o jogador 2 irá preferir se juntar aos jogadores 1 e 3 e formar a estrutura de coalizão $CS' = \{\{1,2,3\}\}$. A estrutura de coalizão CS' é Nash estável, pois todos preferem ficar em {1, 2, 3} do que desviar para formar uma coalizão unitária. É importante observar também que CS' não pertence ao *core*, dado que a coalizão $\{1,3\}$ seria um desvio de CS'. Com este exemplo pode-se constatar que estabilidade de Nash não implica em pertencimento ao *core*. Ao mesmo tempo, pertencer ao *core* não implica em estabilidade de Nash.

Essas definições dão diferentes noções de estabilidade e são amplamente utilizadas na caracterização dos resultados de um jogo hedônico. A próxima seção trata de um tipo específico de jogo hedônico que é adotado neste trabalho: os jogos hedônicos aditivamente separáveis.

3.3.3 Jogos aditivamente separáveis

É dito que um jogo hedônico tem preferências aditivamente separáveis quando existe uma função $P_i : N \to \mathbb{R}, \forall i \in N$ de tal forma que $C_1 \succeq_i C_2$ se, e somente se, $\sum_{j \in C_1} P_i(j) \geq \sum_{j \in C_2} P_i(j)$. É interessante salientar que as funções P_i são representadas em alguns trabalhos por uma matriz M de dimensões $|N| \times |N|$ em que $P_i(j)$ é dado por $M[i,j]$ (CHALKIADAKIS *et al.*, 2011). Em jogos desse tipo, observa-se que a preferência de um jogador por uma coalizão está diretamente relacionada ao nível de afinidade que ele tem aos demais participantes que a compõem. É intuitivo verificar que jogos aditivamente separáveis fazem parte dos jogos *NTU*. Por exemplo, dada uma partição CS qualquer, pode-se definir a utilidade de um jogador $i \in C \in CS$ por $U(i) = \sum_{j \in C} P_i(j)$. Além disso, também pode-se dizer que $v(C) = \sum_{i \in C} U(i)$. Dado que cada jogador tem uma utilidade $U(i)$ fixa que é definida naturalmente pela formação da estrutura de coalizão CS, não é possível distribuir $v(C)$ de qualquer forma entre os jogadores.

Dimitrov *et al.* (2006) propuseram duas classes de jogos hedônicos aditivamente

separáveis: apreciação de amigos e aversão à inimigos. Nestes jogos, cada jogador i possui um conjunto de amigos A_i e um conjunto de inimigos I_i, de modo que $N - \{i\} = A_i \cup I_i$ e $A_i \cap I_i = \emptyset$. Desta forma, pode-se definir a preferência dos jogadores pelas coalizões da seguinte forma:

- **Jogos de apreciação dos amigos**: $C_1 \succ_i C_2$ se, e somente se, $|C_1 \cap A_i| > |C_2 \cap A_i|$ ou $|C_1 \cap A_i| = |C_2 \cap A_i|$ e $|C_1 \cap I_i| < |C_2 \cap I_i|$. Ou seja, o jogador prioriza coalizões que contenham mais amigos, e em caso de empate, prefere aquelas com menos inimigos.
- **Jogos de aversão aos inimigos**: são semelhantes aos jogos de apreciação dos amigos, mas, neste cenário, os jogadores tentam minimizar a convivência com inimigos. Formalmente, $C_1 \succ_i C_2$ se, e somente se, $|C_1 \cap I_i| < |C_2 \cap I_i|$ ou $|C_1 \cap I_i| = |C_2 \cap I_i|$ e $|C_1 \cap A_i| > |C_2 \cap A_i|$. Assim, o jogador prefere estar em coalizões com menos inimigos; em caso de empate, ele escolhe a coalizão que contém mais amigos.

Dimitrov *et al.* (2006) também citam que os jogos propostos são aditivamente separáveis. Em particular, no jogo de apreciação dos amigos, é possível definir a função P_i da seguinte forma: se $j \in A_i$, então $P_i(j) = n$. Por outro lado, se $j \in I_i$ então $P_i(j) = -1$. Portanto, dada a existência da função P_i, fica claro que o jogo de apreciação dos amigos pertence a classes de jogos hedônicos aditivamente separáveis. O autor também cita que o jogo de apreciação dos amigos pode ser modelado como um grafo direcionado em que os jogadores são os vértices do grafo e existe uma aresta de um jogador i para j se $j \in A_i$. O Algoritmo 9 mostra como gerar um digrafo a partir deste jogo.

Algoritmo 9: Construção do digrafo no jogo de apreciação dos amigos

```
Input: Conjuntos A_i, ∀i ∈ N.
Output: Um digrafo D = (V, E) representando as relações de amizade.
Inicialize D ← (N, ∅), onde N é o conjunto de jogadores e E o conjunto de arestas;
foreach i ∈ N do
    foreach j ∈ N \ {i} do
        if j ∈ F_i then
            E ← E ∪ {(i, j)};
        end
    end
end
return D;
```

Outra contribuição de Dimitrov *et al.* (2006), é a definição do conceito de *core* estrito para esses jogos. Seja um jogo hedônico $J = (N, \succeq_1, \ldots, \succeq_n)$, e CS uma estrutura de coalizão, isto é, uma partição de N. Dizemos que uma coalizão $C \subseteq N$ constitui um desvio fraco de CS se,

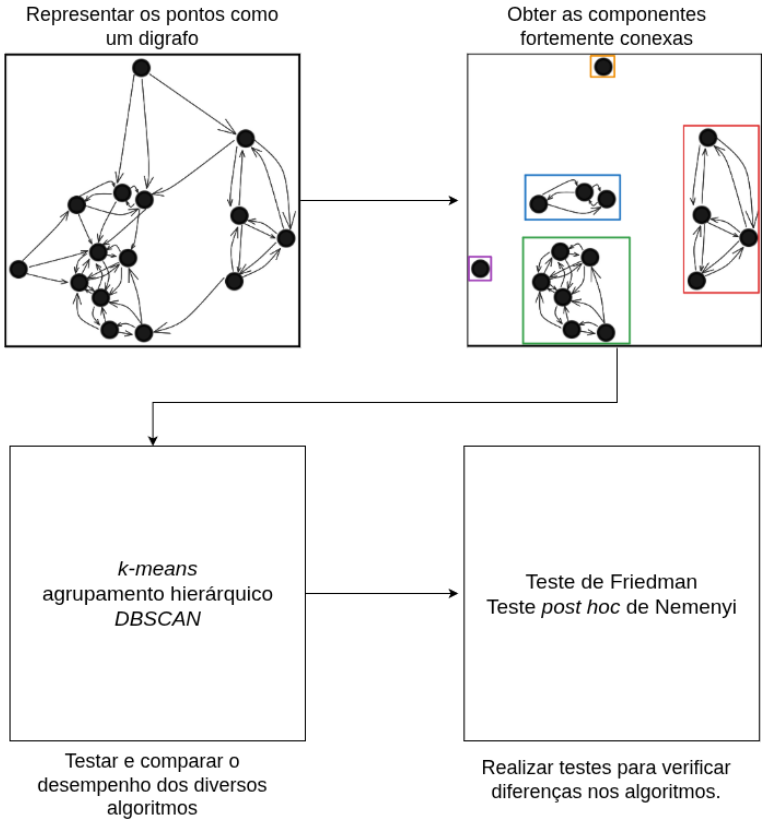
para todo jogador $i \in C$, tem-se $C \succeq_i CS_i$ e existe pelo menos um jogador $j \in C$ tal que $C \succ_j CS_j$, onde CS_i denota a coalizão de CS que contém o jogador i. O *core* estrito de um jogo hedônico é o conjunto de todas as estruturas de coalizão CS que não admitem desvio fraco.

Dimitrov *et al.* (2006) mostram que encontrar um elemento do *core* no jogo de aversão aos inimigos é um problema NP-Difícil. Por outro lado, de acordo com os autores, encontrar o *core* estrito em jogos de apreciação dos amigos pode ser feito em tempo polinomial. Para tanto, os autores mostram que encontrar o *core* estrito do jogo de apreciação dos amigos é equivalente à encontrar os componentes fortemente conexos do digrafo do jogo. Diante desses resultados, surge a seguinte questão: ao modelar o problema de *clustering* como um jogo hedônico aditivamente separável, seria possível utilizar o conceito de *core* estrito como uma ferramenta para identificar agrupamentos nos dados? Nesse contexto, explorar essa relação pode fornecer uma nova perspectiva para o desenvolvimento de algoritmos de *clustering* baseados em teoria dos jogos.

4 METODOLOGIA

Este capítulo descreve a metodologia adotada para mapear o problema de *clustering* em um jogo de apreciação de amigos, além de apresentar as métricas utilizadas para avaliar a abordagem proposta e compará-la com algoritmos clássicos da literatura. O fluxo completo da metodologia pode ser visualizado na Figura 10.

Figura 10 – Metodologia usada do trabalho.



Fonte: Elaborado pelo autor.

A Seção 4.1 detalha o método proposto, que consiste, de forma resumida, em transformar um *dataset* em um jogo de apreciação de amigos representado por um digrafo. A partir dos conjuntos de amigos de um ponto p, constrói-se o digrafo correspondente e, em seguida, aplica-se um algoritmo para encontrar as componentes fortemente conexas (TARJAN, 1972; SHARIR, 1981). Esse processo gera uma partição no conjunto de dados, pertencente ao *core* estrito do jogo, que é então utilizada como solução para o problema de *clustering*. A estratégia de construção do jogo é descrita em detalhes na referida seção.

Por fim, a Seção 4.2 descreve as métricas utilizadas para avaliar o desempenho dos algoritmos, bem como os testes estatísticos empregados para verificar a existência de diferenças estatisticamente significativas entre eles.

4.1 Mapeamento do problema de *clustering* em um jogo de apreciação de amigos

A abordagem proposta considera os pontos das bases de dados como jogadores de um jogo hedônico de apreciação de amigos definido por Dimitrov *et al.* (2006). Ou seja os jogadores são um conjunto de pontos de uma base de dados $B = \{p_1, p_2, \ldots, p_n\}$ em que cada ponto $p \in B$ tem m atributos.

Após a etapa de definição dos jogadores, o próximo passo é definir o conjunto A_p de cada jogador $p \in B$ com o intuito de formar o grafo definido no trabalho de Dimitrov *et al.* (2006). Este passo é fundamental, pois, dependendo da definição de A_p, existe a possibilidade dos componentes fortemente conexos do grafo gerado não representarem uma boa solução de *clustering*. Para tal, foram escolhidas duas abordagens para a definição de A_p, cada uma gerou um algoritmo distinto. A primeira abordagem consiste em escolher os k-vizinhos mais próximos de um ponto na base de dados. Ou seja, dada uma medida de dissimilaridade qualquer d e um ponto $p \in B$, dizemos que o conjunto A_p é dado pelos k-vizinhos mais próximos de p, denominado por $kNN(p)$, sujeito à: $kNN(p) \subseteq B$, $|kNN(p)| = k$ e $\forall q \in B - kNN(p)$, tem-se a Equação 4.1.

$$d(p,q) \geq \max_{r \in kNN(p)} d(p,r) \tag{4.1}$$

Esse método, denominado CEAC, consiste na construção de um digrafo a partir do *dataset*, no qual cada ponto é representado como um vértice. Para cada ponto p, são adicionadas arestas direcionadas ligando-o a seus k vizinhos mais próximos. Em seguida, o algoritmo identifica as componentes fortemente conexas desse digrafo. O Algoritmo 10 formaliza essa abordagem, fornecendo uma visão clara de seu funcionamento.

Algoritmo 10: CEAC

Input: *Dataset B* e a quantidade de vizinhos k

Output: As componentes fortemente conexas C, cada uma representando um *cluster*

$D \leftarrow$ digrafo em que cada vértice representa um ponto do *dataset B*;

foreach *cada ponto* $p_i \in B$ **do**

  $A \leftarrow$ conjunto dos k vizinhos mais próximos de p_i;

  foreach *cada ponto* $p_j \in A$ **do**

    adicionar aresta direcionada de p_i para p_j em D;

  end

end

$C \leftarrow$ componentes fortemente conexas do digrafo D;

return C;

Em relação à complexidade, a construção do digrafo a partir do *dataset* pode ser realizada de forma eficiente. Para identificar os k vizinhos mais próximos de um ponto p, pode-se utilizar uma estrutura de dados conhecida como árvore k-d (BENTLEY, 1975). Nesse caso, a complexidade da busca é $O(\log n + k)$, onde n representa o número de pontos e k o número de vizinhos considerados. Ao realizar esse procedimento para todos os pontos do *dataset*, a complexidade total torna-se $O(n(\log n + k))$, que pode ser escrita como $O(n \log n + nk)$. Após a construção do digrafo, é necessário calcular suas componentes fortemente conexas. Esse procedimento pode ser realizado em tempo $O(n + nk)$, uma vez que o número de arestas é proporcional a nk. Novamente, para $k \ll n$, essa etapa possui complexidade aproximada de $O(n)$. Dessa forma, considerando todas as etapas do processo, a complexidade total do algoritmo é $O(n \log n + nk)$, que, para valores de k significativamente menores que n, pode ser simplificada para $O(n \log n)$.

A segunda abordagem proposta consiste em selecionar os k vizinhos mais próximos de um ponto, desde que estejam dentro de um raio r. Essa abordagem é bastante semelhante ao CEAC, diferindo apenas pelo fato de que os k vizinhos considerados são restritos ao raio r. Essa variação foi denominada *Core* Estrito Aplicado à *Clustering* ++ (CEAC++), e o Algoritmo 11 formaliza essa ideia.

Algoritmo 11: CEAC++

Input: *Dataset B*, a quantidade de vizinhos k e um raio r

Output: As componentes fortemente conexas C, em que cada *SCC* representa um *cluster*

$D \leftarrow$ digrafo em que cada vértice representa um ponto do *dataset B*;

foreach *cada ponto* $p_i \in B$ **do**

  $A \leftarrow$ conjunto dos k vizinhos mais próximos de p_i;

  foreach *cada ponto* $p_j \in A$ **do**

    if $d(p_i, p_j) \leq r$ **then**

      adicionar aresta direcionada de p_i para p_j em D;

    end

  end

end

$C \leftarrow$ componentes fortemente conexas do digrafo D;

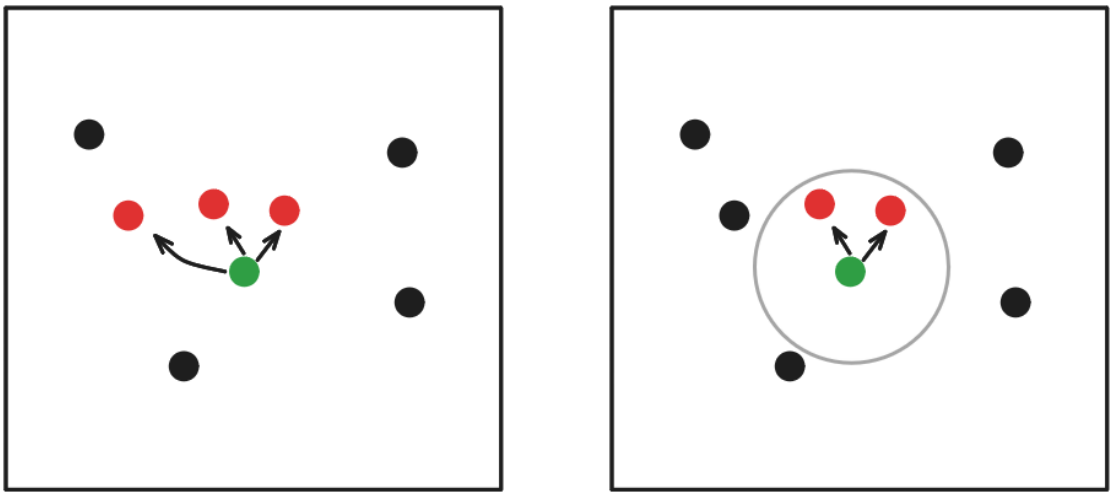
return C;

Em relação à complexidade, novamente, a busca pelos k vizinhos mais próximos de um ponto p pode ser realizada em $O(\log n + k)$ utilizando uma árvore k-d. Em seguida, a verificação de quais desses vizinhos estão dentro de um raio r possui custo $O(k)$. Dessa forma, a complexidade total da consulta permanece $O(\log n + k)$. Ao executar esse procedimento para todos os n pontos do conjunto de dados, obtém-se uma complexidade total de $O(n \log n + nk)$. Encontrar as componentes fortemente conexas do digrafo pode ser feito em $O(n + nk)$. Portanto, a complexidade total do algoritmo é $O(n \log n + nk)$, que para $k \ll n$ pode ser simplificada para $O(n \log n)$.

A Figura 11 ilustra a diferença entre o CEAC e o CEAC++. No exemplo apresentado, considerando $k = 3$ e o raio r apresentado na figura, é possível observar claramente a distinção entre os algoritmos. Enquanto no CEAC o ponto verde se conecta automaticamente aos seus três vizinhos mais próximos, no CEAC++ isso não ocorre, pois o terceiro vizinho mais próximo encontra-se fora do raio r, impedindo a criação da conexão.

Nota-se ainda que, para valores muito elevados de r, ambos os algoritmos tendem a apresentar comportamentos semelhantes, uma vez que, para esses valores, todos os k vizinhos mais próximos passam a estar contidos dentro do raio r. Por outro lado, para valores muito elevados de k, o CEAC++ apresenta um comportamento interessante, tornando-se bastante semelhante ao *DBSCAN* com o parâmetro $minPts = 1$. Isso ocorre porque, quando k é muito

Figura 11 – Diferença entre o CEAC e o CEAC++.



Fonte: Elaborado pelo autor.

grande, o CEAC++ tende a considerar todos os pontos dentro do raio r. Como a distância entre dois pontos é simétrica, o grafo gerado torna-se não direcionado, e o algoritmo passa a retornar as componentes conexas desse grafo. Ao analisar o comportamento do *DBSCAN*, observa-se que, quando $minPts = 1$, seu funcionamento é bastante semelhante, também resultando nas componentes conexas de um grafo. A principal diferença ocorre na presença de pontos extremamente isolados, que não possuem nenhum vizinho dentro do raio r.

Esse comportamento demonstra que o CEAC++ é mais flexível que o CEAC, pois permite variar simultaneamente os parâmetros k e r, possibilitando a construção de dígrafos com estruturas mais diversas.

Isso não ocorre na abordagem original do CEAC, sendo essa a principal vantagem do CEAC++ em relação ao CEAC. Por outro lado, o CEAC apresenta a vantagem de utilizar apenas um único parâmetro inteiro, o que torna o processo de ajuste de parâmetros mais simples quando comparado ao CEAC++, que depende de dois parâmetros (um inteiro e um real). Além disso, no CEAC não é necessário possuir um grande conhecimento prévio do *dataset*. Já no CEAC++, por depender de um raio r, é necessário ter ao menos uma noção das distâncias entre os pontos do conjunto de dados.

Por fim, convém destacar que ambos os algoritmos propostos, conforme os tipos de agrupamentos apresentados na Seção 3.1.1, realizam agrupamentos de natureza particional, exclusiva e completa.

4.2 Métodos de Avaliação

Um dos objetivos desta pesquisa foi avaliar a eficácia dos algoritmos propostos. Para isso, os algoritmos foram testados em bases de dados artificiais (PARMAR, 2019) e em bases de dados de problemas reais disponibilizadas no repositório UCI (ASUNCION *et al.*, 2007). Nesse contexto, utilizam-se métodos de avaliação externa para verificar a qualidade das partições geradas por algoritmos de *clustering*. Tais métodos empregam informações previamente conhecidas sobre os dados — como os rótulos de classe —, que indicam o grupo real ao qual cada ponto pertence. Dessa forma, torna-se possível comparar o agrupamento gerado por um algoritmo de *clustering* com a partição real dos dados.

O *Rand Index* (*RI*) é um método de avaliação externo amplamente utilizado. Dadas duas partições quaisquer X e Y, o *RI* entre elas é definido pela Equação 4.2.

$$RI = \frac{TP + TN}{\binom{n}{2}} \tag{4.2}$$

Em que TP é a quantidade de pares de pontos que pertencem ao mesmo grupo em X e em Y e TN é dado pela quantidade de pares de pontos que estão em grupos diferentes em X e em Y. Pela definição, é fácil verificar que o *RI* varia entre 0 e 1. O *Adjusted Rand Index* (*ARI*) é uma versão melhorada do *RI* que pode assumir valores entre -1 e 1. O valor de 1 indica duas partições idênticas, 0 corresponde a um resultado que poderia ser obtido por uma partição gerada de forma aleatória e -1 indica duas partições que são completamente diferentes (HUBERT; ARABIE, 1985). Por ser um índice mais utilizado na literatura, este trabalho adotou o *ARI* para verificar a qualidade dos agrupamentos gerados. O *ARI* possui implementação em *Python* na biblioteca *Scikit-Learn* (HAO; HO, 2019).

Além de avaliar o CEAC e o CEAC++ neste trabalho por meio do *ARI* e dos rótulos reais dos dados, também foram avaliados, utilizando as mesmas bases de dados, algoritmos clássicos de *clustering*. Para possibilitar a comparação entre os algoritmos, foi adotado o teste de Friedman (DEMŠAR, 2006), amplamente utilizado na comparação de múltiplos algoritmos de *machine learning* em diferentes bases de dados.

O teste de Friedman é um teste estatístico não paramétrico utilizado para comparar três ou mais grupos. No contexto de algoritmos, de acordo com Demšar (2006), considera-se a hipótese nula: H_0, em que todos os algoritmos apresentam desempenho equivalente, e a hipótese alternativa H_A, em que pelo menos um algoritmo apresenta desempenho diferente dos demais.

O teste classifica os algoritmos em rankings com base em seu desempenho em cada base de dados e, em seguida, calcula a média desses rankings. A partir disso, obtém-se um valor-p. Caso esse valor seja menor que o nível de significância adotado, neste trabalho $\alpha = 0{,}05$, rejeita-se a hipótese nula, indicando a existência de diferenças estatisticamente significativas em pelo menos um dos algoritmos.

Em caso de rejeição da hipótese nula no teste de Friedman, indicando a existência de diferenças estatisticamente significativas em pelo menos um dos algoritmos avaliados, seguindo a metodologia proposta por Demšar (2006), foi aplicado o teste *post hoc* de Nemenyi. Esse teste realiza comparações par a par entre os algoritmos com base nos rankings médios obtidos no teste de Friedman, com o objetivo de identificar quais pares apresentam diferenças estatisticamente significativas em seu desempenho.

Para cada par de algoritmos, é calculada uma estatística baseada na diferença entre seus rankings médios. A partir dessa estatística, obtém-se o valor-p associado à comparação. Quando o valor-p é menor que o nível de significância adotado de $\alpha = 0.05$, conclui-se que existe uma diferença estatisticamente significativa entre os algoritmos comparados.

5 RESULTADOS

Este capítulo apresenta uma análise comparativa do desempenho do CEAC e do CEAC++ em relação a algoritmos clássicos da literatura. A Seção 5.1 mostra a configuração experimental usada para realizar os testes; a Seção 5.2 descreve e discute os resultados obtidos pelo CEAC e CEAC++ em *datasets* artificiais, enquanto a Seção 5.3 realiza a mesma análise considerando os *datasets* reais.

5.1 Configuração experimental

Foram selecionados seis algoritmos clássicos da literatura como métodos de comparação: *k-means*, *DBSCAN* e variações de agrupamento hierárquico, as quais diferem quanto ao critério adotado para o cálculo da distância entre dois grupos distintos, conforme descrito na Seção 3.1.3. Esses métodos estão entre os mais tradicionais da área, sendo amplamente utilizados e validados em diferentes tipos de *datasets*, o que justifica sua escolha devido ao seu histórico consolidado em diversas aplicações. O CEAC e o CEAC++ foram implementados na linguagem de programação *Python*, utilizando a biblioteca *Scikit-Learn* (HAO; HO, 2019). Para os algoritmos clássicos, foi utilizada a implementação padrão disponível nessa mesma biblioteca.

O objetivo dos experimentos foi, para cada algoritmo, determinar o(s) parâmetro(s) que maximizasse(m) o valor do *ARI*. Para isso, os testes foram conduzidos de forma específica para cada método. No caso do CEAC e dos algoritmos de agrupamento hierárquico, o procedimento consistiu em variar incrementalmente o parâmetro k até seu valor máximo, correspondente ao número de pontos do *dataset*, buscando identificar o valor que produzisse o maior *ARI*.

Para o *k-means*, essa abordagem não pôde ser aplicada diretamente, uma vez que na sua versão clássica, os k centroides iniciais são escolhidos de forma aleatória, por isso o *k-means* clássico é não determinístico. Assim, para cada valor de k, o algoritmo foi executado 30 vezes, sendo calculada a média do *ARI*. Esse processo foi repetido com incrementos sucessivos em k, até a identificação do valor que maximizou a média obtida.

Já para o *DBSCAN* e o CEAC++, que possuem um parâmetro inteiro e outro contínuo, não é viável realizar uma busca incremental exaustiva no espaço de parâmetros. Para contornar essa limitação, foi utilizada a biblioteca *Optuna* (AKIBA *et al.*, 2019), em *Python*, para realizar a otimização dos parâmetros.

Por fim, destaca-se que, diferentemente dos demais algoritmos, o *DBSCAN* pode

classificar determinados pontos como ruído. Para evitar que uma classe inteira seja rotulada como ruído, o que poderia inflar artificialmente o valor do *ARI*, foi utilizada a abordagem proposta por Bryant e Cios (2017), na qual cada ponto identificado como ruído é considerado como pertencente a uma classe unitária.

Todos os testes foram feitos em um *laptop* com processador AMD Ryzen 5 5500U (12) @ 4.06 GHz, com 8 GB de memória RAM e no sistema operacional Debian GNU/Linux forky/sid (forky) x86_64.

5.2 Desempenho nos *datasets* artificiais

Para a análise, foram selecionados 15 *datasets* artificiais (PARMAR, 2019). A principal característica desses *datasets* é que todos possuem exatamente duas dimensões. Em relação ao tamanho, há uma variação significativa: o menor conjunto contém 240 pontos, enquanto o maior possui 3100 pontos. A quantidade de grupos também apresenta grande variabilidade. Alguns *datasets* contêm apenas dois grupos distintos, enquanto outros possuem mais de 30 grupos, sendo que um deles chega a 40 grupos. A Tabela 1 apresenta uma breve descrição dos *datasets* artificiais. Essa descrição tem como objetivo evidenciar a diversidade dos conjuntos utilizados nos testes, tanto em termos de tamanho quanto de número de grupos.

Tabela 1 – Descrição dos *datasets* artificiais.

	dimensões	tamanho	grupos
3MC	2	400	3
Aggregation	2	788	7
Compound	2	399	6
D31	2	3100	31
Flame	2	240	2
Fourty	2	1000	40
Jain	2	373	2
Longsquare	2	900	6
Pathbased	2	300	3
Pearl	2	266	3
R15	2	600	15
Rings	2	1000	3
Spherical_6_2	2	300	6
Spiral	2	312	3
Target	2	770	6

A Tabela 2 apresenta o desempenho dos algoritmos nos *datasets* artificiais segundo a métrica *ARI*. De modo geral, observa-se que todos os algoritmos obtiveram bom desempenho na maioria dos conjuntos, com poucas exceções pontuais. Esse resultado pode ser explicado,

principalmente, pela simplicidade dos *datasets*, visto que todos são bidimensionais e apresentam estruturas relativamente bem definidas, o que tende a favorecer métodos de agrupamento. Para facilitar a visualização, os melhores resultados obtidos em cada *dataset* estão destacados em negrito.

Tabela 2 – Desempenho dos algoritmos com *datasets* artificiais utilizando a métrica *ARI*

		Hierárquico aglomerativo						
	k-means	*ward*	*complete*	*single*	*average*	*DBSCAN*	CEAC	CEAC++
3MC	0,707	**1,000**	0,678	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**
Aggregation	0,763	0,989	0,782	0,812	**1,000**	0,992	0,809	0,812
Compound	0,706	0,775	0,841	0,945	0,849	0,945	0,944	**0,949**
D31	0,839	0,920	0,924	0,643	**0,931**	0,907	0,568	0,724
Flame	0,462	0,537	0,511	0,912	0,690	**0,966**	0,063	0,912
Fourty	0,881	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	0,994	**1,000**
Jain	0,309	0,515	0,779	0,941	0,779	0,941	**1,000**	**1,000**
Longsquare	0,760	0,852	0,785	0,948	0,902	**0,959**	0,941	0,951
Pathbased	0,463	0,485	0,346	0,612	0,669	0,703	0,329	**0,739**
Pearl	0,631	0,649	0,616	**1,000**	0,620	**1,000**	**1,000**	**1,000**
R15	0,865	0,982	0,986	0,951	**0,989**	**0,989**	0,865	0,951
Rings	0,441	0,453	0,378	0,440	0,439	**0,518**	0,355	0,440
Spherical_6_2	0,842	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**
Spiral	0,115	0,130	0,093	**1,000**	0,162	**1,000**	**1,000**	**1,000**
Target	0,655	0,658	0,628	**1,000**	0,664	**1,000**	**1,000**	**1,000**

Os métodos utilizados para a obtenção dos parâmetros responsáveis por esses resultados são apresentados na Seção 5.1, enquanto os valores obtidos encontram-se nas Tabelas 3 e 4.

Apesar do bom desempenho geral, nota-se variação entre os algoritmos em determinados conjuntos mais desafiadores, como aqueles com formatos não convexos ou maior número de grupos. Ainda assim, em diversos casos, múltiplos algoritmos atingiram valores de *ARI* próximos ou iguais a 1, indicando recuperação quase perfeita da estrutura original dos dados.

Para verificar se as diferenças observadas são estatisticamente significativas, foi aplicado o teste de Friedman. O teste resultou em um valor-p de aproximadamente $1{,}750 \times 10^{-7}$, substancialmente inferior ao nível de significância adotado de $\alpha = 0{,}05$. Dessa forma, rejeita-se a hipótese nula de que todos os algoritmos apresentam desempenho equivalente, concluindo-se que há diferença estatisticamente significativa entre eles nos *datasets* artificiais avaliados.

Diante dessa diferença global, foi aplicado o teste *post hoc* de Nemenyi para realizar comparações par a par entre os algoritmos. Os resultados são apresentados no mapa de calor da Figura 12. Conforme descrito na Seção 4.2, valores-p próximos de zero indicam maior diferença entre os métodos, e valores-p inferiores a $\alpha = 0{,}05$ apontam diferença estatisticamente

Tabela 3 – Parâmetros usados nos algoritmos CEAC, *k-means* e agrupamento hierárquico com *datasets* artificiais.

		Hierárquico aglomerativo				
	k-means	*ward*	*complete*	*single*	*average*	CEAC
3MC	3	3	5	3	3	9
Aggregation	5	6	4	7	27	6
Compound	3	4	8	8	56	9
D31	38	31	31	32	194	5
Flame	2	3	5	3	12	3
Fourty	53	40	40	40	40	12
Jain	2	2	2	2	5	9
Longsquare	6	10	6	9	26	7
Pathbased	3	3	3	9	29	4
Pearl	5	4	7	7	3	7
R15	22	15	16	15	33	6
Rings	11	13	18	16	162	5
Spherical_6_2	7	6	6	6	6	8
Spiral	19	32	41	27	3	4
Target	5	5	11	9	6	7

Tabela 4 – Parâmetros usados nos algoritmos *DBSCAN* e CEAC++ com *datasets* artificiais.

	DBSCAN	CEAC++
3MC	'minPts': 9; 'ε': 1,0460201020523248	'k': 20; 'r': 1,0482778840406901
Aggregation	'minPts': 7; 'ε': 1,385448310202904	'k': 14; 'r': 0,9406827776925093
Compound	'minPts': 2; 'ε': 1,4913811166047735	'k': 6; 'r': 1,4902818265684497
D31	'minPts': 69; 'ε': 1,3300804848931733	'k': 15; 'r': 0,3732789501785986
Flame	'minPts': 4; 'ε': 0,9249887876285694	'k': 21; 'r': 0,8334192591644776
Fourty	'minPts': 11; 'ε': 1,3436040166655379	'k': 26; 'r': 1,3596460526803644
Jain	'minPts': 1; 'ε': 2,4839162760564175	'k': 8; 'r': 2,8745433572238444
Longsquare	'minPts': 5; 'ε': 1,1385197897872876	'k': 8; 'r': 1,1299153050034731
Pathbased	'minPts': 5; 'ε': 1,7892743385585652	'k': 4; 'r': 2,0145230999854187
Pearl	'minPts': 9; 'ε': 0,04159976412054117	'k': 12; 'r': 0,03280946274039073
R15	'minPts': 30; 'ε': 0,7000411069257589	'k': 15; 'r': 0,32726584689394167
Rings	'minPts': 8; 'ε': 0,9589210283278574	'k': 23; 'r': 0,6439692990981364
Spherical_6_2	'minPts': 1; 'ε': 1,1204774498879362	'k': 13; 'r': 1,13468958749388
Spiral	'minPts': 3; 'ε': 1,2500082000465313	'k': 10; 'r': 1,2437086419539958
Target	'minPts': 2; 'ε': 0,2591504996692918	'k': 13; 'r': 0,2940470225300929

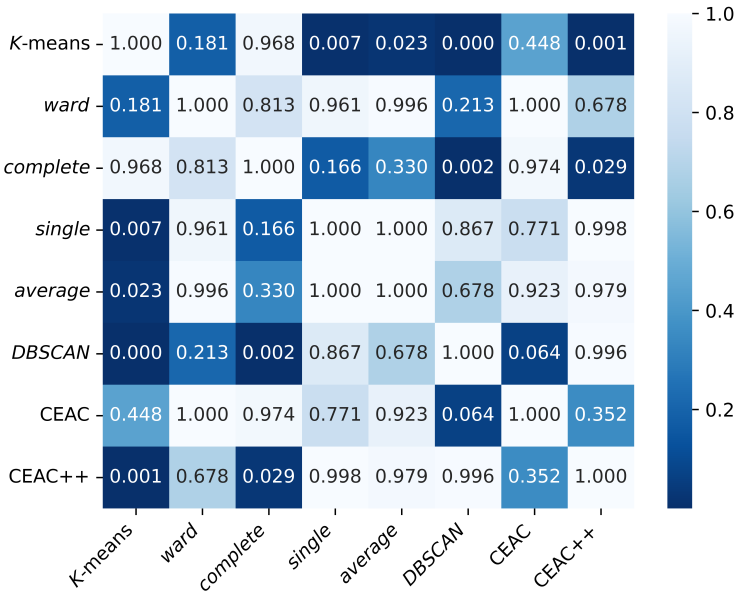
significativa.

Observa-se, inicialmente, uma diferença estatística expressiva entre o *k-means* e os métodos *single*, *average*, *DBSCAN* e CEAC++. Esse resultado é coerente com o desempenho apresentado na tabela, em que o *k-means* obteve resultados inferiores em diversos *datasets*. Além disso, também se nota uma diferença entre o hierárquico **complete** com o *DBSCAN* e

CEAC++. Ao observar os resultados apresentados na tabela, essa diferença pode ser explicada pelo desempenho superior do *DBSCAN* e do CEAC++ em comparação com o agrupamento hierárquico *complete*.

Para os demais pares de algoritmos, as diferenças observadas são menos pronunciadas, o que é compatível com a relativa simplicidade estrutural dos *datasets* artificiais analisados.

Figura 12 – Teste *Post-Hoc* de Nemenyi sobre os resultados dos *datasets* artificiais.



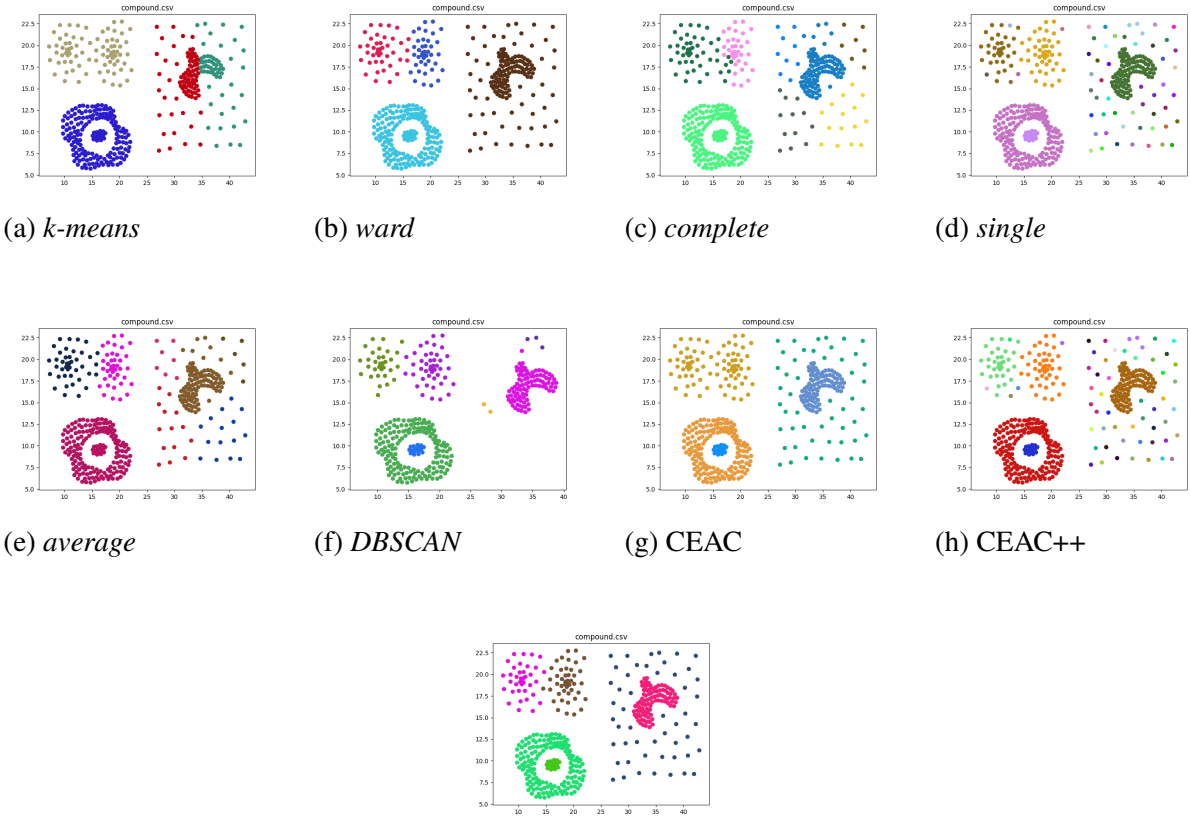
Fonte: Elaborado pelo autor.

Como os *datasets* artificiais são todos bidimensionais, é possível visualizar graficamente os *clusters* produzidos pelos algoritmos. A Figura 13 apresenta uma comparação entre os métodos no *dataset* Compound, que é particularmente interessante por conter *clusters* convexos e não convexos, além de apresentar diferenças significativas de densidade entre as regiões. Essas características o tornam adequado para evidenciar as limitações e potencialidades dos diferentes algoritmos.

Observa-se que o *k-means* apresentou dificuldades consideráveis nesse *dataset*, sobretudo devido à presença de *clusters* não convexos, uma vez que o algoritmo tende a particionar os dados em regiões aproximadamente esféricas. O *DBSCAN*, por sua vez, identificou a região menos densa como ruído, o que impactou diretamente sua partição final.

Já o CEAC++ e o método hierárquico com ligação *single* demonstraram dificuldade em agrupar adequadamente a região mais dispersa. Entre os métodos analisados, o CEAC foi o que apresentou, visualmente, o melhor desempenho nessa área específica, conseguindo

Figura 13 – *Dataset* Compound.



(a) *k-means* (b) *ward* (c) *complete* (d) *single*

(e) *average* (f) *DBSCAN* (g) CEAC (h) CEAC++

(i) Partição real

Fonte: Elaborado pelo autor.

representar de forma mais consistente a estrutura desses dados, embora apresente falhas ao agrupar alguns *clusters* próximos.

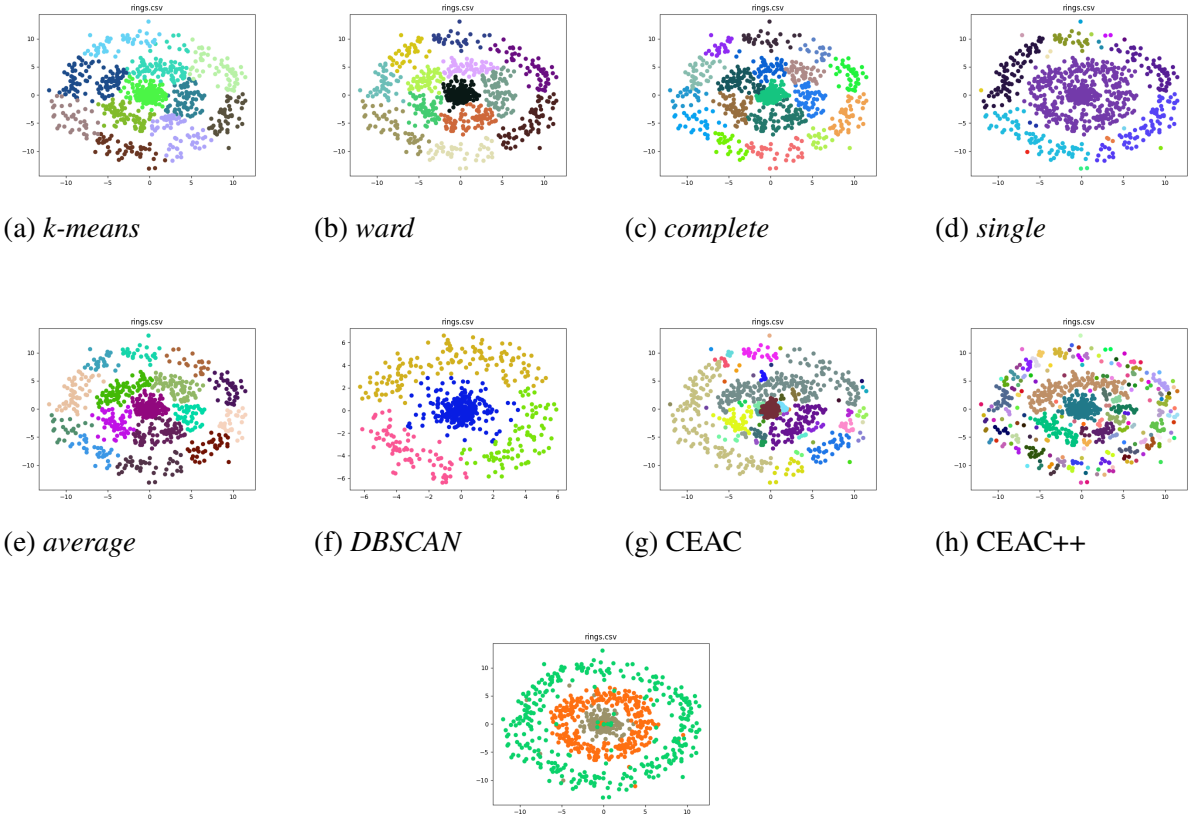
Outro *dataset* interessante de ser analisado é o Rings, cujos resultados são apresentados na Figura 14. Entre os *datasets* artificiais avaliados, este foi o único em que nenhum dos algoritmos obteve desempenho elevado, o que evidencia sua maior complexidade estrutural.

A dificuldade está relacionada, principalmente, ao formato dos grupos, que apresentam estruturas concêntricas e regiões com diferentes níveis de separação, tornando a tarefa de agrupamento mais desafiadora.

Um caso particularmente relevante é o do *DBSCAN*, que chegou a classificar uma classe inteira como ruído. Embora tenham sido adotadas medidas para mitigar esse problema, a situação ainda ocorreu, demonstrando a sensibilidade do algoritmo à escolha de parâmetros e às variações de densidade presentes nos dados.

Também é relevante analisar o *dataset* Pathbased, na Figura 15, no qual os algoritmos apresentaram, de modo geral, resultados intermediários. A Figura 15 ilustra de forma mais clara o comportamento de cada método nesse *dataset*. Observa-se, inicialmente, que a maioria dos

Figura 14 – *Dataset* Rings.



(a) *k-means* (b) *ward* (c) *complete* (d) *single*

(e) *average* (f) *DBSCAN* (g) CEAC (h) CEAC++

(i) Partição real

Fonte: Elaborado pelo autor.

algoritmos teve dificuldade em identificar corretamente o *cluster* mais externo. A única exceção foi o CEAC++, que conseguiu agrupá-lo de maneira adequada. No entanto, apesar desse bom desempenho na região externa, o CEAC++ apresentou dificuldades na identificação dos *clusters* internos, evidenciando o desafio imposto pela estrutura global do conjunto de dados.

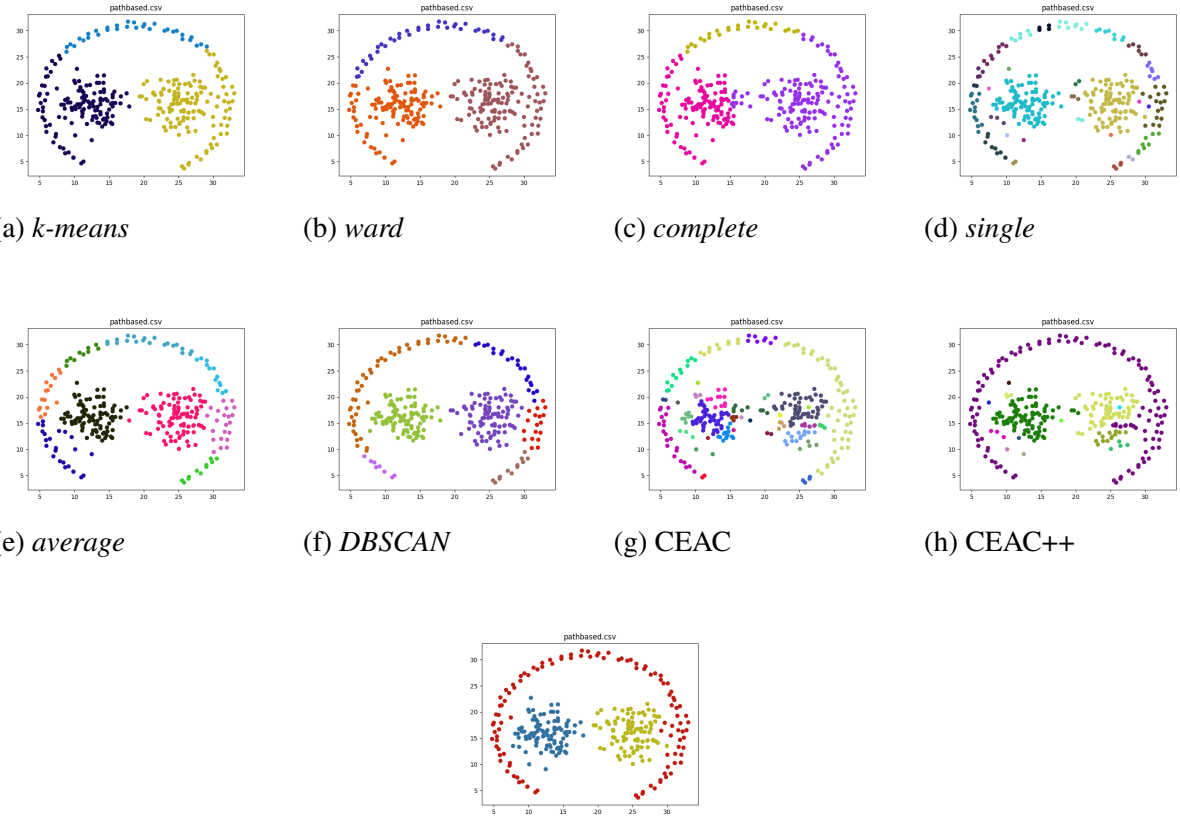
Também vale ressaltar o *dataset* Jain, na Figura 16, no qual apenas o CEAC e o CEAC++ conseguiram realizar o agrupamento perfeito, demonstrando o potencial desses algoritmos.

Por último, a Figura 17 apresenta outros *datasets* em que o CEAC++ obteve desempenho perfeito, ou seja, não errou nenhum agrupamento, reforçando o potencial do método em identificar corretamente estruturas de agrupamento mesmo em cenários mais desafiadores.

5.3 Desempenho nos *datasets* reais

Em relação aos *datasets* reais, foram selecionados 12 conjuntos de dados amplamente utilizados na literatura (ASUNCION *et al.*, 2007). Esses conjuntos abrangem diferentes

Figura 15 – *Dataset* Pathbased.



(a) *k-means* (b) *ward* (c) *complete* (d) *single*

(e) *average* (f) *DBSCAN* (g) CEAC (h) CEAC++

(i) Partição real

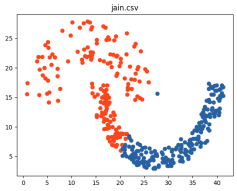
Fonte: Elaborado pelo autor.

áreas do conhecimento, como genética, química, biologia e reconhecimento de padrões. Diferentemente dos *datasets* artificiais, os dados reais apresentam maior complexidade estrutural e variabilidade intrínseca. Além disso, a dimensionalidade varia de 4 a 4096 atributos, o que eleva consideravelmente a dificuldade enfrentada pelos algoritmos, sobretudo em cenários de alta dimensionalidade, nos quais os efeitos da maldição da dimensionalidade podem comprometer a qualidade das partições obtidas. A Tabela 5 fornece uma visão mais detalhada da estrutura e da variação desses *datasets*, evidenciando sua maior complexidade em comparação aos *datasets* artificiais utilizados nos experimentos.

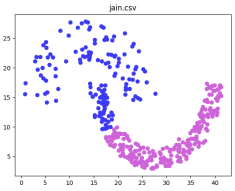
Em relação aos resultados obtidos com a métrica *ARI*, a Tabela 6 apresenta o desempenho dos algoritmos nos *datasets* reais. Observa-se, de modo geral, um desempenho inferior quando comparado aos *datasets* artificiais, o que reforça a maior complexidade e variabilidade estrutural dos dados reais. Novamente, para facilitar a visualização, os melhores resultados obtidos em cada *dataset* estão destacados em negrito.

Novamente, os métodos utilizados para a obtenção dos parâmetros responsáveis por esses resultados são apresentados na Seção 5.1, enquanto os valores obtidos encontram-se nas

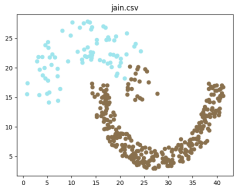
Figura 16 – *Dataset* Jain.



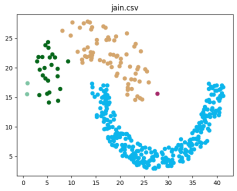
(a) *k-means*



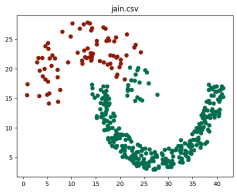
(b) *ward*



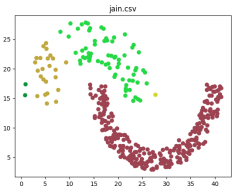
(c) *complete*



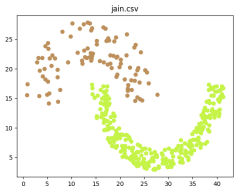
(d) *single*



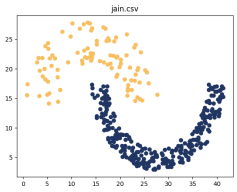
(e) *average*



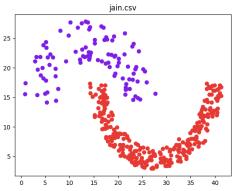
(f) *DBSCAN*



(g) CEAC



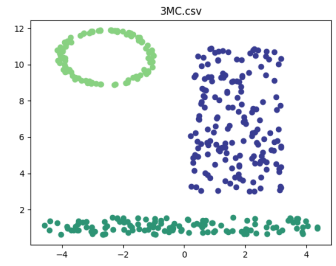
(h) CEAC++



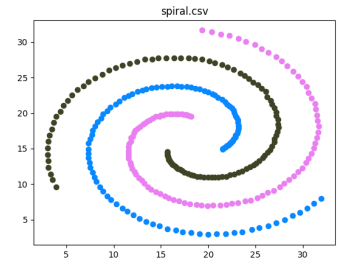
(i) Partição real

Fonte: Elaborado pelo autor.

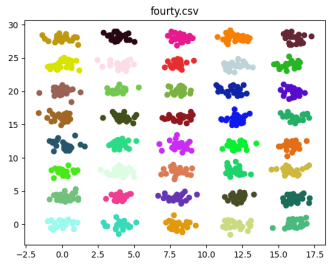
Figura 17 – *Datasets* artificiais em que o CEAC++ obteve desempenho perfeito.



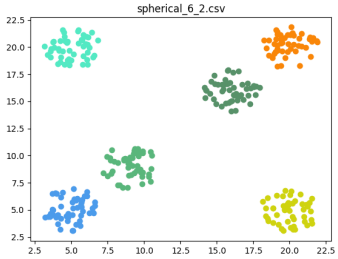
(a) 3MC



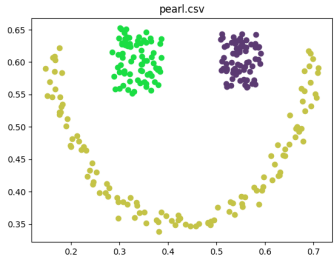
(b) Spiral



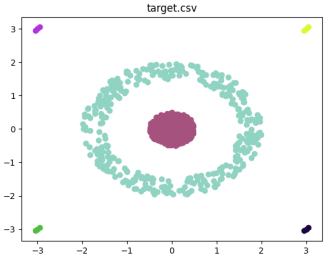
(c) Fourty



(d) Spherical_6_2



(e) Pearl



(f) Target

Fonte: Elaborado pelo autor.

Tabela 5 – Descrição dos *datasets* reais

	dimensões	tamanho	grupos
Arrhythmia	262	452	13
Banknote	4	1372	2
Biodeg	41	1055	2
Cns2	1379	42	5
Ionosphere	34	351	2
Libras	90	360	15
Mesothelioma	1626	181	2
Olivetti_Faces	4096	400	40
Sonar	60	208	2
Tea_Complete	5	151	3
Zoo	16	78	7
Zoo_Complete	16	101	7

Tabela 6 – Desempenho dos algoritmos com *datasets* reais utilizando a métrica *ARI*

		Hierárquico aglomerativo						
	k-means	*ward*	*complete*	*single*	*average*	*DBSCAN*	CEAC	CEAC++
Arrhythmia	0,093	0,113	0,182	0,307	0,306	**0,326**	0,250	0,308
Banknote	0,223	0,281	0,126	0,558	0,370	0,599	0,628	**0,712**
Biodeg	0,035	0,069	0,052	0,066	0,042	0,066	0,018	**0,075**
Cns2	0,389	**0,655**	0,442	0,273	0,283	0,273	0,528	0,540
Ionosphere	0,215	0,322	0,197	0,541	0,323	0,541	0,486	**0,546**
Libras	0,303	0,345	0,293	0,246	0,335	0,246	0,337	**0,349**
Mesothelioma	0,172	0,154	0,277	0,230	0,118	0,299	**0,363**	**0,363**
Olivetti_Faces	0,442	**0,608**	0,472	0,468	0,529	0,468	0,512	0,512
Sonar	0,028	0,034	0,051	0,073	0,033	0,073	**0,098**	**0,098**
Tea_Complete	0,023	0,026	0,030	0,021	0,027	0,021	0,034	**0,035**
Zoo	0,563	0,686	0,655	**0,697**	0,676	**0,697**	0,612	**0,697**
Zoo_Complete	0,584	0,735	0,721	0,761	**0,763**	0,761	0,652	0,761

Tabelas 7 e 8.

Tabela 7 – Parâmetros usados nos algoritmos CEAC, *k-means* e agrupamento hierárquico com *datasets* reais.

		Hierárquico aglomerativo				
	k-means	*ward*	*complete*	*single*	*average*	CEAC
Arrhythmia	7	11	16	116	56	10
Banknote	6	8	14	9	6	11
Biodeg	6	8	4	68	40	7
Cns2	8	7	13	30	27	7
Ionosphere	4	4	52	112	70	14
Libras	22	24	26	123	30	6
Mesothelioma	4	3	5	11	24	14
Olivetti_Faces	71	66	104	185	121	4
Sonar	24	10	10	102	35	6
Tea_Complete	13	12	9	54	12	7
Zoo	7	12	10	15	15	5
Zoo_Complete	6	10	10	15	14	12

Tabela 8 – Parâmetros usados nos algoritmos *DBSCAN* e CEAC++ com *datasets* reais.

	DBSCAN	CEAC++
Arrhythmia	'minPts': 189; 'ε': 193.09512414033776	'k': 26; 'r': 171.0905644313338
Banknote	'minPts': 30; 'ε': 2.3587654122856367	'k': 20; 'r': 1.8984742975955802
Biodeg	'minPts': 1; 'ε': 8.33284600432112	'k': 48; 'r': 9.473561567721097
Cns2	'minPts': 2; 'ε': 22758.094614312715	'k': 9; 'r': 47838.532812715515
Ionosphere	'minPts': 9; 'ε': 1.5578283763325391	'k': 80; 'r': 1.7122491681858394
Libras	'minPts': 1; 'ε': 0.7075059090205275	'k': 5; 'r': 1.2646476515911995
Mesothelioma	'minPts': 81; 'ε': 16108.112853437437	'k': 13; 'r': 23049.371815283404
Olivetti_Faces	'minPts': 1; 'ε': 6.020719381242425	'k': 3; 'r': 8.289604653328281
Sonar	'minPts': 1; 'ε': 0.7565560920350988	'k': 5; 'r': 1.3916558316619607
Tea_Complete	'minPts': 1; 'ε': 3.779551571175481	'k': 7; 'r': 7.641749115443325
Zoo	'minPts': 1; 'ε': 1.421518878688817	'k': 10; 'r': 1.4314410952331527
Zoo_Complete	'minPts': 2; 'ε': 1.6218672717115026	'k': 26; 'r': 1.5980541825288679

Além disso, observa-se uma variação expressiva dos valores de *ARI* entre os diferentes *datasets* possivelmente explicada pela heterogeneidade desses conjuntos, que diferem significativamente quanto à dimensionalidade, número de classes, distribuição dos dados e nível de sobreposição entre grupos. Essa diversidade torna o problema de agrupamento substancialmente mais desafiador e impacta de maneira distinta o desempenho de cada algoritmo.

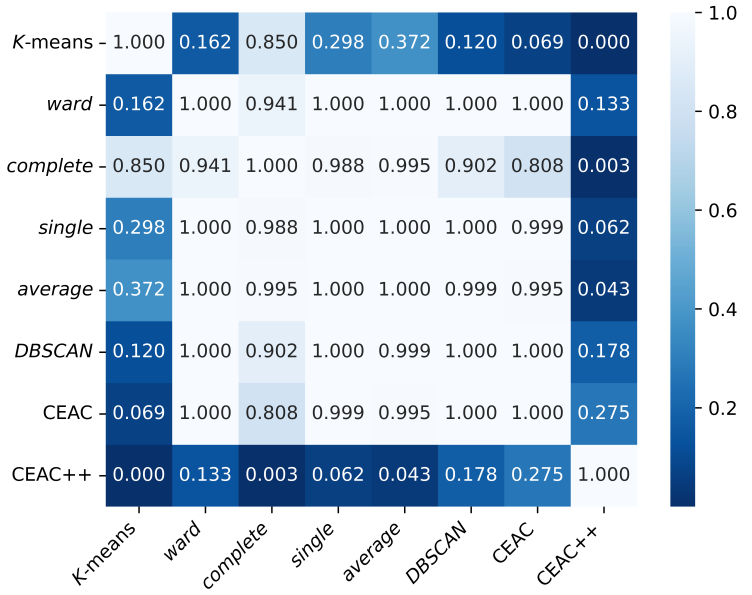
Para verificar a existência de diferenças estatisticamente significativas entre os métodos, foi aplicado novamente o teste de Friedman, que resultou em um valor-p de $6{,}008 \times 10^{-5}$. Como esse valor é inferior ao nível de significância adotado de $\alpha = 0{,}05$, rejeita-se a hipótese nula de equivalência de desempenho entre os algoritmos.

Diante desse resultado, foi realizado o teste *post hoc* de Nemenyi, cujos resultados são apresentados no mapa de calor da Figura 18. Observa-se que o CEAC++ apresenta diferença significativa em relação ao *k-means* e aos métodos de agrupamento hierárquico com *linkage complete* e *average*. Essa diferença pode ser explicada pelo desempenho superior do CEAC++ nesses *datasets*, uma vez que, em diversos casos, ele obteve os melhores resultados.

Para os demais pares de algoritmos, não foram observadas diferenças estatisticamente significativas, indicando desempenhos relativamente próximos nos *datasets* reais considerados.

Esses resultados evidenciam que os algoritmos propostos, especialmente o CEAC++, conseguem particionar bem os dados quando comparados aos algoritmos clássicos. No entanto, não é possível generalizar esse resultado, devido à baixa quantidade de *datasets* utilizados nos experimentos.

Figura 18 – Teste *Post-Hoc* de Nemenyi sobre os resultados dos *datasets* reais.



Fonte: Elaborado pelo autor.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho explorou o conceito de *core* estrito e o utilizou como base para propor um novo método de *clustering*. Como resultado, foram desenvolvidos dois algoritmos: o CEAC e o CEAC++.

Os experimentos realizados demonstraram que ambos os algoritmos apresentam bom desempenho quando comparados a métodos clássicos da literatura. Destaca-se, em particular, a performance do CEAC++ em *datasets* reais, na qual o algoritmo obteve resultados superiores aos de diversos algoritmos tradicionais.

Esses resultados indicam que o *core* estrito de um jogo de apreciação de amigos pode ser utilizado de forma eficaz para representar boas partições em conjuntos de dados.

Entre as possíveis direções para trabalhos futuros, destaca-se a investigação de tipos específicos de problemas ou domínios de aplicação nos quais os algoritmos propostos apresentem desempenho particularmente favorável. Neste trabalho, foram utilizados *datasets* provenientes de diferentes áreas, sem foco em um domínio específico, o que abre espaço para estudos mais direcionados.

Outro objetivo consiste em explorar diferentes formas de definir as relações de amizade entre os jogadores. Esse aspecto é especialmente relevante, pois novas definições podem levar a representações mais adequadas das preferências dos jogadores e, consequentemente, a resultados de *clustering* ainda melhores.

Por fim, pretende-se investigar métodos mais eficientes para a definição dos parâmetros dos algoritmos. Neste trabalho, os parâmetros foram determinados por meio de busca exaustiva (força bruta), abordagem que não é viável em aplicações reais. Dessa forma, torna-se importante desenvolver estratégias mais simples e eficientes para estimar esses valores, especialmente no caso do CEAC++, que possui um parâmetro inteiro e outro real.

REFERÊNCIAS

AKIBA, T.; SANO, S.; YANASE, T.; OHTA, T.; KOYAMA, M. Optuna: A next-generation hyperparameter optimization framework. In: **Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining**. [S.l.: s.n.], 2019. p. 2623–2631.

ASUNCION, A.; NEWMAN, D. *et al.* **UCI machine learning repository**. [S.l.]: Irvine, CA, USA, 2007.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, ACM New York, NY, USA, v. 18, n. 9, p. 509–517, 1975.

BOGOMOLNAIA, A.; JACKSON, M. O. The stability of hedonic coalition structures. **Games and Economic Behavior**, Elsevier, v. 38, n. 2, p. 201–230, 2002.

BRYANT, A.; CIOS, K. Rnn-dbscan: A density-based clustering algorithm using reverse nearest neighbor density estimates. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 30, n. 6, p. 1109–1121, 2017.

CHALKIADAKIS, G.; ELKIND, E.; WOOLDRIDGE, M. **Computational aspects of cooperative game theory**. [S.l.]: Morgan & Claypool Publishers, 2011.

COELHO, A. L.; SANDES, N. C. Data clustering via cooperative games: a novel approach and comparative study. **Information Sciences**, Elsevier, v. 545, p. 791–812, 2021.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. [S.l.]: MIT press, 2022.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **Journal of Machine learning research**, v. 7, n. Jan, p. 1–30, 2006.

DHAMAL, S.; BHAT, S.; ANOOP, K.; EMBAR, V. R. Pattern clustering using cooperative game theory. **arXiv preprint arXiv:1201.0461**, 2012.

DIMITROV, D.; BORM, P.; HENDRICKX, R.; SUNG, S. C. Simple priorities and core stability in hedonic games. **Social Choice and Welfare**, Springer, v. 26, n. 2, p. 421–433, 2006.

DRIVER, H. E.; KROEBER, A. L. Quantitative expression of cultural relationships. **(No Title)**, 1932.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. *et al.* A density-based algorithm for discovering clusters in large spatial databases with noise. In: **kdd**. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231.

EULER, L. Solutio problematis ad geometriam situs pertinentis. **Commentarii academiae scientiarum Petropolitanae**, p. 128–140, 1741.

GARG, V. K.; NARAHARI, Y.; MURTY, M. N. Novel biobjective clustering (bigc) based on cooperative game theory. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 25, n. 5, p. 1070–1082, 2012.

GOOGLE. **Advantages and Disadvantages of K-Means**. 2025. Acessado em: 30 set. 2025. Disponível em: <https://developers.google.com/machine-learning/clustering/kmeans/advantages-disadvantages>.

HAO, J.; HO, T. K. Machine learning made easy: a review of scikit-learn package in python programming language. **Journal of educational and behavioral statistics**, SAGE Publications Sage CA: Los Angeles, CA, v. 44, n. 3, p. 348–361, 2019.

HUBERT, L.; ARABIE, P. Comparing partitions. **Journal of classification**, Springer, v. 2, n. 1, p. 193–218, 1985.

MCQUEEN, J. B. Some methods of classification and analysis of multivariate observations. In: **Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.** [S.l.: s.n.], 1967. p. 281–297.

NEUMANN, J. von; MORGENSTERN, O. **Theory of Games and Economic Behavior**. Princeton, NJ: Princeton University Press, 1944.

PARMAR, M. **Clustering-Datasets**. 2019. <https://github.com/milaan9/Clustering-Datasets>. Acessado em: 13 mar. 2026.

PIECH, C. **K Means — Handout para CS221, Stanford**. 2013. Acessado em: 30 set. 2025. Disponível em: <https://stanford.edu/~cpiech/cs221/handouts/kmeans.html>.

SANDES, N. C.; COELHO, A. L. Clustering ensembles: A hedonic game theoretical approach. **Pattern Recognition**, Elsevier, v. 81, p. 95–111, 2018.

SCIENCE, D. D. of D. **The Limitations of DBSCAN Clustering Which Many Often Overlook**. 2023. Acessado em: 30 set. 2025. Disponível em: <https://blog.dailydoseofds.com/p/the-limitations-of-dbscan-clustering>.

SHARIR, M. A strong-connectivity algorithm and its applications in data flow analysis. **Computers & Mathematics with Applications**, Elsevier, v. 7, n. 1, p. 67–72, 1981.

SHEKHAR, M.; THOMAS, L.; KARLAPALEM, K. High dimensional clustering: a strongly connected component clustering solution (sccc). In: IEEE. **2018 IEEE International Conference on Data Mining Workshops (ICDMW)**. [S.l.], 2018. p. 1104–1111.

SOKAL, R. R.; MICHENER, C. D. *et al.* A statistical method for evaluating systematic relationships. University of Kansas Scientific Bulletin, 1958.

SULTANA, S. I. **How the Hierarchical Clustering Algorithm Works**. 2020. Acessado em: 30 set. 2025. Disponível em: <https://dataaspirant.com/hierarchical-clustering-algorithm/>.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introdução ao datamining: mineração de dados**. [S.l.]: Ciência Moderna, 2009.

TARJAN, R. Depth-first search and linear graph algorithms. **SIAM journal on computing**, SIAM, v. 1, n. 2, p. 146–160, 1972.

TRYON, R. **Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality**. Edwards brother, Incorporated, lithoprinters and publishers, 1939. Disponível em: <https://books.google.com.br/books?id=gsnrAAAAMAAJ>.

ANEXO

DECLARAÇÃO

Declaro para os devidos fins que este Trabalho de Conclusão de Curso (Monografia/Tese/Dissertação), escrito sob minha orientação, está em versão final, de acordo com as solicitações realizadas pela banca examinadora.

Informo também que procedi à revisão final do texto, constatando que atende às especificações das normas da ABNT para apresentação de trabalhos acadêmicos da UFCA, no que diz respeito ao conteúdo e à formatação.

Data e local da assinatura eletrônica

gov.br
Documento assinado digitalmente
NELSON CARVALHO SANDES
Data: 06/05/2026 09:59:46-0300
Verifique em https://validar.iti.gov.br