MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Colegiado do curso de Engenharia de Produção

Criação de um catálogo de grafos e soluções para problemas de dominação em grafos a partir de uma interface gráfica

André Fernandes do Prado Tessaro

João Monlevade, MG
2025

André Fernandes do Prado Tessaro

Criação de um catálogo de grafos e soluções para problemas de dominação em grafos a partir de uma interface gráfica

Trabalho de conclusão de curso apresentado ao curso de Engenharia de Produção do Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto, como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia de Produção.

Orientador: Prof. Alexandre Xavier Martins

Coorientador: Prof. Paganini Barcellos de Oliveira

João Monlevade, MG

2025

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

T338c Tessaro, André Fernandes do Prado.
Criação de um catálogo de grafos e soluções para problemas de dominação em grafos a partir de uma interface gráfica. [manuscrito] / André Fernandes do Prado Tessaro. - 2025.
71 f.: il.: color., color., tab..

Orientador: Prof. Dr. Alexandre Xavier Martins.
Coorientador: Prof. Dr. Paganini Barcellos de Oliveira.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Produção .

1. Algoritmos. 2. Grafos de ligação. 3. Modelos matemáticos. 4. Pesquisa operacional. 5. Representações dos grafos. 6. Teoria dos grafos. I. Martins, Alexandre Xavier. II. Oliveira, Paganini Barcellos de. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 519.17

Bibliotecário(a) Responsável: Flavia Reis - CRB6/2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO - ICEA

FOLHA DE APROVAÇÃO

André Fernandes do Prado Tessaro

Criação de um catálogo de grafos e soluções para problemas de dominação em grafos a partir de uma interface gráfica

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

Aprovada em 04 de agosto de 2025

Membros da banca

Dr. Alexandre Xavier Martins - Orientador - Universidade Federal de Ouro Preto
Dr. Paganini Barcellos de Oliveira - Universidade Federal de Ouro Preto
Msc. Braulio Frances Barcelos - Universidade Federal de Ouro Preto
Dr. Samuel Martins Drei - Universidade Federal de Ouro Preto

Alexandre Xavier Martins, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 06/08/2025

Documento assinado eletronicamente por **Alexandre Xavier Martins**, **CHEFE DO DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO**, em 06/08/2025, às 15:43, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0954948** e o código CRC **8F91A8A1**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0817 - www.ufop.br

Agradecimentos

A Deus, minha gratidão por ter me concedido força, sabedoria e saúde para concluir mais esta etapa da minha vida. Aos professores doutores Alexandre Martins, Paganini de Oliveira e Annegret Wagler, expresso meu sincero agradecimento pela orientação dedicada, incentivo constante e pelas valiosas contribuições que foram fundamentais para o desenvolvimento deste trabalho. Estendo meus agradecimentos aos demais docentes do curso, que, ao longo da graduação, compartilharam seus conhecimentos e experiências, enriquecendo minha formação acadêmica. À minha família, pelo amor incondicional, apoio nos momentos difíceis e por nunca deixarem de acreditar em mim. À minha namorada Elessara, por todo o carinho, paciência e apoio. Aos colegas e amigos que caminharam ao meu lado nessa jornada, em especial João Pedro, Maurício e Vitor, pela parceria e companheirismo. Ao Edgard, meu supervisor na ArcelorMittal, agradeço pela oportunidade de crescimento e aprendizado durante o estágio. Manifesto também minha gratidão à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), pelo apoio e financiamento à pesquisa realizada na França, a qual serviu de base para este trabalho. Por fim, a todos que, de alguma forma, contribuíram para a realização desta conquista, deixo registrado meu mais sincero muito obrigado.

Resumo

As propriedades de dominação em grafos têm sido amplamente estudadas em razão de sua relevância na modelagem e resolução de problemas complexos em diversas áreas, como redes de comunicação, logística e otimização de sistemas. Este trabalho tem como objetivo a criação de um catálogo sistematizado de grafos, hipergrafos e respectivas soluções, com o intuito de apoiar pesquisas voltadas para problemas de dominação. Para isso, utiliza-se uma interface gráfica previamente desenvolvida pelo autor durante o período de intercâmbio acadêmico no *Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes* (LIMOS), a qual integra funcionalidades que permitem a criação, manipulação e transformação de grafos, além da geração automática das listas de restrições que definem os problemas de cobertura associados. O estudo contempla grafos das classes *Path, Cycle, Clique, Pan e House*, sobre os quais são aplicadas diferentes operações e extraídas soluções computacionais organizadas em um repositório público, visando facilitar o acesso e a reutilização dos dados. A metodologia adotada possui caráter prático e exploratório, propondo-se a incorporar uma nova funcionalidade à ferramenta, bem como a realizar testes sistemáticos sobre as estruturas e operações suportadas pela interface. Todos os resultados obtidos são devidamente documentados, dessa forma, espera-se com isso contribuir para a disseminação do conhecimento e o avanço dos estudos no campo da teoria dos grafos.

Palavras-chaves: Grafos e hipergrafos, Problemas de dominação, Catálogo.

Abstract

Domination properties in graphs have been widely studied due to their relevance in modeling and solving complex problems in several areas, such as communication networks, logistics and systems optimization. This work aims to create a systematic catalog of graphs, hypergraphs and their respective solutions, with the aim of supporting research focused on domination problems. For this purpose, a graphical interface previously developed by the author during an academic exchange period at LIMOS is used, which integrates functionalities that allow the creation, manipulation and transformation of graphs, in addition to the automatic generation of lists of constraints that define the associated coverage problems. The study includes graphs of the Path, Cycle, Clique, Pan and House classes, on which different operations are applied and computational solutions are extracted and organized in a public repository, aiming to facilitate access and reuse of data. The adopted methodology has a practical and exploratory nature, proposing to incorporate a new functionality to the tool, as well as to perform systematic tests on the structures and operations supported by the interface. All results obtained are duly documented, thus, it is expected to contribute to the dissemination of knowledge and the advancement of studies in the field of graph theory.

Keywords: Graphs, Domination Problems, Catalog.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de abreviaturas e siglas

CPLEX *IBM ILOG CPLEX Optimization Studio*

BRAFITEC *Programa de Cooperação Franco-Brasileira para a Formação de Engenheiros*

DTD *differentiating total dominating set*

ID *identifying code set*

LD *locating-dominating set*

LIMOS *Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes*

LTD *location-total domination set*

OLD *open location domination set*

OSD *open separating dominating set*

PD *pair-locating dominating set*

PTD *pair-locating open dominating set*

Sumário

1 Introdução

As propriedades de dominação, separação e localização em grafos têm despertado amplo interesse na comunidade científica, em razão de sua capacidade de modelar e oferecer soluções a problemas complexos em distintos contextos aplicados. Segundo Haynes, Hedetniemi e Slater (2013), os conjuntos gerados a partir dessas propriedades exibem características e comportamentos particulares, que frequentemente impõem desafios computacionais expressivos na busca por soluções ótimas. Nesse cenário, Slater (1987) destaca que a combinação dessas propriedades amplia o potencial de aplicação prática, abrangendo desde a cobertura eficiente de redes e sistemas até a localização de elementos críticos em estruturas organizacionais e infraestruturas diversas.

Com o avanço dos estudos nesta temática, novas abordagens metodológicas têm sido propostas para o tratamento dos problemas de dominação e suas variantes. Nesse sentido, Argiroffo, Bianchi e Wagler (2016), bem como Argiroffo *et al.* (2018) e Argiroffo *et al.* (2022), introduziram reformulações fundamentadas em problemas de cobertura em hipergrafos, utilizando a modelagem poliédrica como estratégia para identificar conjuntos mínimos de solução. Essa perspectiva contribuiu de forma relevante para o aprimoramento das técnicas de modelagem e resolução desses problemas complexos, ampliando as possibilidades de análise e aplicação prática.

A fim de apoiar e operacionalizar tais modelagens, Boasquevisque (2023) desenvolveu ferramentas computacionais voltadas à criação de hipergrafos, eliminação de linhas redundantes das matrizes de incidência e conversão dessas estruturas em listas de restrições adequadas à formulação dos problemas de cobertura correspondentes. Complementando esses avanços, De Almeida, Mendes e Tessaro (2024) implementaram uma ferramenta adicional, capaz de automatizar tanto a criação quanto a aplicação de operações sobre grafos, facilitando a geração de instâncias para análise.

Com o objetivo de integrar e centralizar o uso dessas distintas ferramentas em um único ambiente computacional, Tessaro (2024) propôs o desenvolvimento de uma interface gráfica unificada. Essa interface não apenas incorporou as funcionalidades pré-existentes, mas também introduziu um novo recurso, permitindo a obtenção automatizada de inequações próprias para integração com o solver *IBM ILOG CPLEX Optimization Studio* (*CPLEX*), otimizando o processamento e ampliando as possibilidades de exploração computacional dos problemas.

Dentre os esforços voltados à compreensão do custo computacional associado aos problemas de dominação, destaca-se o trabalho de Telle (1994), que propôs uma estrutura unificadora baseada em pares ordenados para classificar problemas de dominação em grafos. O autor apresentou uma taxonomia que distingue os problemas solucionáveis em tempo polinomial daqueles que pertencem à classe dos *NP*-completos, evidenciando o impacto da estrutura do grafo e das restrições locais sobre a dificuldade de resolução. Tal abordagem fornece subsídios teóricos relevantes para o desenvolvimento de algoritmos e estratégias de resolução mais eficazes.

Embora existam diversos estudos que abordam problemas específicos ou variantes isoladas, há uma lacuna no que diz respeito à integração de múltiplas operações, classes de grafos e problemas em um único ambiente de análise. Diante do exposto, identificou-se a oportunidade de promover avanços e melhorias na interface gráfica previamente desenvolvida no trabalho de Tessaro (2024), o que deu origem a proposta deste trabalho.

Nesse sentido, justifica-se o presente trabalho pela proposta de ampliação das funcionalidades da ferramenta de Tessaro (2024), por meio da implementação de um novo recurso que possibilite a geração automática de imagens ilustrativas dos grafos criados. Além dessa melhoria, propõe-se a execução sistemática da interface com o objetivo de construir um catálogo abrangente, contendo diversas instâncias de grafos, problemas formulados e respectivas soluções computacionais, contribuindo para a consolidação de um acervo útil à pesquisa e ao ensino na área de teoria dos grafos.

1.1 Objetivo geral

O objetivo geral deste trabalho consiste no desenvolvimento de um catálogo sistemático de grafos, hipergrafos e respectivas soluções, com foco na exploração das propriedades de dominação. Para isso, utiliza-se uma interface gráfica como instrumento central na criação, manipulação e transformação de diferentes estruturas de grafos. O catálogo resultante visa constituir uma ferramenta prática e acessível, capaz de auxiliar pesquisadores, estudantes e profissionais no estudo aprofundado de problemas de dominação, além de fomentar a aplicação desses conceitos em contextos tanto acadêmicos quanto operacionais.

1.1.1 Objetivos específicos

Para o cumprimento do objetivo geral, torna-se necessário atender a uma série de objetivos específicos. Entre eles, destaca-se a intenção de atender a uma das sugestões de trabalhos futuros indicadas por Tessaro (2024), voltada à ampliação das funcionalidades da interface gráfica no contexto de estudos em teoria dos grafos. Nesse sentido, este trabalho propõe a incorporação de uma nova funcionalidade à ferramenta, voltada à geração automática de representações visuais dos grafos construídos.

Adicionalmente, objetiva-se realizar uma análise computacional de uma das funcionalidades centrais da interface, por meio da avaliação de seu desempenho em termos de tempo de execução e uso de armazenamento. Essa análise visa fornecer subsídios técnicos relevantes para o aprimoramento da ferramenta, bem como orientar futuras aplicações e desenvolvimentos no âmbito da teoria dos grafos e da modelagem computacional.

Dessa forma, os objetivos específicos deste trabalho são:

- Incorporar à interface a funcionalidade de geração automática de figuras representativas dos grafos;
- Testar e documentar os grafos das classes *Path*, *Cycle*, *Clique*, *Pan* e *House*, gerados por meio da interface;
- Aplicar, a cada grafo, as operações estruturais disponíveis na ferramenta;
- Criar os respectivos hipergrafos e gerar as inequações de entrada no formato compatível com o *solver CPLEX*;
- Executar os arquivos no *solver CPLEX* para obtenção das soluções mínimas dos problemas formulados;
- Organizar os resultados obtidos e disponibilizá-los em repositório público;
- Analisar os custos computacionais envolvidos, com base no tempo de execução e no tamanho dos arquivos gerados.

1.2 Contribuições

Diante desse contexto, esta pesquisa tem como propósito contribuir para o estudo dos problemas de dominação em grafos, por meio da disponibilização de um repositório estruturado e abrangente contendo grafos das classes Path, Cycle, Clique, Pan e House. O catálogo inclui, de forma organizada, as combinações entre os diferentes problemas abordados e suas respectivas soluções computacionais, possibilitando consultas sistemáticas e comparações entre os resultados.

Com o intuito de expandir o alcance e promover a acessibilidade, o material será disponibilizado em uma pasta pública na plataforma *Google Drive*. Isso permitirá que estudantes, pesquisadores e profissionais interessados possam utilizá-lo como ferramenta de apoio em atividades acadêmicas e investigativas relacionadas à teoria dos grafos.

Além disso, a melhoria implementada na interface, relacionada à geração de figuras, não se limita a facilitar a visualização e a compreensão das estruturas geradas. Ela também contribui de forma significativa para o aprimoramento geral da ferramenta. Assim, amplia-se seu potencial enquanto recurso de apoio didático no contexto dos estudos em teoria dos grafos.

1.3 Organização do Trabalho

Este trabalho está estruturado em cinco capítulos. O capítulo 1 trata da introdução e objetivos do trabalho. O capítulo 2 oferece uma fundamentação teórica relevante e uma breve revisão da literatura sobre o problema em estudo. O capítulo 3 descreve a metodologia empregada e o capítulo 4 apresenta uma análise do custo computacional em segundos para os problemas. Por fim, o capítulo 5 discute a conclusão e propõe direções futuras para o desenvolvimento e aprimoramento do trabalho.

2 Fundamentação teórica

2.1 Definição de grafos

De forma resumida, pode-se dizer que um grafo é composto por um par ordenado $G = (V, E)$ de conjuntos de vértices e arestas (Morales, 2019). O conjunto de elementos denominados vértices (V) são frequentemente representados graficamente como pontos ou nós e são acompanhados por um grupo de conexões entre eles, chamadas arestas (E), que comumente são ilustradas como segmentos de retas (Tóth, 2000). Outra forma de representar um grafo é pela lista de adjacência ou vizinhos (Cormen *et al.*, 2009).

Segundo Brandstädt, Le e Spinrad (1999) a vizinhança aberta de um vértice v, denotada por $N(v)$, é definida como o conjunto de todos os vértices adjacentes a v, ou seja, aqueles que estão diretamente conectados a v por uma aresta. Por outro lado, segundo os autores, a vizinhança fechada de v, representada por $N[v]$, inclui todos os vértices da vizinhança aberta mais o próprio vértice v. Formalmente, $N[v] = N(v) \cup \{v\}$.

2.2 Dominação, localização e separação em grafos

A dominação em grafos é um conceito importante na teoria dos grafos, amplamente estudado devido às suas diversas aplicações reais como conjunto de representantes, rotas de ônibus e redes de comunicação (Silva, 2010). Balakrishnan e Ranganathan (2012) definem dominação da seguinte forma: dado um grafo $G = (V, E)$, um conjunto $S \subseteq V$ é um conjunto dominante se todo vértice $v \in V$ que não pertence a S é adjacente a pelo menos um vértice em S. Segundo os autores, isso garante que todos os vértices estejam no conjunto dominante ou tenham um vizinho nele, o que possibilita a cobertura eficiente de redes e sistemas distribuídos (Xu; Li; Song, 2013).

A localização na teoria dos grafos envolve a identificação de vértices com base em seus relacionamentos com um subconjunto específico. De acordo com Foucaud e Henning (2016) um conjunto de localização dominante S é um subconjunto de vértices tal que cada vértice não presente em S é determinado exclusivamente pela interseção de sua vizinhança com S. Isso significa que para quaisquer dois vértices distintos u e v não presentes em S, os conjuntos $N(u) \cap S$ e $N(v) \cap S$ são diferentes, onde $N(u)$ denota a vizinhança aberta de u. Essa propriedade tem aplicações em redes de sensores, onde um conjunto reduzido de nós pode ser utilizado para identificar a posição de eventos na rede.

As noções de separação aberta e fechada em grafos é encontrada em Chakraborty e Wagler (2025). O conceito é similar à noção de localização, entretanto na separação, todos os vértices são identificáveis, incluindo aqueles presentes no conjunto C. Segundo os autores um conjunto C é considerado um conjunto separante aberto se a interseção da vizinhança aberta $N(v) \cap C$ for única para cada vértice $v \in V$, no caso da separação aberta. Ou seja, cada vértice do grafo é distinguido pela interseção de seus vértices adjacentes com o conjunto C. Da mesma forma, a separação fechada ocorre se a interseção da vizinhança fechada $N[v] \cap C$ for única para cada vértice $v \in V$. Esse conceito é relevante para problemas de identificação em redes de comunicação e sistemas de monitoramento.

Nesse contexto, trabalhos como o de Slater (1987) apresentaram combinações dessas propriedades. Nos trabalhos de Argiroffo *et al.* (2018), Argiroffo *et al.* (2022) e Chakraborty e Wagler (2025), há um aprofundamento nessa temática abordando um ponto de vista poliédrico. Dando sequência aos estudos citados, Boasquevisque (2023) apresenta oito X-problemas resultantes dessas combinações, onde, dado um grafo $G = (V, E)$, X-*set* é um conjunto, na qual X-*set* $\subseteq V$. conforme mostrado no Quadro 1.

Quadro 1 – Propriedades de cada problema

Nome do Problema	**Propriedade**
identifying code set (ID)-*set*	Dominação fechada Separação fechada
pair-locating dominating set (PD)-*set*	Dominação fechada Localização de pares
locating-dominating set (LD)-*set*	Dominação fechada Localização
open separating dominating set (OSD)-*set*	Dominação fechada Separação aberta
differentiating total dominating set (DTD)-*set*	Dominação aberta Separação fechada
pair-locating open dominating set (PTD)-*set*	Dominação aberta Localização de pares
location-total domination set (LTD)-*set*	Dominação aberta Localização
open location domination set (OLD)-*set*	Dominação aberta Separação aberta

Fonte: elaborado pelo autor.

Os problemas em questão possuem uma ampla gama de aplicações relevantes. Como exemplo, a detecção de falhas em redes multiprocessadoras (Karpovsky; Chakrabarty; Levitin, 1998), a localização de ameaças ou intrusos em ambientes monitorados por redes de sensores (Ungrangsi; Trachtenberg; Starobinski, 2004), a investigação da definibilidade lógica de grafos (Pikhurko; Veith; Verbitsky, 2006) e a rotulagem canônica voltada ao problema do isomorfismo de grafos (Babai, 1980).

Segundo Chakraborty e Wagler (2025), a instalação de dispositivos de vigilância em edifícios, com o objetivo de detectar e localizar a presença de intrusos, incêndios ou atos de sabotagem, proposta por Ray *et al.* (2003), pode ser modelada por meio de problemas de dominação e localização em grafos. Nesse contexto, o edifício é representado por um grafo, onde vértices representam os cômodos e as arestas indicam conexões entre eles.

Chakraborty e Wagler (2025) afirmam que as variações nos tipos de sensores utilizados impactam diretamente o modelo adotado. Sensores capazes de identificar eventos apenas no próprio cômodo remetem à dominação fechada, enquanto aqueles que detectam em cômodos adjacentes estão associados à dominação aberta. Assim, conjuntos dominantes são utilizados para assegurar a detecção da presença do intruso, enquanto propriedades de localização de vértices ou pares de vértices, são aplicadas para identificar com precisão a posição do evento no grafo.

2.3 Trabalhos relacionados

Trabalhos como os de Jean e Seo (2023) e Jean e Seo (2024) investigam, de forma mais aprofundada e específica, as particularidades de uma das variantes dos problemas de dominação abordados neste trabalho. Em ambos os estudos, os autores analisam variações do problema OLD, explorando cenários em que há capacidade de correção de falhas ou em que a presença de redundância é uma característica desejável, especialmente em aplicações sobre redes.

A principal diferença entre essas abordagens e o presente trabalho reside no foco adotado. Enquanto os trabalhos de Jean e Seo (2023) e Jean e Seo (2024) concentram-se, exclusivamente, em problemas de localização dominante, restringindo-se a um escopo mais delimitado, o presente trabalho tem como objetivo oferecer uma visão mais abrangente, ao sistematizar diferentes combinações de problemas de dominação aplicados a diversas classes de grafos, com ênfase na construção e catalogação das estruturas.

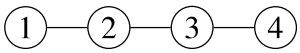
Um aspecto relevante deste trabalho é a aplicação de operações estruturais sobre grafos. Nesse sentido, há trabalhos relacionados que investigam transformações estruturais semelhantes. É o caso do estudo de Jr, Malacas e Tarepe (2014), que examina o número de localização-dominação em produtos do tipo *corona*. O estudo expande a estrutura do grafo original de forma controlada, permitindo a análise de como essa expansão afeta propriedades combinatórias como dominância e localização. Em contrapartida, mais uma vez, o presente trabalho se diferencia por adotar uma abordagem mais abrangente, ao considerar um conjunto mais amplo de operações estruturais aplicadas a diferentes classes de grafos.

2.4 Tipos de grafos abordados no estudo

Este trabalho contempla a análise de cinco tipos específicos de grafos, os quais são conceituados e exemplificados ainda nesta seção. A definição precisa dessas estruturas é fundamental, uma vez que cada uma delas será gerada, manipulada e submetida a testes ao longo do desenvolvimento da pesquisa, servindo como base para a aplicação dos problemas de dominação e das operações implementadas na interface.

Um grafo de n-*path*, denotado P_n, é um tipo de grafo que representa uma sequência, composta de vértices (ou nós) e arestas (Silva, 2010). Escalante, Montejano e Rojano (1974) expõe que em um *path* com n vértices, cada vértice v_i está conectado a v_{i-1} e v_{i+1}, exceto as extremidades. No caso, o primeiro nó é conectado apenas ao segundo nó, e o último nó está vinculado apenas ao nó anterior. A Figura 1 mostra um exemplo de um grafo do tipo *path* com 4 vértices representado por seus vizinhos.

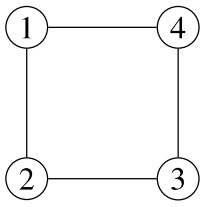
Figura 1 – Exemplo de um 4-*path*



Fonte: elaborado pelo autor.

Um n-*cycle*, denotado C_n e com $n \geq 3$, é um tipo de grafo como um *path*, porém com uma aresta extra que conecta o último vértice do grafo ao primeiro, criando assim uma estrutura cíclica (Santos, 2016). Assim, em um *cycle*, o último nó está ligado ao nó anterior e ao primeiro nó. Um exemplo de grafo cíclico com 4 vértices pode ser observado na Figura 2.

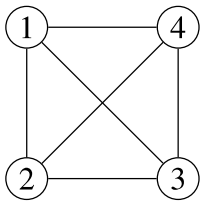
Figura 2 – Exemplo de um 4-*cycle*



Fonte: elaborado pelo autor.

Um n-*clique*, denotado por K_n e com $n \geq 3$ (Rocha, 2020), é um grafo que consiste em n nós, onde cada nó está conectado a todos os outros nós no grafo (Lozada, 1996). Isso significa que cada nó é vizinho de todos os outros nós e cada par de nós distintos em um n-*clique* é conectado por uma aresta (Ferreira, 2020). Para facilitar a compreensão, a Figura 3 mostra um exemplo de um *clique* com 4 vértices.

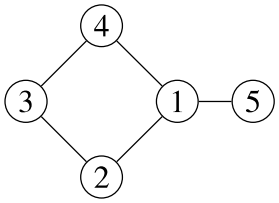
Figura 3 – Exemplo de um 4-*clique*



Fonte: elaborado pelo autor.

Um n-*pan* é um tipo especial de grafo, muito semelhante ao n-*cycle*. Entretanto, no caso do *pan* trata-se de um *cycle* somado à um nó adicional (Weisstein, 2025b) ou seja, adjacente à apenas um vértice do grafo (Rocha, 2020), o que visualmente assemelha-se ao formato de uma panela. A Figura 4 mostra um exemplo de *pan* com 4 vértices.

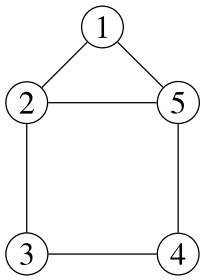
Figura 4 – Exemplo de um 4-*pan*



Fonte: elaborado pelo autor.

O último tipo de grafo que será trabalhado também é muito semelhante ao *cycle*. Entretanto, em uma *house* há uma conexão entre o segundo nó e o último nó (Brandstädt; Le; Spinrad, 1999). Em um grafo com 5 vértices, como na Figura 5, tem-se a ilustração esquemática desse grafo, com visual semelhante a uma casa com telhado (Weisstein, 2025a).

Figura 5 – Exemplo de um 4-*house*

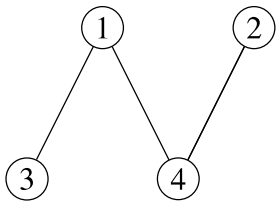


Fonte: elaborado pelo autor.

2.5 Tipos de operações abordadas no trabalho

Dentre as possíveis operações, a primeira a ser destacada é o *complement*. O *complement* de um grafo consiste em uma nova estrutura construída a partir de um grafo original, na qual são adicionadas arestas entre todos os pares de vértices que não estão conectados no grafo inicial, e, simultaneamente, são removidas as arestas existentes entre vértices adjacentes no grafo original (Sampathkumar; Pushpalatha, 1998). Em outras palavras, dois vértices que não possuem ligação direta no grafo base tornam-se adjacentes no *complement*, enquanto conexões existentes são suprimidas nessa nova configuração. A Figura 6 exemplifica essa operação aplicada ao grafo *4-path*, evidenciando a transformação estrutural resultante.

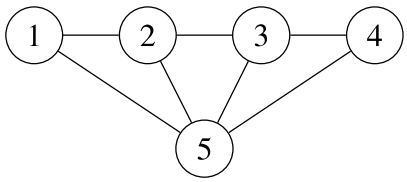
Figura 6 – Exemplo de um *complement-4-path*



Fonte: elaborado pelo autor.

A operação denominada *universal node* consiste na adição de um novo vértice ao grafo original, estabelecendo conexões diretas entre esse novo vértice e todos os demais vértices já existentes na estrutura (Bonato; Kemkes; Prałat, 2012). Dessa forma, o vértice adicionado assume o papel de um nó universal, isto é, um vértice adjacente a todos os outros do grafo. A Figura 7 ilustra essa operação aplicada ao grafo *4-path*, evidenciando a inclusão do vértice adicional e sua ligação com os demais elementos da estrutura.

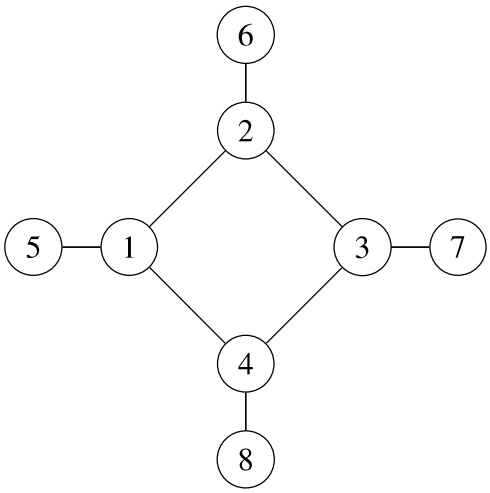
Figura 7 – Exemplo de um *universal-4-path*



Fonte: elaborado pelo autor.

Outra operação disponível na interface é a 1-*Corona*, que segundo (Barik; Pati; Sarma, 2007) consiste na adição de um novo vértice para cada vértice existente no grafo original, de modo que cada novo vértice seja conectado exclusivamente ao vértice correspondente. A Figura 8 ilustra essa operação aplicada à instância *4-path*, evidenciando a adição dos novos vértices periféricos.

Figura 8 – Exemplo de um *1-Corona-4-cycle*



Fonte: elaborado pelo autor.

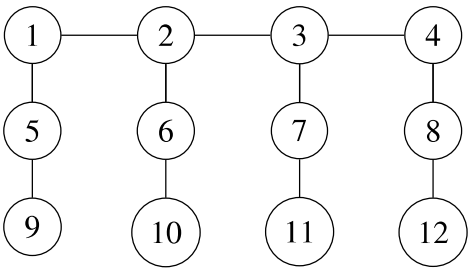
Formalmente, para um grafo $G = (V, E)$, a operação 1-*Corona* resulta em um novo grafo definido como $1C(G) = (V \cup U, E')$, em que $V = \{v_1, \ldots, v_n\}$ representa o conjunto original de vértices, $U = \{u_1, \ldots, u_n\}$ corresponde ao conjunto de vértices adicionados, e $E' = E \cup \{v_i u_i : 1 \leq i \leq n\}$ define o novo conjunto de arestas, composto pelas arestas originais e pelas conexões entre cada vértice v_i e seu respectivo vértice auxiliar u_i. Assim, a vizinhança dos vértices de $1C(G)$ são dados pelas Equações (2.1) e (2.2).

$$N(v_i) = N_G(v_i) \cup \{u_i\} \tag{2.1}$$

$$N(u_i) = \{v_i\} \tag{2.2}$$

A partir da operação 1-*Corona*, a operação 2-*Corona* basicamente consiste em adicionar, para cada vértice introduzido na operação anterior, um novo vértice adicional conectado exclusivamente a ele (Chakraborty *et al.*, 2024), como demonstra a Figura 9.

Figura 9 – Exemplo de um *2-Corona-4-path*



Fonte: elaborado pelo autor.

Formalmente, considerando um grafo $G = (V \cup E)$ a operação 2-*Corona* gera um novo grafo definido como $2C(G) = (V \cup U \cup W, E')$ onde os conjuntos V, U e W possuem a mesma cardinalidade, isto é, $|V| = |U| = |W|$. O novo conjunto de arestas E' é composto pelas arestas originais do grafo G, pelas conexões entre cada vértice $v_i \in V$ e seu respectivo vértice $u_i \in U$,

e pelas conexões entre cada vértice $u_i \in U$ e seu correspondente $w_i \in W$. Assim, a estrutura resultante apresenta três níveis de vértices conectados sequencialmente. Dessa forma, os vizinhos dos vértices de $2C(G)$ são dados pelas Equações (2.3), (2.4) e (2.5).

$$N(v_i) = N_G(v_i) \cup \{u_i\} \quad \forall v_i \in V \tag{2.3}$$

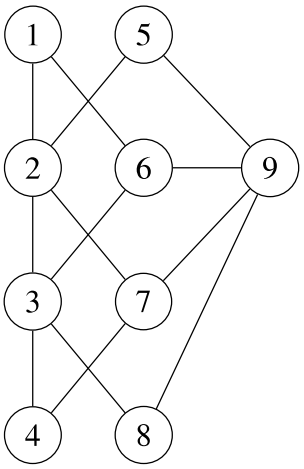
$$N(u_i) = \{v_i, w_i\} \quad \forall u_i \in U \tag{2.4}$$

$$N(w_i) = \{u_i\} \quad \forall w_i \in W \tag{2.5}$$

Finalmente, a última operação disponível à ser aplicada é a operação *mycielski*, ilustrada na Figura 10. Na teoria dos grafos, a operação *mycielski* consiste em pegar um grafo G e construir um grafo G' da seguinte forma (Hameed, 2024):

1. Para cada vértice v de G, cria-se um novo vértice correspondente v'.
2. Para cada aresta (u, v) de G, cria-se duas novas arestas (u, v') e (v, u') em G'.
3. Adiciona-se um vértice extra w em G' e conecta-o aos novos vértices de v'.

Figura 10 – Exemplo de um *mycielski-4-path*



Fonte: elaborado pelo autor.

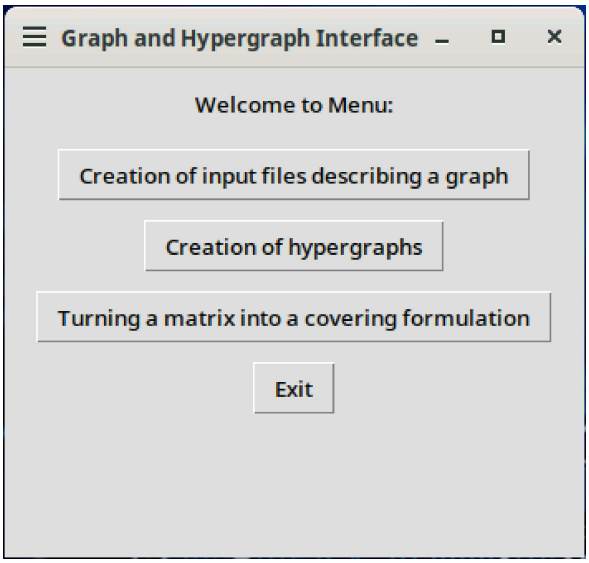
2.6 A Interface Gráfica

A motivação para a criação da interface surgiu da necessidade de integrar, em um único ambiente, as funcionalidades desenvolvidas por Boasquevisque (2023) e De Almeida, Mendes e Tessaro (2024), conforme sugestão da orientadora Annegret Wagler. Com esse objetivo, a ferramenta foi desenvolvida pelo próprio autor durante o período de estágio realizado no LIMOS, enquanto bolsista do *Programa de Cooperação Franco-Brasileira para a Formação de Engenheiros* (BRAFITEC).

Desse modo, a interface desenvolvida em Tessaro (2024) foi implementada em linguagem de programação *Python*, escolhida em virtude de sua ampla adoção e versatilidade no desenvolvimento de aplicações voltadas à manipulação e análise de grafos. Adicionalmente, as demais ferramentas integradas à interface também foram desenvolvidas em *Python*, o que favoreceu a compatibilidade entre os módulos e simplificou o processo de integração das funcionalidades.

Ao ser executado o programa por meio do terminal de comando da máquina, a aplicação inicializa automaticamente, abrindo a janela correspondente ao menu principal. Nesta tela, o usuário se depara com quatro opções de operação, organizadas sob a forma de botões funcionais, cuja disposição e estrutura estão ilustradas na Figura 11.

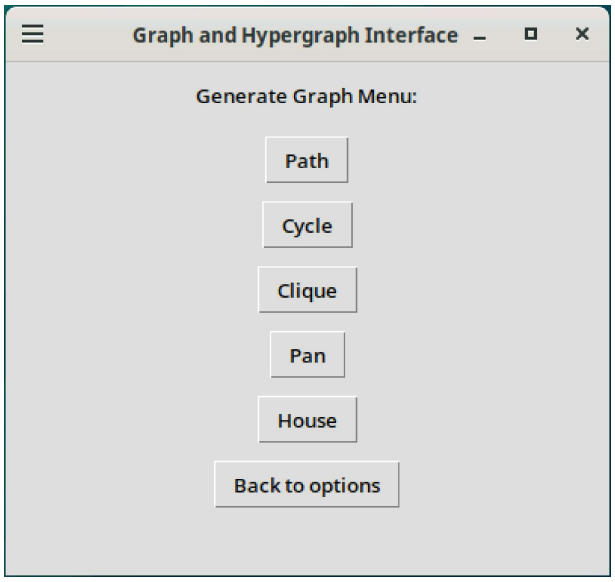
Figura 11 – Menu principal



Fonte: elaborado pelo autor.

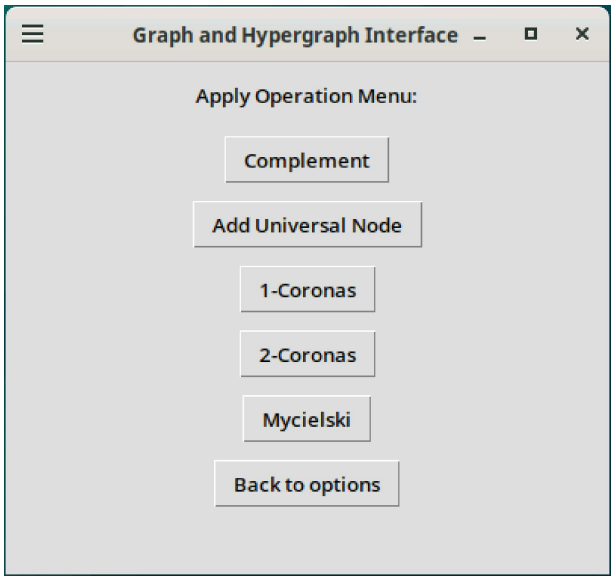
O primeiro botão disponibilizado no menu principal da interface tem como finalidade a criação de arquivos. Ao selecionar essa opção, o usuário é direcionado a um novo menu, no qual são apresentadas duas possibilidades distintas para a geração de grafos. A primeira alternativa, representada na Figura 12, permite a construção de um grafo pertencente a um dos cinco tipos previamente descritos na Seção 2.3, a partir da definição do número de vértices desejado. A segunda alternativa, por sua vez, possibilita a aplicação de uma das cinco operações sobre grafos, conforme detalhado na Seção 2.4 e ilustrado na Figura 13.

Figura 12 – Menu para criar grafos



Fonte: elaborado pelo autor.

Figura 13 – Menu para aplicar operações



Fonte: elaborado pelo autor.

Cabe destacar que, originalmente, os grafos gerados pela interface eram representados unicamente por meio de listas de vizinhos, como exemplificado na Figura 14, o que restringia a visualização estrutural das instâncias construídas. Com o avanço do presente trabalho, foi incorporada uma nova funcionalidade à interface, permitindo a geração automática de representações ilustrativas dos grafos. Essa funcionalidade amplia significativamente a compreensão visual das estruturas geradas, facilitando tanto a análise topológica quanto a interpretação dos resultados obtidos.

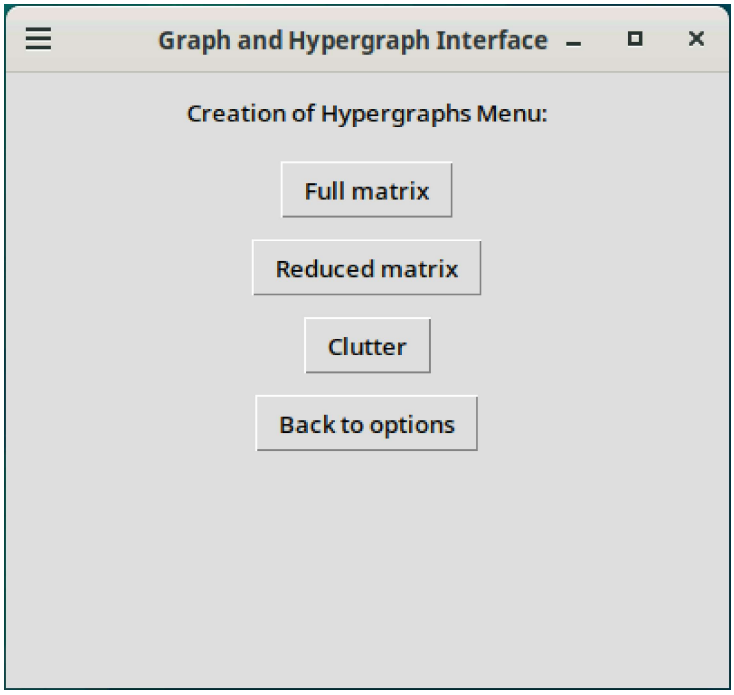
Figura 14 – Exemplo de arquivo 4-*path*.gra

```
DIM = 4

NEIGHBORHOOD_LISTS
N(1) = 2
N(2) = 1, 3
N(3) = 2, 4
N(4) = 3

END
```

Fonte: elaborado pelo autor.

O segundo botão disponibilizado no menu principal da interface denomina-se "*Creation of hypergraphs*". Ao selecionar essa opção, o usuário tem acesso à criação de diferentes estruturas matriciais, podendo gerar uma matriz completa "*Full matrix*", uma matriz reduzida "*Reduced matrix*"ou ainda um *"Clutter*", conforme ilustrado na Figura 15. As matrizes geradas correspondem, em essência, às representações formais dos oito problemas abordados e sistematizados no Quadro 1, servindo como base estrutural para a formulação e resolução das respectivas instâncias computacionais.

Figura 15 – Menu para gerar matrizes



Fonte: elaborado pelo autor.

Conforme exposto por Boasquevisque (2023), a redução da quantidade de linhas nas matrizes tem como principal objetivo facilitar a preparação e a aplicação do método poliédrico, além de contribuir diretamente para a diminuição do custo computacional envolvido na execução dos algoritmos. Após a criação da matriz, seja ela completa ou reduzida, é necessário aplicar a operação denominada *"Clutter"*, que consiste na eliminação automática de todas as linhas redundantes da matriz, garantindo assim uma estrutura mais enxuta e adequada para a formulação do problema de cobertura correspondente.

Por fim, o último botão presente no menu principal da interface é intitulado "*Turning a matrix into a covering formulation*"e tem como finalidade a geração automática de inequações matemáticas. Essa funcionalidade permite converter as matrizes previamente criadas em formulações algébricas compatíveis com o ambiente de resolução do *solver CPLEX*. A partir das inequações geradas, o usuário pode submeter a instância diretamente ao *solver*, viabilizando a obtenção da solução ótima para o problema de cobertura correspondente de forma automatizada e eficiente.

3 Metodologia

3.1 Tipo e abordagem da pesquisa

Este trabalho caracteriza-se como uma pesquisa aplicada, de natureza exploratória e descritiva, tendo como foco a geração, análise e catalogação de grafos e respectivas soluções para problemas de dominação, por meio de uma interface gráfica previamente desenvolvida pelo autor em pesquisa anterior.

Segundo Marconi e Lakatos (2003), a pesquisa aplicada visa gerar conhecimento voltado à solução de problemas específicos, com aplicação prática direta. No presente estudo, busca-se consolidar um repositório organizado de soluções computacionais para diferentes classes de grafos, contribuindo com o avanço de estudos na área de teoria dos grafos.

A pesquisa assume também um caráter exploratório, ao examinar de forma detalhada as funcionalidades da interface e a aplicabilidade de suas operações sobre os grafos, bem como descritivo, ao sistematizar e documentar as soluções geradas (Gil, 2002). Essa combinação metodológica permite não apenas aprofundar a compreensão dos problemas abordados, mas também organizar os resultados de maneira acessível e didática.

3.2 Procedimentos metodológicos

3.2.1 Levantamento teórico

A primeira etapa do trabalho consistiu em um levantamento bibliográfico, com o objetivo de embasar teoricamente os conceitos de grafos e os problemas de dominação estudados. Para tal, foram consultados artigos científicos, livros e materiais acadêmicos relevantes à temática, com ênfase em publicações recentes, buscando compreender as propriedades estruturais dos grafos, as definições formais dos problemas de dominação, bem como suas aplicações práticas, tal como descrito na Capítulo 2.

Este referencial teórico fundamentou as definições dos tipos de grafos analisados (*path, cycle, clique, pan e house*), das operações (*complement, universal node, 1-coronas, 2-coronas e mycielski* e das combinações entre problemas de dominação, localização e separação.

3.3 Exploração da interface gráfica

Na sequência, foi conduzida uma análise exploratória detalhada da interface gráfica de Tessaro (2024), com o objetivo de mapear suas funcionalidades, identificar eventuais limitações e reconhecer seu potencial de expansão. Essa etapa foi essencial para validar a aplicabilidade da ferramenta na criação e manipulação dos grafos de interesse, assegurando que as operações implementadas fossem compatíveis com os objetivos do presente estudo.

Paralelamente, avaliou-se a viabilidade da incorporação de uma nova funcionalidade voltada à geração automática de imagens no formato .png, representando visualmente os grafos construídos. A adaptação proposta visou aprimorar a visualização das estruturas, permitindo um registro gráfico padronizado e facilitando a análise estrutural de cada instância gerada ao longo do trabalho.

3.4 Catalogação de soluções

A etapa seguinte da pesquisa envolveu a sistematização dos grafos e respectivas soluções encontradas, organizando-os em um catálogo digital. Esse catálogo foi estruturado de forma a apresentar o tipo de grafo construído, sua imagem representativa e as soluções geradas para cada problema de dominação aplicado.

Para garantir ampla acessibilidade, o catálogo foi disponibilizado em uma pasta pública no *Google Drive*, servindo como fonte de consulta para pesquisadores, estudantes e profissionais interessados na área de dominação em grafos.

3.5 Mensuração do custo computacional

Por fim, como última etapa do trabalho, foi realizado o levantamento do custo computacional associado a uma das funcionalidades da interface gráfica, considerando dois parâmetros principais: o tempo de processamento, medido em segundos, e o tamanho do arquivo gerado, expresso em *quilobytes* (kB). Essa análise objetiva comparar o desempenho algorítmico entre as diferentes categorias de problemas abordados, considerando, inclusive, as variações de custo computacional observadas entre problemas que permitem ou não o descarte de diferenças simétricas redundantes.

O registro dos tempos de execução obtidos para os problemas aplicados aos diferentes tipos de grafos representa um insumo fundamental para investigações futuras. A partir desse registro é possível realizar análises de complexidade computacional e desempenho dos algoritmos envolvidos, auxiliando no desenvolvimento de soluções mais eficientes no campo da teoria dos grafos.

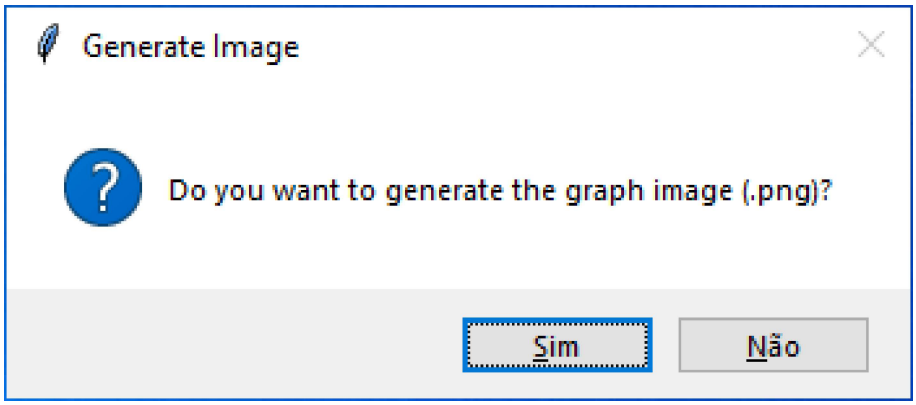
4 Resultados

4.1 Nova funcionalidade para a ferramenta

Como mencionado anteriormente, este trabalho propôs a introdução de uma nova funcionalidade à interface proposta por Tessaro (2024), voltada à geração automática de imagens representativas dos grafos construídos. Essa funcionalidade visa aprimorar a visualização das estruturas geradas, contribuindo para uma melhor compreensão de suas topologias e relações de adjacência.

Ao definir a quantidade de vértices para a criação de um novo grafo, o sistema solicita ao usuário a confirmação sobre o interesse em gerar, simultaneamente, a imagem correspondente, como exemplificado na Figura 16. Da mesma forma, quando uma operação é aplicada a um grafo já existente, a interface também oferece a opção de criação da imagem resultante da transformação. Os códigos completos dessa atualização encontram-se nos Apêndices A e B.

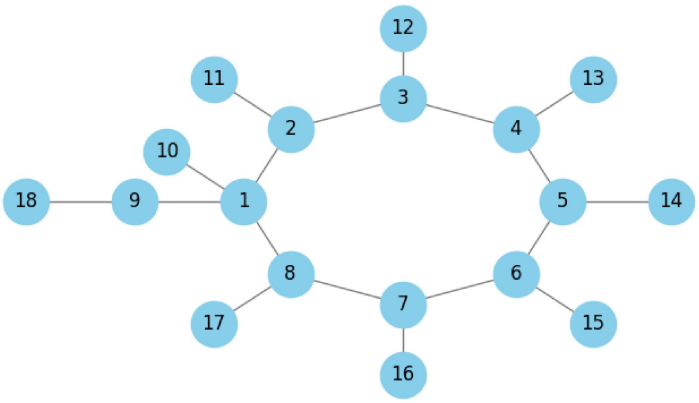
Figura 16 – Gerar imagem do grafo



Fonte: elaborado pelo autor.

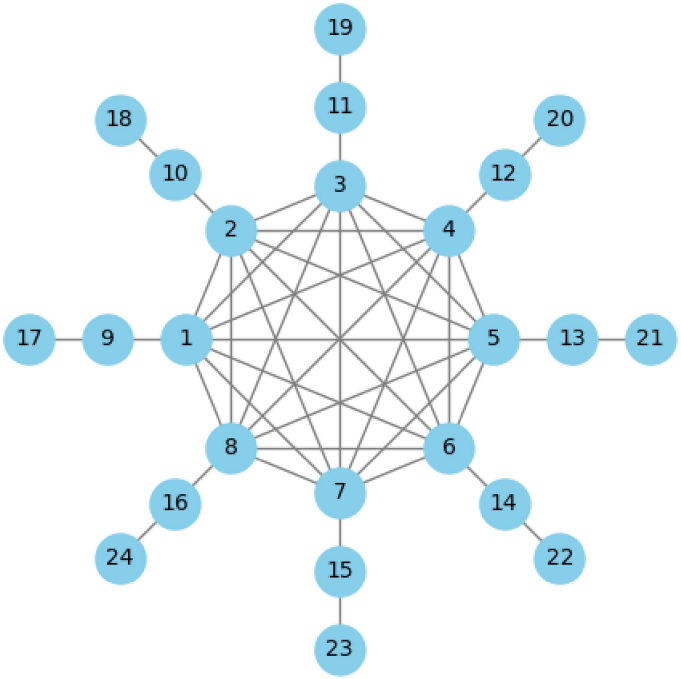
Seguindo essa lógica, as Figuras 17, 18 e 19 apresentam exemplos ilustrativos de grafos gerados com o auxílio dessa funcionalidade, evidenciando o aspecto visual das estruturas após a aplicação das respectivas operações.

Figura 17 – Ilustração de um *1-coronas-8-pan*



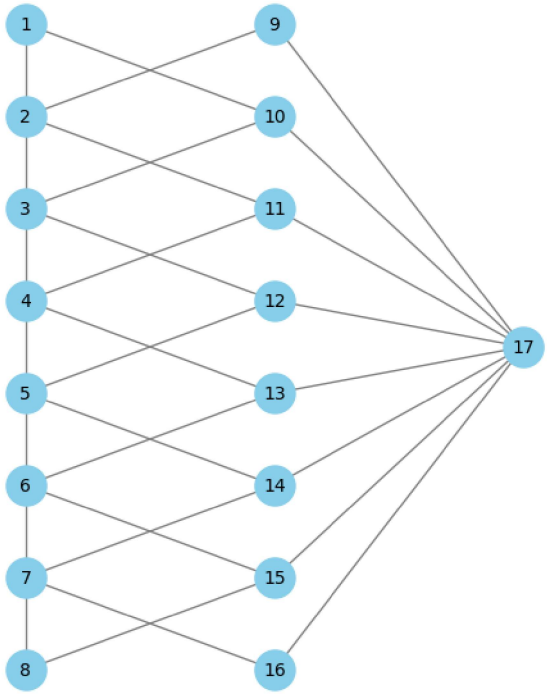
Fonte: elaborado pelo autor.

Figura 18 – Ilustração de um *2-coronas-8-clique*



Fonte: elaborado pelo autor.

Figura 19 – Ilustração de um *mycielski-8-path*



Fonte: elaborado pelo autor.

4.2 Catálogo de grafos e soluções

Ao término de todo o processo de criação dos grafos, definição dos problemas e obtenção de soluções, foi estruturado um repositório digital público do *Google Drive*. Esse repositório foi organizado em três pastas principais, classificadas de acordo com as categorias centrais do trabalho: “Grafos”, “Problemas” e “Soluções”.

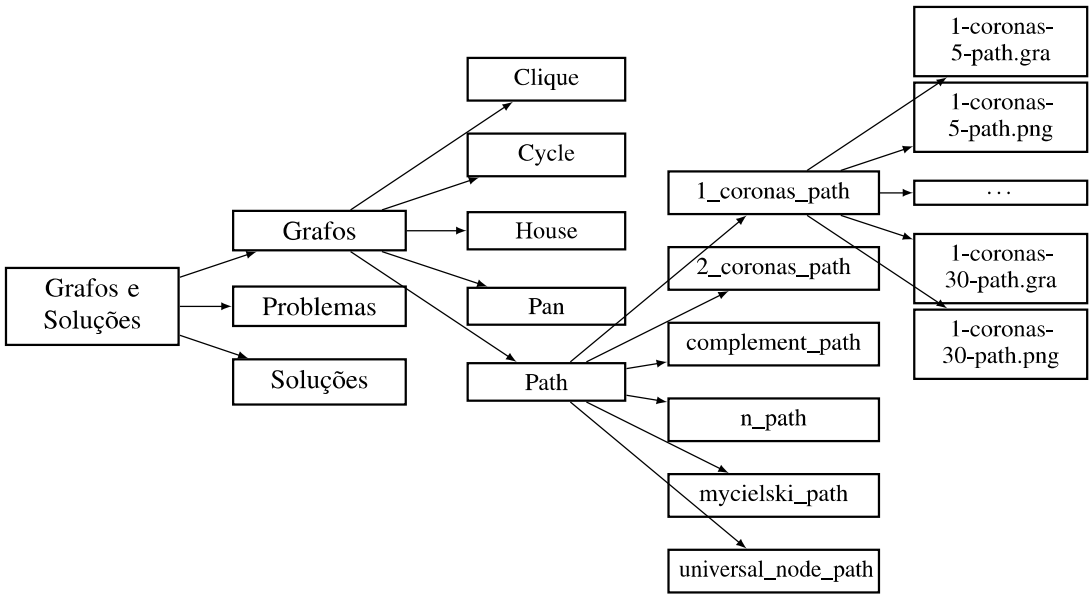
4.2.1 Pasta Grafos

A pasta intitulada “Grafos” está organizada segundo os cinco tipos de grafos abordados neste trabalho: *path, cycle, clique, pan* e *house*. Cada uma dessas categorias contém subpastas específicas referentes às operações aplicáveis, dentre as quais estão: *complement, universal node, 1-coronas, 2-coronas* e *mycielski*, além da estrutura original de cada grafo. Para esta etapa, optou-se por gerar grafos com quantidades de vértices variando de 5 até 30, a fim de permitir uma análise progressiva e controlada das estruturas, contemplando desde instâncias simples até grafos de maior complexidade.

O conteúdo dessas subpastas é apresentado em duas formas complementares: a primeira consiste em arquivos no formato .gra, contendo a representação do grafo por meio da lista de vizinhos; a segunda, em arquivos no formato .png, correspondentes à representação visual das estruturas geradas. Essa dupla abordagem visa facilitar tanto a leitura computacional quanto a análise visual dos grafos, contribuindo para a acessibilidade e compreensão dos dados disponibilizados.

Com o objetivo de facilitar a compreensão da estrutura adotada, a Figura 20 apresenta a hierarquização da pasta "Grafos", evidenciando a forma como os arquivos foram organizados segundo os tipos de grafos e as respectivas operações aplicadas.

Figura 20 – Hierarquia da pasta Grafos

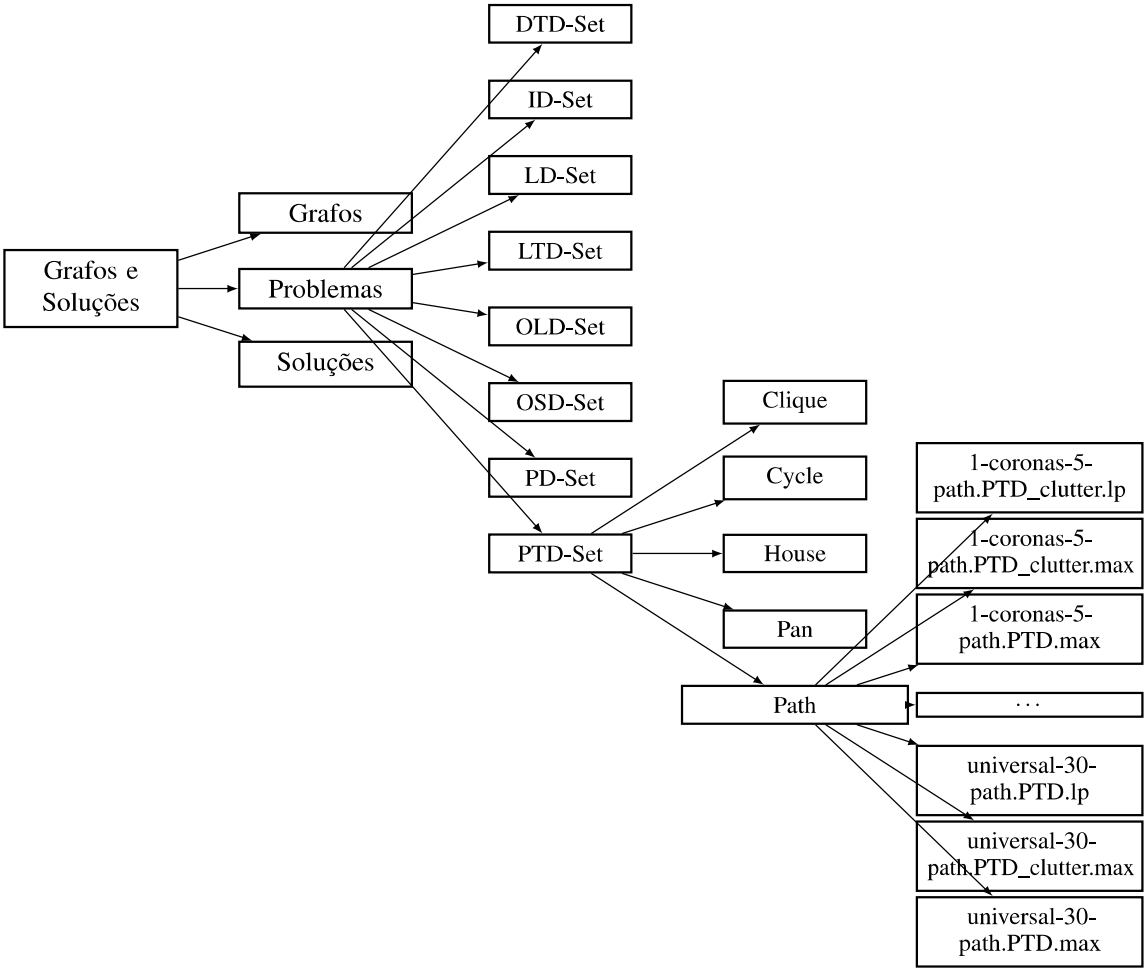


Fonte: elaborado pelo autor.

4.2.2 Pasta Problemas

A pasta intitulada "Problemas" está estruturada em oito subpastas, cada uma correspondente a um dos problemas abordados no Quadro 1, como apresentado na Figura 21. No interior de cada uma dessas subpastas, encontram-se novas divisões organizadas conforme os tipos de grafos utilizados na pesquisa. Para cada tipo de grafo, estão disponíveis os arquivos referentes ao problema em questão no formato .max, os respectivos clutters também no formato .max, bem como os arquivos contendo as inequações de entrada compatíveis com o *solver* *CPLEX*, salvos no formato .lp. Essa organização visa facilitar a navegação entre os diferentes níveis de abstração da modelagem, permitindo o acesso rápido tanto às representações estruturais quanto às formulações algébricas dos problemas.

Figura 21 – Hierarquia da pasta Problemas



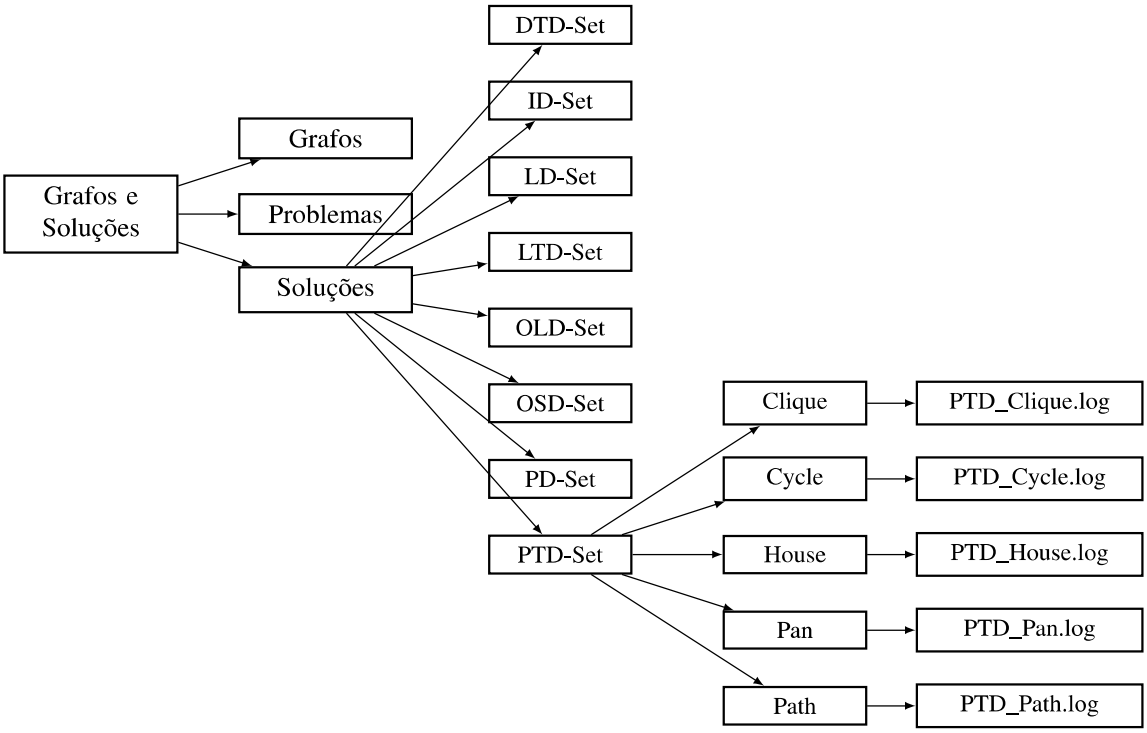
Fonte: Elaborado pelo autor.

4.2.3 Pasta Soluções

De forma análoga à estrutura adotada na categoria “Problemas”, a pasta intitulada “Soluções” também está organizada em oito subpastas, cada uma correspondente a um dos problemas apresentados no Quadro 1. No interior de cada uma dessas subpastas, encontram-se cinco pastas adicionais, relativas aos cinco tipos de grafos considerados no estudo. Diferentemente das demais seções do repositório, o conteúdo dessas pastas restringe-se a um único arquivo no formato .log, o qual reúne a solução ótima e as variáveis da solução para todas as instâncias geradas com grafos variando de 5 a 30 vértices. Esses arquivos constituem o registro dos resultados obtidos por meio do *solver CPLEX*.

Vale destacar que, para automatizar o processo de obtenção das soluções, foi desenvolvido um algoritmo capaz de ler todos os arquivos no formato .lp e resolvê-los no *solver* *CPLEX*. Em razão da lógica de funcionamento do algoritmo, os arquivos foram processados inicialmente a partir dos grafos com 10 vértices, seguindo em ordem crescente até os grafos com 30 vértices. Somente após essa etapa, foram lidos e solucionados os arquivos correspondentes aos grafos de 5 a 9 vértices. A estrutura hierárquica da pasta "Soluções", organizada conforme essa lógica, está ilustrada na Figura 22, evidenciando a distribuição dos arquivos por problema e tipo de grafo.

Figura 22 – Hierarquia da pasta Soluções



Fonte: Elaborado pelo autor.

4.3 Custo computacional

Para a análise desenvolvida neste trabalho, optou-se por investigar o custo computacional, avaliando o tempo de processamento para criação dos problemas e o tamanho do arquivo gerado em *quilobytes* (kB), através da interface gráfica. Inicialmente, foram realizados testes experimentais com instâncias contendo 10, 30, 100, 200, 300 e 500 vértices, aplicadas de forma sistemática aos oito problemas implementados na ferramenta e segundo os tipos de grafos e operações. Com base nos resultados obtidos nessa primeira etapa, dois problemas foram selecionados para análise comparativa mais aprofundada, levando em consideração critérios como tempo de execução e relevância estrutural para a discussão teórica proposta.

A escolha do problema LTD-*set* se justifica pelo fato de que, conforme citado por Boasquevisque (2023), diferenças simétricas entre vizinhanças de vértices não adjacentes sem vizinhos comuns são consideradas redundantes e, portanto, podem ser descartadas na construção do hipergrafo associado. Isso reduz significativamente o número de subconjuntos a serem processados, implicando em menor custo computacional. Assim como o LTD-*set*, problemas como {ID, LD, DTD, PTD e OLD} possuem essa característica, apresentando tempos similares nos testes realizados.

Por outro lado, o problema PD-*set*, assim como o problema OSD-*set*, pertence ao grupo de problemas nos quais tais redundâncias não podem ser eliminadas, exigindo a consideração de todos os pares de vértices, o que acarreta uma construção mais onerosa em termos de tempo de execução.

Com isso, foi realizado um comparativo do custo computacional entre esses dois problemas, utilizando como base diferentes instâncias de grafos. Essa comparação tem por objetivo evidenciar, de forma prática, o impacto das redundâncias no desempenho algorítmico na geração das hiperarrestas envolvidas em cada problema.

Vale destacar que, para a realização da análise proposta, também foram selecionados os tipos de grafos e as operações a serem consideradas. Optou-se por apresentar os custos computacionais associados à quatro das cinco classes de grafos em sua estrutura original, bem como após a aplicação da operação *mycielski*, possibilitando uma comparação entre o desempenho algorítmico antes e depois da transformação estrutural.

Essa escolha fundamenta-se na observação de que os grafos em sua estrutura original apresentaram os menores tempos de execução durante os teste. Enquanto isso, aqueles resultantes da aplicação da operação *mycielski* registraram os maiores custos computacionais, evidenciando um contraste relevante para fins de análise comparativa.

A classe de grafos do tipo *clique* foi excluída da análise devido à presença de vértices gêmeos idênticos em sua estrutura. Em um grafo do tipo *clique*, qualquer par de vértices compartilha exatamente o mesmo conjunto de vizinhança fechada, o que significa que todos os vértices são indistinguíveis sob esse critério. Essa característica inviabiliza a aplicação de problemas que envolvem propriedades de separação fechada ou localização de pares. Isso ocorre porque ambas as propriedades requerem uma diferença simétrica do conjunto de vizinhança fechada de todos os vértices adjacentes. Dessa forma, a presença de gêmeos idênticos levaria inevitavelmente à violação da condição básica da existência do conjunto.

Outro aspecto relevante a ser mencionado diz respeito à escolha pela etapa de criação dos problemas para a análise. Essa operação apresenta os maiores tempos de execução dentre todas as funcionalidades disponibilizadas pela interface gráfica. Em razão desse comportamento, a análise de desempenho computacional foi concentrada especificamente nessa funcionalidade.

Nas Tabelas 1 e 2 são apresentados os tempos de execução e o tamanho do arquivo obtido para os problemas LTD-*set* e PD-*set*, aplicados a grafos das classes *path* e *mycielski_path*, considerando diferentes instâncias em termos de número de vértices. Cabe destacar que, para algumas dessas instâncias, o tempo de processamento ultrapassou o limite máximo de 1800 segundos estipulado para os testes. Nesses casos, a execução foi interrompida e os resultados foram considerados inacessíveis, indicando elevado custo computacional ou inviabilidade prática de resolução dentro do tempo estabelecido.

Tabela 1 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *path*

Vértices	Tempo LTD_*path*	Tamanho LTD_*path*	Tempo PD_*path*	Tamanho PD_*path*
10	0.01	1	0.02	2
30	0.02	6	0.05	28
100	0.24	59	5.07	992
200	1.74	234	238.49	7872
300	4.72	527	1374.81	26500
500	24.18	1464	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tabela 2 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *mycielski_path*

Vértices	Tempo LTD_*path*	Tamanho LTD_*path*	Tempo PD_*path*	Tamanho PD_*path*
10	0.01	1	0.01	2
30	0.07	6	0.43	28
100	14.04	2378	204.84	7990
200	419.17	17325	*in**	*in**
300	*in**	*in**	*in**	*in**
500	*in**	*in**	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

De forma análoga, as Tabelas 3 e 4 apresentam os tempos de execução e tamanhos de arquivo obtidos para os grafos com estrutura do tipo *cycle*, considerando as mesmas instâncias utilizadas na análise anterior. Por fim, as Tabelas 5, 6, 7 e 8 exibem, respectivamente, os tempos de execução correspondentes aos grafos das classes *pan* e *house*, permitindo uma comparação sistemática do desempenho computacional entre as diferentes topologias estruturais avaliadas.

Tabela 3 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *cycle*

Vértices	Tempo LTD_*cycle*	Tamanho LTD_*cycle*	Tempo PD_*cycle*	Tamanho PD_*cycle*
10	0	1	0	2
30	0.04	6	0.08	28
100	0.3	59	6.91	992
200	1.81	235	266.87	7872
300	7.3	529	*in**	*in**
500	23.03	1467	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tabela 4 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *mycielski_cycle*

Vértices	**Tempo LTD_*cycle***	**Tamanho LTD_*cycle***	**Tempo PD_*cycle***	**Tamanho PD_*cycle***
10	0.02	7	0.04	10
30	0.14	93	0.28	228
100	14.09	2382	373.74	7990
200	661.03	17332	*in**	*in**
300	*in**	*in**	*in**	*in**
500	*in**	*in**	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tabela 5 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *pan*

Vértices	**Tempo LTD_*pan***	**Tamanho LTD_*pan***	**Tempo PD_*pan***	**Tamanho PD_*pan***
10	0.01	1	0.01	2
30	0.04	6	0.09	31
100	0.26	61	7.95	1022
200	1.65	238	308.34	7990
300	5.11	533	*in**	*in**
500	51.83	1474	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tabela 6 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *mycielski_pan*

Vértices	**Tempo LTD_*pan***	**Tamanho LTD_*pan***	**Tempo PD_*pan***	**Tamanho PD_*pan***
10	0.02	9	0.02	13
30	0.09	101	0.2	251
100	15.03	2451	192.56	8230
200	461.83	17587	*in**	*in**
300	*in**	*in**	*in**	*in**
500	*in**	*in**	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tabela 7 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *house*

Vértices	**Tempo LTD_*house***	**Tamanho LTD_*house***	**Tempo PD_*house***	**Tamanho PD_*house***
10	0.01	1	0.01	2
30	0.02	6	0.03	28
100	0.21	60	5.21	992
200	1.49	236	184.86	7872
300	4.92	530	1375.82	26500
500	21.97	1469	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

A análise dos tempos de execução e do tamanho dos arquivos gerados para os problemas LTD-*set* e PD-*set*, aplicados a grafos das classes *path, cycle, pan* e *house*, revela variações significativas em função do número de vértices e da aplicação da operação *mycielski*. De modo geral, constatou-se que os problemas do tipo PD-*set* apresentam um custo computacional substancialmente mais elevado em comparação aos problemas LTD-*set*, sobretudo em instâncias com maior quantidade de vértices.

Tabela 8 – Tempo de execução em segundos e tamanho em kB para os problemas com estrutura *mycielski_house*

Vértices	**Tempo LTD_*house***	**Tamanho LTD_*house***	**Tempo PD_*house***	**Tamanho PD_*house***
10	0.04	7	0.06	10
30	0.15	93	0.18	228
100	14.09	2384	177.77	7990
200	658.79	17336	*in**	*in**
300	*in**	*in**	*in**	*in**
500	*in**	*in**	*in**	*in**

*in**: tempo de execussão considerado inacessível.
Fonte: Elaborado pelo autor.

Tal comportamento pode ser atribuído à complexidade adicional inerente à formulação do PD-*set*, que demanda a localização de pares distintos por meio da análise combinatória das interseções entre os conjuntos de vizinhança dos vértices não pertencentes à solução. Esse requisito impõe uma sobrecarga computacional significativa ao algoritmo, impactando diretamente o tempo de execução e a escalabilidade da solução.

Adicionalmente, observou-se que a aplicação da operação *mycielski* exerceu um impacto significativo no aumento do tempo de execução e tamanho do arquivo. A aplicação dessa operação tornou diversas instâncias computacionalmente inviáveis, especialmente a partir de grafos com 200 vértices, cujos tempos ultrapassaram o limite previamente estabelecido para os testes.

Esse comportamento foi recorrente em todas as classes de grafos analisadas, com maior intensidade nos grafos do tipo *pan* e *house*, cujas estruturas menos regulares tendem a favorecer a geração de redundâncias e a complexificação dos caminhos de resolução. A irregularidade topológica dessas classes contribui para a elevação do custo computacional, uma vez que amplia o espaço de busca e a quantidade de combinações possíveis a serem avaliadas pelo algoritmo de resolução.

Os dados analisados também indicam que os grafos das classes *path* e *cycle*, em suas configurações originais, mantêm tempos de execução consideravelmente mais baixos para ambos os problemas avaliados. Esse resultado reforça a hipótese de que estruturas mais regulares e previsíveis tendem a favorecer a obtenção de soluções computacionalmente mais eficientes, em virtude da menor complexidade topológica envolvida.

No entanto, ao serem submetidas à operação *mycielski*, mesmo essas classes apresentaram um crescimento exponencial nos tempos de processamento, evidenciando a elevada sensibilidade dos algoritmos de resolução frente ao aumento estrutural e à complexificação das instâncias. Esse comportamento ressalta a importância de considerar não apenas o tipo de grafo, mas também as transformações aplicadas, ao se avaliar a viabilidade computacional de diferentes abordagens na resolução de problemas de dominação.

5 Considerações finais

Este trabalho teve como principal objetivo a criação de um catálogo de soluções para problemas de dominação em grafos, a partir da utilização de uma interface gráfica previamente desenvolvida. A iniciativa buscou contribuir com os estudos em teoria dos grafos, oferecendo uma base prática e sistematizada que possibilita a observação dos comportamentos computacionais associados a diferentes tipos de grafos e problemas combinatórios.

Além disso, este trabalho também se propôs a desenvolver e incorporar uma nova funcionalidade à interface gráfica, voltada à geração automática das representações visuais dos grafos criados. Tal aprimoramento contribuiu significativamente para tornar a ferramenta mais completa e didática, facilitando a interpretação estrutural dos grafos e ampliando seu potencial de uso em contextos acadêmicos, investigativos e computacionais.

Ao longo do trabalho, foram analisados problemas de combinação, em grafos do tipo *Path*, *Cycle*, *Clique*, *Pan* e *House*, tanto em suas versões originais quanto após a aplicação de operações.

A fundamentação teórica apresentou os conceitos centrais de dominação, localização e separação, bem como suas combinações, baseando-se em uma abordagem poliédrica. O levantamento bibliográfico demonstrou que tais propriedades possuem aplicações reais relevantes, como em sistemas de comunicação, monitoramento, redes de sensores e projetos de infraestrutura.

A análise dessas propriedades, quando combinadas, tal como destacado na literatura, mostra-se útil para a modelagem e a resolução de problemas complexos em redes distribuídas, em que o posicionamento estratégico de vértices ou sensores é essencial para garantir cobertura, distinção ou localização eficiente de eventos.

Os experimentos conduzidos com foco na análise do custo computacional possibilitaram a avaliação empírica do desempenho da ferramenta proposta por Tessaro (2024) ao resolver instâncias dos problemas *LTD-set* e *PD-set*. Os resultados indicaram um aumento expressivo no tempo de execução à medida que a ordem dos grafos cresce, especialmente após a aplicação da operação de *Mycielski*.

Essa operação, apesar de importante do ponto de vista teórico, mostrou-se um fator de elevação significativa da complexidade computacional, tornando inviável a resolução de instâncias maiores dentro do tempo limite estabelecido. Tais dados evidenciam a importância de estratégias mais eficientes, como a utilização de heurísticas ou técnicas de decomposição, para lidar com instâncias de maior porte.

Além disso, foi possível constatar que os problemas *PD-set*, por envolverem a localização de pares distintos de vértices, apresentaram tempos de execução consideravelmente superiores em relação aos problemas *LTD-set*. Esse comportamento reforça o entendimento de que diferentes combinações de propriedades implicam diferentes níveis de complexidade, fato que justifica a importância de estudos comparativos como o aqui proposto.

Também foi possível perceber que estruturas regulares, como grafos *Path* e *Cycle*, são mais manejáveis do ponto de vista computacional, ao passo que estruturas como *Pan* e *House*, por possuírem outras conectividades, impactam mais intensamente os algoritmos de resolução.

A base de dados produzida neste trabalho, contendo os grafos, os tipos de problemas aplicados e as soluções obtidas, representa uma boa alternativa de consulta para pesquisadores, professores e estudantes da área. Ao ser disponibilizada publicamente, ela amplia o alcance da pesquisa, democratiza o acesso à informação e possibilita o reuso em diferentes contextos acadêmicos e profissionais.

Por fim, este estudo reafirma a relevância da integração entre teoria e prática na pesquisa em grafos. Ao aliar um ambiente gráfico acessível a testes sistemáticos e bem documentados, o trabalho se insere em um contexto maior de fomento à ciência aplicada e à formação técnica especializada. Para trabalhos futuros, recomenda-se a ampliação da base de grafos com outros tipos estruturais, a análise de novas operações e transformações e, principalmente, a otimização dos algoritmos envolvidos, visando tornar a ferramenta ainda mais eficiente e robusta para o tratamento de grandes instâncias.

Referências

ARGIROFFO, G.; BIANCHI, S.; LUCARINI, Y.; WAGLER, A. Polyhedra associated with locating-dominating, open locating-dominating and locating total-dominating sets in graphs. **Discrete Applied Mathematics**, v. 322, p. 465–480, 2022. ISSN 0166-218X. Disponível em: <https://doi.org/10.1016/j.dam.2022.06.025>. Acesso em: 16 jan. 2025.

ARGIROFFO, G.; BIANCHI, S.; WAGLER, A. Progress on the description of identifying code polyhedra for some families of split graphs. **Discrete Optimization**, v. 22, p. 225–240, 2016. ISSN 1572-5286. SI: ISCO 2014. Disponível em: <https://doi.org/10.1016/j.disopt.2016.06.002>. Acesso em: 16 jan. 2025.

ARGIROFFO, G. R.; BIANCHI, S. M.; LUCARINI, Y. P.; WAGLER, A. K. Polyhedra associated with identifying codes in graphs. **Discrete Applied Mathematics**, v. 245, p. 16–27, 2018. ISSN 0166-218X. LAGOS'15 — Eighth Latin-American Algorithms, Graphs, and Optimization Symposium, Fortaleza, Brazil — 2015. Acesso em: 16 jan. 2025.

BABAI, L. On the complexity of canonical labeling of strongly regular graphs. **SIAM Journal on Computing**, SIAM, v. 9, n. 1, p. 212–216, 1980. Disponível em: <https://doi.org/10.1137/0209018>. Acesso em: 07 jul. 2025.

BALAKRISHNAN, R.; RANGANATHAN, K. **A Textbook of Graph Theory**. Springer New York, 2012. (Universitext). ISBN 9781441985057. Disponível em: <https://books.google.gg/books?id=Hl8ECAAAQBAJ>. Acesso em: 07 jul. 2025.

BARIK, S.; PATI, S.; SARMA, B. The spectrum of the corona of two graphs. **SIAM Journal on Discrete Mathematics**, SIAM, v. 21, n. 1, p. 47–56, 2007. Disponível em: <https://doi.org/10.1137/050624029>. Acesso em: 07 jul. 2025.

BOASQUEVISQUE, A. C. Développement d'un outil soutenant l'étude de problèmes de domination de graphes. Relatório de estágio, Université Clermont Auvergne, França, 2023. 38 f. 2023. Disponível em: <https://drive.google.com/file/d/17w4YRtlAQwLFtSypZ2cX_uVXPXEP3IWB/view?usp=drive_link>. Acesso em: 16 jan. 2025.

BONATO, A.; KEMKES, G.; PRAŁAT, P. Almost all cop-win graphs contain a universal vertex. **Discrete Mathematics**, v. 312, n. 10, p. 1652–1657, 2012. ISSN 0012-365X. Disponível em: <https://doi.org/10.1016/j.disc.2012.02.018>. Acesso em: 16 jan. 2025.

BRANDSTÄDT, A.; LE, V. B.; SPINRAD, J. P. **Graph Classes: A Survey**. SIAM, 1999. Disponível em: <http://dx.doi.org/10.1137/1.9780898719796>. Acesso em: 07 jul. 2025.

CHAKRABORTY, D.; FOUCAUD, F.; HAKANEN, A.; HENNING, M. A.; WAGLER, A. K. Progress towards the two-thirds conjecture on locating-total dominating sets. **Discrete Mathematics**, v. 347, n. 12, p. 114176, 2024. ISSN 0012-365X. Disponível em: <https://doi.org/10.1016/j.disc.2024.114176>. Acesso em: 16 jan. 2025.

CHAKRABORTY, D.; WAGLER, A. K. On open-separating dominating codes in graphs. **Discrete Applied Mathematics**, v. 375, p. 215–238, 2025. ISSN 0166-218X. Disponível em: <https://doi.org/10.1016/j.dam.2025.05.045>. Acesso em: 07 jul. 2025.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Introduction to algorithms (3rd edition). **MIT Press and McGraw-Hill**, 2009. Disponível em: <https://archive.org/details/introduction-to-algorithms-third-edition-2009>. Acesso em: 05 jul. 2025.

DE ALMEIDA, H. A. R.; MENDES, B. M. A.; TESSARO, A. F. D. P. Développement d'un outil soutenant l'étude de problèmes d'identification de graphes. Relatório de projeto, Université Clermont Auvergne, França, 2024. 12 f. 2024. Disponível em: <https://drive.google.com/file/d/16bUhier71_82fkbuD3NmNEVreBdDUi5B/view?usp=drive_link>. Acesso em: 15 jan. 2025.

ESCALANTE, F.; MONTEJANO, L.; ROJANO, T. Characterization of n-path graphs and of graphs having nth root. **Journal of Combinatorial Theory, Series B**, v. 16, n. 3, p. 282–289, 1974. ISSN 0095-8956. Disponível em: <https://doi.org/10.1016/0095-8956(74)90074-4>. Acesso em: 16 jan. 2025.

FERREIRA, L. P. **Grafos e Redes Sociais: Cliques e Relaxações de Cliques; Medidas de Centralidade**. 53 p. Dissertação (Mestrado em Estatística e Investigação Operacional) — Universidade de Lisboa (Portugal), Lisboa, 2020. Disponível em: <http://hdl.handle.net/10451/45762>. Acesso em: 07 jul. 2025.

FOUCAUD, F.; HENNING, M. A. Location-domination and matching in cubic graphs. **Discrete Mathematics**, v. 339, n. 4, p. 1221–1231, 2016. ISSN 0012-365X. Disponível em: <https://doi.org/10.1016/j.disc.2015.11.016>. Acesso em: 16 jan. 2025.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. Atlas, São Paulo, 2002. v. 4. Disponível em: <https://docente.ifrn.edu.br/mauriciofacanha/ensino-superior/redacao-cientifica/livros/gil-a.-c.-como-elaborar-projetos-de-pesquisa.-sao-paulo-atlas-2002./view>. Acesso em: 07 jul. 2025.

HAMEED, H. On semi-transitivity of (extended) mycielski graphs. **Discrete Applied Mathematics**, Elsevier, v. 359, p. 83–88, 2024. Disponível em: <https://doi.org/10.1016/j.dam.2024.07.028>. Acesso em: 05 jul. 2025.

HAYNES, T. W.; HEDETNIEMI, S.; SLATER, P. **Fundamentals of Domination in Graphs**. CRC Press, 2013. Disponível em: <https://doi.org/10.1201/9781482246582>. Acesso em: 07 jul. 2025.

JEAN, D. C.; SEO, S. J. On redundant locating-dominating sets. **Discrete Applied Mathematics**, v. 329, p. 106–125, 2023. ISSN 0166-218X. Disponível em: <https://doi.org/10.1016/j.dam.2023.01.023>. Acesso em: 06 jul. 2025.

JEAN, D. C.; SEO, S. J. Open-locating-dominating sets with error correction. In: **Proceedings of the 2024 ACM Southeast Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (ACMSE '24), p. 297–301. ISBN 9798400702372. Disponível em: <https://doi.org/10.1145/3603287.3651212>. Acesso em: 06 jul. 2025.

JR, S. R. C.; MALACAS, G. A.; TAREPE, D. Locating-dominating sets in graphs. **Applied Mathematical Sciences**, Hikari, Ltd., v. 8, n. 88, p. 4381–4388, 2014. Disponível em: <http://dx.doi.org/10.12988/ams.2014.46400>. Acesso em: 06 jul. 2025.

KARPOVSKY, M.; CHAKRABARTY, K.; LEVITIN, L. On a new class of codes for identifying vertices in graphs. **IEEE Transactions on Information Theory**, v. 44, n. 2, p. 599–611, 1998. Disponível em: <https://doi.org/10.1109/18.661507>. Acesso em: 07 jul. 2025.

LOZADA, L. A. P. **Tópicos na classe dos grafos clique**. 81 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Estadual de Campinas, Campinas, SP, 1996. Disponível em: <https://doi.org/10.47749/T/UNICAMP.1996.104726>. Acesso em: 07 jul. 2025.

MARCONI, M. d. A.; LAKATOS, E. M. **Metodologia Científica**. Atlas, São Paulo, 2003. v. 5. Disponível em: <https://docente.ifrn.edu.br/olivianeta/disciplinas/copy_of_historia-i/historia-ii/china-e-india/view>. Acesso em: 07 jul. 2025.

MORALES, J. D. A. **Dominação em grafos**. 189 p. Tese (Doutorado em Matemática) — Universidade Federal Fluminense, Niterói, RJ, 2019. Disponível em: <https://app.uff.br/riuff/handle/1/12705>. Acesso em: 07 jul. 2025.

PIKHURKO, O.; VEITH, H.; VERBITSKY, O. The first order definability of graphs: Upper bounds for quantifier depth. **Discrete Applied Mathematics**, v. 154, n. 17, p. 2511–2529, 2006. ISSN 0166-218X. Disponível em: <https://doi.org/10.1016/j.dam.2006.03.002>. Acesso em: 07 jul. 2025.

RAY, S.; UNGRANGSI, R.; PELLEGRINI, D.; TRACHTENBERG, A.; STAROBINSKI, D. Robust location detection in emergency sensor networks. In: IEEE. **IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)**. 2003. v. 2, p. 1044–1053. Disponível em: <https://doi.org/10.1109/INFCOM.2003.1208941>. Acesso em: 07 jul. 2025.

ROCHA, A. **Coloração arco-íris em classes de grafos**. 61 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Tecnológica Federal do Paraná, Ponta Grossa, PR, 2020. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/5232>. Acesso em: 07 jul. 2025.

SAMPATHKUMAR, E.; PUSHPALATHA, L. Complement of a graph: A generalization. **Graphs and Combinatorics**, Springer, v. 14, p. 377–392, 1998. Disponível em: <https://doi.org/10.1007/PL00021185>. Acesso em: 07 jul. 2025.

SANTOS, M. d. S. **Ciclos hamiltonianos em grafos**. 80 p. Dissertação (Mestrado em Matemática Aplicada) — Universidade Federal do Rio Grande do Sul, Porto Alegre RS, 2016. Disponível em: <http://hdl.handle.net/10183/150239>. Acesso em: 08 jul. 2025.

SILVA, W. G. d. **Conjuntos dominantes em grafos**. 86 p. Dissertação (Mestrado em Ciência da Computação) — Universidade de São Paulo, São Paulo, SP, 2010. Disponível em: <https://teses.usp.br/teses/disponiveis/45/45134/tde-20230727-113715/>. Acesso em: 08 jul. 2025.

SLATER, P. J. Domination and location in acyclic graphs. **Networks**, Wiley Online Library, v. 17, n. 1, p. 55–64, 1987. Disponível em: <https://doi.org/10.1002/net.3230170105>. Acesso em: 07 jul. 2025.

TELLE, J. A. Complexity of domination-type problems in graphs. **Nordic J. of Computing**, Publishing Association Nordic Journal of Computing, FIN, v. 1, n. 1, p. 157–171, 1994. ISSN 1236-6064. Disponível em: <https://dl.acm.org/doi/10.5555/640186.640195>. Acesso em: 07 jul. 2025.

TESSARO, A. F. D. P. Développement d'une interface pour des outils soutenant l'étude de problèmes d'identification de graphes. Relatório de estágio, Université Clermont Auvergne, França, 2024. 28 f. 2024. Disponível em: <https://drive.google.com/file/d/1MM4zuiT2UAb0YW7thpBZ36cj7oVv-W2Z/view?usp=drive_link>. Acesso em: 16 jan. 2025.

TÓTH, G. Note on geometric graphs. **Journal of Combinatorial Theory, Series A**, v. 89, n. 1, p. 126–132, 2000. ISSN 0097-3165. Disponível em: <https://doi.org/10.1006/jcta.1999.3001>. Acesso em: 07 jul. 2025.

UNGRANGSI, R.; TRACHTENBERG, A.; STAROBINSKI, D. An implementation of indoor location detection systems based on identifying codes. In: AAGESEN, F. A.; ANUTARIYA, C.; WUWONGSE, V. (Ed.). **International Conference on Intelligence in Communication Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 175–189. ISBN 978-3-540-30179-0. Disponível em: <https://doi.org/10.1007/978-3-540-30179-0_16>. Acesso em: 07 jul. 2025.

WEISSTEIN, E. W. **House Graph**. 2025. From MathWorld - A Wolfram Web Resource. Disponível em: <https://mathworld.wolfram.com/HouseGraph.html>. Acesso em: 16 jan. 2025.

WEISSTEIN, E. W. **Pan Graph**. 2025. From MathWorld - A Wolfram Web Resource. Disponível em: <https://mathworld.wolfram.com/PanGraph.html>. Acesso em: 16 jan. 2025.

XU, X.; LI, X.-Y.; SONG, M. Efficient aggregation scheduling in multihop wireless sensor networks with sinr constraints. **IEEE Transactions on Mobile Computing**, v. 12, n. 12, p. 2518–2528, 2013. Disponível em: <https://doi.org/10.1109/TMC.2012.245>. Acesso em: 08 jul. 2025.

APÊNDICE A – Código para gerar grafos com imagem

```
1
2     import networkx as nx
3     import matplotlib.pyplot as plt
4     import math
5
6     def generate_graph_image_from_file(filename):
7         G = nx.Graph()
8         with open(filename, 'r') as f:
9             lines = f.readlines()
10
11        reading_neighbors = False
12        for line in lines:
13            line = line.strip()
14            if line in ["NEIGHBORHOOD_LISTS", "NEIGHBORHOOD_LISTS:"
                  ]:
15                reading_neighbors = True
16                continue
17            if line == "END":
18                break
19            if reading_neighbors and line.startswith("N("):
20                vertex_part, neighbors_part = line.split('=')
21                vertex = int(vertex_part.strip()[2:-1])
22                neighbors = list(map(int, neighbors_part.strip().
                      split(',')))
23                for neighbor in neighbors:
24                    G.add_edge(vertex, neighbor)
25
26        n = len(G.nodes())
27        fig, ax = plt.subplots(figsize=(max(n * 0.8, 5), 5))
28
29        if '-path' in filename:
30            pos = {node: (i, 0) for i, node in enumerate(sorted(G.
                  nodes()))}
31            fig, ax = plt.subplots(figsize=(max(n * 0.8, 5), 2))
32        elif '-cycle' in filename or '-clique' in filename:
```

```
33              radius = 1.5
34              angle_step = 2 * math.pi / n
35              pos = {
36                  node: (
37                      radius * math.cos(math.pi - i * angle_step),
38                      radius * math.sin(math.pi - i * angle_step)
39                  )
40                  for i, node in enumerate(sorted(G.nodes()))
41              }
42              fig, ax = plt.subplots(figsize=(5, 5))
43          elif '-pan' in filename:
44              nodes = sorted(G.nodes())
45              base_nodes = [v for v in nodes if v != max(nodes)]
46              extra_node = max(nodes)
47              radius = 1.5
48              angle_step = 2 * math.pi / len(base_nodes)
49              pos = {
50                  node: (
51                      radius * math.cos(math.pi - i * angle_step),
52                      radius * math.sin(math.pi - i * angle_step)
53                  )
54                  for i, node in enumerate(base_nodes)
55              }
56              x1, y1 = pos[1]
57              pos[extra_node] = (x1 - 0.8, y1)
58              fig, ax = plt.subplots(figsize=(5, 5))
59          elif '-house' in filename:
60              nodes = sorted(G.nodes())
61              radius = 1.5
62              angle_step = 2 * math.pi / len(nodes)
63              pos = {}
64              for i, node in enumerate(nodes):
65                  angle = math.pi / 2 - i * angle_step
66                  r = radius + 0.6 if node == 1 else radius
67                  pos[node] = (r * math.cos(angle), r * math.sin(
                        angle))
68              fig, ax = plt.subplots(figsize=(5, 5))
69          else:
70              pos = nx.spring_layout(G, seed=42)
71              fig, ax = plt.subplots(figsize=(5, 5))
72
73          node_size = max(200, 1200 - (n * 40))
```

```
74          font_size = max(6, 14 - (n // 4))
75
76          nx.draw(
77              G, pos, ax=ax, with_labels=True,
78              node_color='skyblue', edge_color='gray',
79              node_size=node_size, font_size=font_size
80          )
81          ax.set_axis_off()
82          plt.tight_layout()
83          image_filename = filename.replace('.gra', '.png')
84          plt.savefig(image_filename, bbox_inches='tight', pad_inches
                =0.3)
85          plt.close()
86
87      def generate_n_path_file(n, generate_image=False):
88          filename = f"{n}-path.gra"
89          with open(filename, 'w') as file:
90              file.write(f"DIM = {n}\n\nNEIGHBORHOOD_LISTS:\n")
91              for i in range(1, n + 1):
92                  neighbors = [i + 1] if i == 1 else [i - 1] if i ==
                        n else [i - 1, i + 1]
93                  file.write(f"N({i}) = {', '.join(map(str, neighbors
                        ))}\n")
94              file.write("\nEND\n")
95          if generate_image:
96              generate_graph_image_from_file(filename)
97
98      def generate_n_cliques_file(n, generate_image=False):
99          filename = f"{n}-clique.gra"
100         with open(filename, 'w') as file:
101             file.write(f"DIM = {n}\n\nNEIGHBORHOOD_LISTS\n")
102             for i in range(1, n + 1):
103                 neighbors = [t for t in range(1, n + 1) if i != t]
104                 file.write(f"N({i}) = {', '.join(map(str, neighbors
                        ))}\n")
105             file.write("\nEND\n")
106         if generate_image:
107             generate_graph_image_from_file(filename)
108
109     def generate_n_cycle_file(n, generate_image=False):
110         filename = f"{n}-cycle.gra"
111         with open(filename, 'w') as file:
```

```
112             file.write(f"DIM = {n}\n\nNEIGHBORHOOD_LISTS\n")
113             for i in range(1, n + 1):
114                 if i == 1:
115                     neighbors = f"2, {n}"
116                 elif i == n:
117                     neighbors = f"{n-1}, 1"
118                 else:
119                     neighbors = f"{i-1}, {i+1}"
120                 file.write(f"N({i}) = {neighbors}\n")
121             file.write("\nEND")
122         if generate_image:
123             generate_graph_image_from_file(filename)
124 
125     def generate_n_pan_file(n, generate_image=False):
126         n_plus_one = n + 1
127         filename = f"{n}-pan.gra"
128         with open(filename, 'w') as file:
129             file.write(f"DIM = {n_plus_one}\n\nNEIGHBORHOOD_LISTS\n
                ")
130             for i in range(1, n_plus_one + 1):
131                 if i == 1:
132                     neighbors = f"2, {n}, {n_plus_one}"
133                 elif i == n:
134                     neighbors = f"{n-1}, 1"
135                 elif i == n_plus_one:
136                     neighbors = "1"
137                 else:
138                     neighbors = f"{i-1}, {i+1}"
139                 file.write(f"N({i}) = {neighbors}\n")
140             file.write("\nEND")
141         if generate_image:
142             generate_graph_image_from_file(filename)
143 
144     def generate_n_house_file(n, generate_image=False):
145         filename = f"{n}-house.gra"
146         with open(filename, 'w') as file:
147             file.write(f"DIM = {n}\n\nNEIGHBORHOOD_LISTS\n")
148             for i in range(1, n + 1):
149                 if i == 1:
150                     neighbors = f"2, {n}"
151                 elif i == 2:
152                     neighbors = f"1, 3, {n}"
```

```
153                 elif i == n:
154                     neighbors = f"1, 2, {n-1}"
155                 else:
156                     neighbors = f"{i-1}, {i+1}"
157                 file.write(f"N({i}) = {neighbors}\n")
158             file.write("\nEND")
159         if generate_image:
160             generate_graph_image_from_file(filename)
```

APÊNDICE B – Código para aplicar operações com imagem

```
1
2     import networkx as nx
3     import matplotlib.pyplot as plt
4     import copy
5     import math
6     import os
7
8     def write_output(output_filename, dim, neighborhood_lists):
9         with open(output_filename, 'w') as f:
10            f.write(f"DIM = {dim}\n\nNEIGHBORHOOD_LISTS:\n")
11            for i in range(1, dim + 1):
12                neighbors = ', '.join(map(str, sorted(
                      neighborhood_lists[i])))
13                f.write(f"N({i}) = {neighbors}\n")
14            f.write("\nEND")
15
16    def read_file(filename):
17        try:
18            with open(filename, 'r') as file:
19                graph_data = {'DIM': 0, 'NEIGHBORHOOD_LISTS': {}}
20                for line in file:
21                    parts = line.split('=')
22                    if parts[0].strip() == 'DIM':
23                        graph_data['DIM'] = int(parts[1].strip())
24                    elif parts[0].strip().startswith('N('):
25                        node = int(parts[0].strip()[2:-1])
26                        neighbors = [int(x) for x in parts[1].strip
                              ().split(',') if x.strip()]
27                        graph_data['NEIGHBORHOOD_LISTS'][node] =
                              neighbors
28                return graph_data if graph_data['DIM'] and
                      graph_data['NEIGHBORHOOD_LISTS'] else None
29        except FileNotFoundError:
30            print(f"Error: File '{filename}' not found.")
31            return None
```

```
32
33      def generate_graph_image_from_file(filename):
34          G = nx.Graph()
35          with open(filename, 'r') as f:
36              lines = f.readlines()
37
38          reading = False
39          for line in lines:
40              line = line.strip()
41              if line in ["NEIGHBORHOOD_LISTS", "NEIGHBORHOOD_LISTS:"
                    ]:
42                  reading = True
43                  continue
44              if line == "END":
45                  break
46              if reading and line.startswith("N("):
47                  v = int(line.split('=')[0].strip()[2:-1])
48                  neighbors = list(map(int, line.split('=')[1].strip
                        ().split(',')))
49                  for u in neighbors:
50                      G.add_edge(v, u)
51
52          pos = nx.spring_layout(G, seed=42)
53          n = len(G.nodes())
54          node_size = max(200, 1200 - (n * 40))
55          font_size = max(6, 14 - (n // 4))
56
57          fig, ax = plt.subplots(figsize=(max(n * 0.6, 6), 6))
58          nx.draw(G, pos, ax=ax, with_labels=True,
59                  node_color='skyblue', edge_color='gray',
60                  node_size=node_size, font_size=font_size)
61          ax.set_axis_off()
62          plt.tight_layout()
63          image_file = filename.replace('.gra', '.png')
64          plt.savefig(image_file, bbox_inches='tight', pad_inches
                =0.3)
65          plt.close()
66
67
68      def generate_graph_image(G, output_filename):
69          dim = len(G.nodes())
70          pos = nx.spring_layout(G, seed=42, k=1.5 / (dim ** 0.5))
```

```
71          node_size = max(200, 1200 - (dim * 40))
72          font_size = max(6, 14 - (dim // 4))
73
74          fig, ax = plt.subplots(figsize=(6, 6))
75          nx.draw(G, pos, ax=ax, with_labels=True,
76                  node_color='skyblue', edge_color='gray',
77                  node_size=node_size, font_size=font_size)
78          ax.set_axis_off()
79          plt.tight_layout()
80          image_file = output_filename.replace('.gra', '.png')
81          plt.savefig(image_file, bbox_inches='tight', pad_inches
                =0.3)
82          plt.close()
83
84
85      def complement(filename, generate_image=False):
86
87          if '-clique' in filename:
88              return
89
90          graph = read_file(filename)
91          if graph:
92              dim = graph['DIM']
93              original_neighbors = graph['NEIGHBORHOOD_LISTS']
94
95              G = nx.Graph()
96              for i in range(1, dim + 1):
97                  for j in original_neighbors[i]:
98                      G.add_edge(i, j)
99
100             G_complement = nx.complement(G)
101
102             complement = {}
103             for i in G_complement.nodes():
104                 complement[i] = sorted(list(G_complement.neighbors(
                        i)))
105
106             output = f"co-{filename}"
107             write_output(output, dim, complement)
108
109
110             if generate_image:
```

```
111                 generate_graph_image(G_complement, output)
112
113
114     def add_universal_node(filename, generate_image=False):
115         graph = read_file(filename)
116         if graph:
117             dim = graph['DIM']
118             neighbors = copy.deepcopy(graph['NEIGHBORHOOD_LISTS'])
119
120             u = dim + 1
121             for i in range(1, dim + 1):
122                 neighbors[i].append(u)
123             neighbors[u] = list(range(1, dim + 1))
124
125             output_filename = f"universal-{filename}"
126             write_output(output_filename, dim + 1, neighbors)
127
128             if generate_image:
129                 G = nx.Graph()
130                 for v in neighbors:
131                     for neighbor in neighbors[v]:
132                         G.add_edge(v, neighbor)
133
134                 if '-path' in filename:
135                     n = dim
136                     fig, ax = plt.subplots(figsize=(max(n * 0.8, 5)
                            , 3))
137                     pos = {i: (i - 1, 0) for i in range(1, n + 1)}
138                     pos[u] = ((n - 1) / 2, -2)
139
140                 elif '-cycle' in filename:
141                     n = dim
142                     fig, ax = plt.subplots(figsize=(5, 5))
143                     radius = 1.5 / (2 * math.sin(math.pi / dim))
144                     angle_step = 2 * math.pi / n
145                     pos = {}
146                     for i, node in enumerate(sorted(G.nodes())
                            [:-1]):
147                         angle = math.pi - i * angle_step
148                         pos[node] = (radius * math.cos(angle),
                                radius * math.sin(angle))
149                     pos[u] = (0, 0)
```

```
150
151                 elif '-clique' in filename:
152                     n = dim
153                     fig, ax = plt.subplots(figsize=(5, 5))
154                     radius = 1.5 / (2 * math.sin(math.pi / dim))
155                     angle_step = 2 * math.pi / n
156                     pos = {}
157                     for i, node in enumerate(sorted(G.nodes())
                            [:-1]):
158                         angle = math.pi - i * angle_step
159                         pos[node] = (radius * math.cos(angle),
                                radius * math.sin(angle))
160                     pos[u] = (0, 0)
161
162                 elif '-pan' in filename:
163                     fig, ax = plt.subplots(figsize=(5, 5))
164                     nodes = sorted(G.nodes())
165                     base_nodes = [v for v in nodes if v != max(
                            nodes) and v != u]
166                     extra_node = max([v for v in nodes if v != u])
167                     radius = 1.5 / (2 * math.sin(math.pi / dim))
168                     angle_step = 2 * math.pi / len(base_nodes)
169                     pos = {}
170                     for idx, node in enumerate(base_nodes):
171                         angle = math.pi - idx * angle_step
172                         pos[node] = (radius * math.cos(angle),
                                radius * math.sin(angle))
173                     x1, y1 = pos[1]
174                     pos[extra_node] = (x1 - 1.0, y1)
175                     center_x = sum(x for x, y in pos.values()) /
                            len(pos)
176                     center_y = sum(y for x, y in pos.values()) /
                            len(pos)
177                     pos[u] = (center_x, center_y)
178
179                 elif '-house' in filename:
180                     fig, ax = plt.subplots(figsize=(5, 5))
181                     nodes = sorted(G.nodes())
182                     base_nodes = [v for v in nodes if v != u]
183                     radius = 1.5 / (2 * math.sin(math.pi / dim))
184                     angle_step = 2 * math.pi / len(base_nodes)
185                     pos = {}
```

```
186                     for idx, node in enumerate(base_nodes):
187                         angle = math.pi / 2 - idx * angle_step
188                         r = radius + 0.6 if node == 1 else radius
189                         pos[node] = (r * math.cos(angle), r * math.
                                sin(angle))
190                     center_x = sum(x for x, y in pos.values()) /
                            len(pos)
191                     center_y = sum(y for x, y in pos.values()) /
                            len(pos)
192                     pos[u] = (center_x, center_y)
193
194                 else:
195                     fig, ax = plt.subplots(figsize=(5, 5))
196                     pos = nx.spring_layout(G, seed=42)
197
198                 node_size = max(200, 1200 - (dim * 40))
199                 font_size = max(6, 14 - (dim // 4))
200                 nx.draw(G, pos, ax=ax, with_labels=True,
201                         node_color='skyblue', edge_color='gray',
202                         node_size=node_size, font_size=font_size)
203                 ax.set_axis_off()
204                 plt.tight_layout()
205                 image_file = output_filename.replace('.gra', '.png'
                        )
206                 plt.savefig(image_file, bbox_inches='tight',
                        pad_inches=0.3)
207                 plt.close()
208
209
210
211     def one_coronas(filename, generate_image=False):
212         graph = read_file(filename)
213         if graph:
214             dim = graph['DIM']
215             neighbors = copy.deepcopy(graph['NEIGHBORHOOD_LISTS'])
216             new_neighbors = copy.deepcopy(neighbors)
217             new_node = dim + 1
218
219
220             for i in range(1, dim + 1):
221                 new_neighbors[i].append(new_node)
222                 new_neighbors[new_node] = [i]
```

```
223                 new_node += 1
224
225             new_dim = new_node - 1
226             output_filename = f"1-coronas-{filename}"
227             write_output(output_filename, new_dim, new_neighbors)
228
229             if generate_image:
230                 # Construcao do grafo para desenhar
231                 G = nx.Graph()
232                 for v in new_neighbors:
233                     for neighbor in new_neighbors[v]:
234                         G.add_edge(v, neighbor)
235
236                 fig, ax = plt.subplots(figsize=(max(dim * 0.8, 5),
                        4))
237
238                 # Verificacao do tipo de grafo
239                 if '-path' in filename:
240                     pos = {}
241                     altura = 0.5
242                     original_nodes = list(range(1, dim + 1))
243                     added_nodes = list(range(dim + 1, dim * 2 + 1))
244
245                     for i, original in enumerate(original_nodes):
246                         x = i
247                         pos[original] = (x, 0)
248                         pos[added_nodes[i]] = (x, -0.8)
249
250                 elif '-cycle' in filename:
251                     radius = 1.5 / (2 * math.sin(math.pi / dim))
252                     offset = 1.5
253                     angle_step = 2 * math.pi / dim
254                     pos = {}
255
256                     original_nodes = list(range(1, dim + 1))
257                     added_nodes = list(range(dim + 1, dim * 2 + 1))
258
259                     for i, node in enumerate(original_nodes):
260                         angle = math.pi - i * angle_step
261                         x = radius * math.cos(angle)
262                         y = radius * math.sin(angle)
263                         pos[node] = (x, y)
```

```
264
265                         x_offset = (radius + offset) * math.cos(
                                angle)
266                         y_offset = (radius + offset) * math.sin(
                                angle)
267                         pos[added_nodes[i]] = (x_offset, y_offset)
268
269
270                 elif '-clique' in filename:
271                     base_nodes = list(range(1, dim + 1))
272                     corona_nodes = list(range(dim + 1, dim * 2 + 1)
                            )
273                     pos = {}
274
275                     radius = 1.5 / (2 * math.sin(math.pi / dim))
276                     extra_distance = 1.5
277                     angle_step = 2 * math.pi / dim
278
279                     for i, node in enumerate(base_nodes):
280                         angle = math.pi - i * angle_step
281                         x_base = radius * math.cos(angle)
282                         y_base = radius * math.sin(angle)
283                         x_corona = (radius + extra_distance) * math
                                .cos(angle)
284                         y_corona = (radius + extra_distance) * math
                                .sin(angle)
285
286                         pos[node] = (x_base, y_base)
287                         pos[corona_nodes[i]] = (x_corona, y_corona)
288
289                 elif '-pan' in filename:
290                     nodes = sorted(G.nodes())
291                     original_nodes = list(range(1, dim + 1))
292                     corona_nodes = list(range(dim + 1, dim * 2 + 1)
                            )
293                     base_nodes = [v for v in original_nodes if v !=
                            max(original_nodes)]
294                     extra_node = max(original_nodes)
295                     radius = 1.5 / (2 * math.sin(math.pi / dim))
296                     extra_distance = 1.5
297                     angle_step = 2 * math.pi / len(base_nodes)
298                     pos = {}
```

```
299
300                     for idx, node in enumerate(base_nodes):
301                         angle = math.pi - idx * angle_step
302                         x = radius * math.cos(angle)
303                         y = radius * math.sin(angle)
304                         pos[node] = (x, y)
305
306                         if node == 1:
307                             adjusted_angle = math.radians(135)
308                             x_extra = x + extra_distance * math.cos
                                (adjusted_angle)
309                             y_extra = y + extra_distance * math.sin
                                (adjusted_angle)
310                         else:
311                             x_extra = (radius + extra_distance) *
                                math.cos(angle)
312                             y_extra = (radius + extra_distance) *
                                math.sin(angle)
313
314                         pos[corona_nodes[idx]] = (x_extra, y_extra)
315
316                     x1, y1 = pos[1]
317                     pos[extra_node] = (x1 - 1.5, y1)
318                     pos[corona_nodes[-1]] = (x1 - 3.0, y1)
319
320
321                 elif '-house' in filename:
322                     original_nodes = list(range(1, dim + 1))
323                     corona_nodes = list(range(dim + 1, dim * 2 + 1)
                        )
324                     pos = {}
325
326                     radius = 1.5 / (2 * math.sin(math.pi / dim))
327                     corona_offset = 1.5
328                     angle_step = 2 * math.pi / dim
329
330                     for i, node in enumerate(original_nodes):
331                         angle = math.pi / 2 - i * angle_step
332                         r = radius + 0.6 if node == 1 else radius
333
334                         x = r * math.cos(angle)
335                         y = r * math.sin(angle)
```

```
336                         pos[node] = (x, y)
337
338                         x_corona = (r + corona_offset) * math.cos(
                               angle)
339                         y_corona = (r + corona_offset) * math.sin(
                               angle)
340                         pos[corona_nodes[i]] = (x_corona, y_corona)
341
342                 else:
343
344                     pos = nx.spring_layout(G, seed=42)
345
346                 # Desenho
347                 node_size = max(200, 1200 - (dim * 40))
348                 font_size = max(6, 14 - (dim // 4))
349                 nx.draw(
350                     G, pos, ax=ax, with_labels=True,
351                     node_color='skyblue', edge_color='gray',
352                     node_size=node_size, font_size=font_size
353                 )
354                 ax.set_axis_off()
355                 plt.tight_layout()
356                 image_file = output_filename.replace('.gra', '.png'
                       )
357                 plt.savefig(image_file, bbox_inches='tight',
                       pad_inches=0.3)
358                 plt.close()
359
360
361     def two_coronas(filename, generate_image=False):
362         graph = read_file(filename)
363
364         if graph:
365             dim = graph['DIM']
366             neighbors = copy.deepcopy(graph['NEIGHBORHOOD_LISTS'])
367             new_neighbors = copy.deepcopy(neighbors)
368             new_node = dim + 1
369
370             # Construcao dos novos nos: v -> v' -> v''
371             for i in range(1, dim + 1):
372                 corona1 = new_node
373                 corona2 = new_node + dim
```

```
374                 new_neighbors[i].append(corona1)
375                 new_neighbors[corona1] = [i, corona2]
376                 new_neighbors[corona2] = [corona1]
377                 new_node += 1
378
379             new_dim = max(new_neighbors.keys())
380             output_filename = f"2-coronas-{filename}"
381             write_output(output_filename, new_dim, new_neighbors)
382
383             if generate_image:
384
385                 G = nx.Graph()
386                 for v in new_neighbors:
387                     for u in new_neighbors[v]:
388                         G.add_edge(v, u)
389                 # Geracao da imagem para o tipo PATH
390                 if '-path' in filename:
391
392                     pos = {}
393                     altura1 = -0.8
394                     altura2 = -1.6
395                     for i in range(1, dim + 1):
396                         x = i - 1
397                         corona1 = i + dim
398                         corona2 = i + 2 * dim
399                         pos[i] = (x, 0)
400                         pos[corona1] = (x, altura1)
401                         pos[corona2] = (x, altura2)
402
403                     fig, ax = plt.subplots(figsize=(max(dim * 0.8,
                            5), 5))
404                     nx.draw(
405                         G, pos, ax=ax, with_labels=True,
406                         node_color='skyblue', edge_color='gray',
407                         node_size=500, font_size=10
408                     )
409                     ax.set_axis_off()
410                     plt.tight_layout()
411                     image_file = output_filename.replace('.gra', '.
                            png')
412                     plt.savefig(image_file, bbox_inches='tight',
                            pad_inches=0.3)
```

```
413                     plt.close()
414
415
416                 elif '-cycle' in filename:
417                     pos = {}
418                     edge_length = 1.5  # ou 2.0 ou o valor que
                           quiser como unidade de distancia
419                     radius = edge_length / (2 * math.sin(math.pi /
                           dim))
420                     offset1 = 1.5  # distancia da 1 camada
421                     offset2 = 3.0  # distancia da 2 camada
422                     angle_step = 2 * math.pi / dim
423
424                     for i in range(1, dim + 1):
425                         angle = math.pi - (i - 1) * angle_step
426                         # No original
427                         x0 = radius * math.cos(angle)
428                         y0 = radius * math.sin(angle)
429                         pos[i] = (x0, y0)
430
431                         # No v'
432                         r1 = radius + offset1
433                         x1 = r1 * math.cos(angle)
434                         y1 = r1 * math.sin(angle)
435                         pos[i + dim] = (x1, y1)
436
437                         # No v''
438                         r2 = radius + offset2
439                         x2 = r2 * math.cos(angle)
440                         y2 = r2 * math.sin(angle)
441                         pos[i + 2 * dim] = (x2, y2)
442
443                     figsize_scale = max(5, dim * 0.3)
444                     fig, ax = plt.subplots(figsize=(figsize_scale,
                           figsize_scale))
445
446                     nx.draw(
447                         G, pos, ax=ax, with_labels=True,
448                         node_color='skyblue', edge_color='gray',
449                         node_size=500, font_size=10
450                     )
451                     ax.set_axis_off()
```

```
452                     plt.tight_layout()
453                     image_file = output_filename.replace('.gra', '.
                            png')
454                     plt.savefig(image_file, bbox_inches='tight',
                            pad_inches=0.3)
455                     plt.close()
456
457
458                 elif '-clique' in filename:
459                     pos = {}
460                     edge_length = 1.5  # ou 2.0 ou o valor que
                            quiser como unidade de distancia
461                     radius = edge_length / (2 * math.sin(math.pi /
                            dim))
462
463                     offset1 = 1.0
464                     offset2 = 2.0
465                     angle_step = 2 * math.pi / dim
466
467                     for i in range(1, dim + 1):
468                         angle = math.pi - (i - 1) * angle_step
469                         x0 = radius * math.cos(angle)
470                         y0 = radius * math.sin(angle)
471                         pos[i] = (x0, y0)
472
473                         # v'
474                         r1 = radius + offset1
475                         x1 = r1 * math.cos(angle)
476                         y1 = r1 * math.sin(angle)
477                         pos[i + dim] = (x1, y1)
478
479                         # v''
480                         r2 = radius + offset2
481                         x2 = r2 * math.cos(angle)
482                         y2 = r2 * math.sin(angle)
483                         pos[i + 2 * dim] = (x2, y2)
484
485
486                     figsize_scale = max(5, dim * 0.5)
487                     fig, ax = plt.subplots(figsize=(figsize_scale,
                            figsize_scale))
488                     nx.draw(
```

```
489                         G, pos, ax=ax, with_labels=True,
490                         node_color='skyblue', edge_color='gray',
491                         node_size=500, font_size=10
492                     )
493                     ax.set_axis_off()
494                     plt.tight_layout()
495                     image_file = output_filename.replace('.gra', '.
                            png')
496                     plt.savefig(image_file, bbox_inches='tight',
                            pad_inches=0.3)
497                     plt.close()
498
499
500                 elif '-pan' in filename:
501                     pos = {}
502                     original_nodes = list(range(1, dim + 1))
503                     base_nodes = [v for v in original_nodes if v !=
                            max(original_nodes)]
504                     extra_node = max(original_nodes)
505
506                     radius = 1.5 / (2 * math.sin(math.pi / dim))
507                     offset1 = 1.5
508                     offset2 = 3.0
509                     angle_step = 2 * math.pi / len(base_nodes)
510
511                     for idx, node in enumerate(base_nodes):
512                         angle = math.pi - idx * angle_step
513                         x = radius * math.cos(angle)
514                         y = radius * math.sin(angle)
515                         pos[node] = (x, y)
516
517                         if node == 1:
518                             adj_angle = math.radians(135)
519                             x1 = x + offset1 * math.cos(adj_angle)
520                             y1 = y + offset1 * math.sin(adj_angle)
521                             x2 = x + offset2 * math.cos(adj_angle)
522                             y2 = y + offset2 * math.sin(adj_angle)
523                         else:
524                             x1 = (radius + offset1) * math.cos(
                                    angle)
525                             y1 = (radius + offset1) * math.sin(
                                    angle)
```

```
526                             x2 = (radius + offset2) * math.cos(
                                    angle)
527                             y2 = (radius + offset2) * math.sin(
                                    angle)
528
529                         pos[node + dim] = (x1, y1)
530                         pos[node + 2 * dim] = (x2, y2)
531
532                     x1, y1 = pos[1]
533                     pos[extra_node] = (x1 - 1.5, y1)
534                     pos[extra_node + dim] = (x1 - 3.0, y1)
535                     pos[extra_node + 2 * dim] = (x1 - 4.5, y1)
536
537                     # Cabo
538                     x1, y1 = pos[1]
539                     pos[extra_node] = (x1 - 1.5, y1)
540                     pos[extra_node + dim] = (x1 - 3.0, y1)
541                     pos[extra_node + 2 * dim] = (x1 - 4.5, y1)
542
543                     figsize_scale = max(5, dim * 0.3)
544                     fig, ax = plt.subplots(figsize=(figsize_scale,
                            figsize_scale))
545                     nx.draw(
546                         G, pos, ax=ax, with_labels=True,
547                         node_color='skyblue', edge_color='gray',
548                         node_size=500, font_size=10
549                     )
550                     ax.set_axis_off()
551                     plt.tight_layout()
552                     image_file = output_filename.replace('.gra', '.
                            png')
553                     plt.savefig(image_file, bbox_inches='tight',
                            pad_inches=0.3)
554                     plt.close()
555
556
557                 elif '-house' in filename:
558                     pos = {}
559                     original_nodes = list(range(1, dim + 1))
560                     radius = 1.5 / (2 * math.sin(math.pi / dim))
561                     offset1 = 1.5
562                     offset2 = 3.0
```

```
563                     angle_step = 2 * math.pi / dim
564
565                     for i, node in enumerate(original_nodes):
566                         angle = math.pi / 2 - i * angle_step
567                         r = radius + 0.6 if node == 1 else radius
568
569                         x0 = r * math.cos(angle)
570                         y0 = r * math.sin(angle)
571                         pos[node] = (x0, y0)
572
573                         # v'
574                         r1 = r + offset1
575                         x1 = r1 * math.cos(angle)
576                         y1 = r1 * math.sin(angle)
577                         pos[node + dim] = (x1, y1)
578
579                         # v''
580                         r2 = r + offset2
581                         x2 = r2 * math.cos(angle)
582                         y2 = r2 * math.sin(angle)
583                         pos[node + 2 * dim] = (x2, y2)
584
585                     figsize_scale = max(5, dim * 0.3)
586                     fig, ax = plt.subplots(figsize=(figsize_scale,
                            figsize_scale))
587                     nx.draw(
588                         G, pos, ax=ax, with_labels=True,
589                         node_color='skyblue', edge_color='gray',
590                         node_size=500, font_size=10
591                     )
592                     ax.set_axis_off()
593                     plt.tight_layout()
594                     image_file = output_filename.replace('.gra', '.
                            png')
595                     plt.savefig(image_file, bbox_inches='tight',
                            pad_inches=0.3)
596                     plt.close()
597
598
599                 else:
600                     generate_graph_image_from_file(output_filename)
601
```

```
602     def mycielski(filename, generate_image=False):
603         graph = read_file(filename)
604         if graph:
605             dim = graph['DIM']
606             neighbors = graph['NEIGHBORHOOD_LISTS']
607             new_dim = 2 * dim + 1
608             new_neighbors = {}
609 
610             for i in range(1, dim + 1):
611                 new_neighbors[i] = neighbors[i] + [j + dim for j in
                        neighbors[i]]
612                 new_neighbors[i + dim] = neighbors[i] + [new_dim]
613             new_neighbors[new_dim] = list(range(dim + 1, new_dim))
614 
615             output_filename = f"mycielski-{filename}"
616             write_output(output_filename, new_dim, new_neighbors)
617 
618             if generate_image:
619                 G = nx.Graph()
620                 for v in new_neighbors:
621                     for u in new_neighbors[v]:
622                         G.add_edge(v, u)
623 
624                 pos = {}
625 
626                 if "-path" in filename:
627                     x_original = 0
628                     x_copia = 1.0
629                     x_z = 2.0
630                     ys = []
631                     for i in range(1, dim + 1):
632                         y = dim - i
633                         ys.append(y)
634                         pos[i] = (x_original, y)
635                         pos[i + dim] = (x_copia, y)
636 
637                     y_center = sum(ys) / len(ys)
638                     pos[new_dim] = (x_z, y_center)
639 
640                 elif "-cycle" in filename:
641                     pos = {}
642                     radius = 2.5
```

```
643                     offset = 1.0
644                     angle_step = math.pi / (dim - 1)  # semicirculo
645
646                     for i in range(1, dim + 1):
647                         angle = math.pi - (i - 1) * angle_step
648
649                         x = radius * math.cos(angle)
650                         y = radius * math.sin(angle)
651                         pos[i] = (x, y)
652
653                         # copias (u_i)
654                         r_copia = radius + offset
655                         x_c = r_copia * math.cos(angle)
656                         y_c = r_copia * math.sin(angle)
657
658                         # ajustar o primeiro e o ultimo u_i
                               ligeiramente para baixo
659                         u_i = i + dim
660                         if u_i == dim + 1 or u_i == 2 * dim:
661                             y_c -= 0.5  # deslocamento vertical
662
663                         pos[u_i] = (x_c, y_c)
664
665                     pos[new_dim] = (0, -1.5)
666
667
668                 elif "-clique" in filename:
669                     pos = {}
670                     radius = 2.5
671                     offset = 1.0
672                     angle_step = math.pi / (dim - 1)
673
674                     for i in range(1, dim + 1):
675                         angle = math.pi - (i - 1) * angle_step
676
677                         x = radius * math.cos(angle)
678                         y = radius * math.sin(angle)
679                         pos[i] = (x, y)
680
681                         # copias (u_i)
682                         r_copia = radius + offset
683                         x_c = r_copia * math.cos(angle)
```

```
684                         y_c = r_copia * math.sin(angle)
685
686                         u_i = i + dim
687                         if u_i == dim + 1 or u_i == 2 * dim:
688                             y_c -= -0.5  # ajuste vertical
689
690                         pos[u_i] = (x_c, y_c)
691
692                     pos[new_dim] = (0, -1.5)
693
694
695
696                 elif "-pan" in filename:
697                     pos = {}
698                     original_nodes = list(range(1, dim + 1))
699                     base_nodes = [v for v in original_nodes if v !=
                            max(original_nodes)]  # ciclo base
700                     extra_node = max(original_nodes)  # cabo (n+1)
701                     n_base = len(base_nodes)
702
703                     radius = 2.5
704                     offset = 1.0
705                     angle_step = math.pi / (n_base - 1)
706
707                     # posicionar ciclo base
708                     for idx, node in enumerate(base_nodes):
709                         angle = math.pi - idx * angle_step
710
711                         x = radius * math.cos(angle)
712                         y = radius * math.sin(angle)
713                         pos[node] = (x, y)
714
715                         x_c = (radius + offset) * math.cos(angle)
716                         y_c = (radius + offset) * math.sin(angle)
717
718                         u_i = node + dim
719                         if u_i == dim + 1 or u_i == 2 * dim - 1:
720                             y_c -= 0.5  # ajuste para as extremas
721
722
723                         pos[u_i] = (x_c, y_c)
724
```

```
725                     x1, y1 = pos[1]
726                     pos[extra_node] = (x1 - 1.5, y1)
727                     pos[extra_node + dim] = (x1 - 2.5, y1)
728                     pos[extra_node + 2 * dim] = (x1 - 3.5, y1)
729
730                     pos[new_dim] = (0, -1.5)
731
732
733                 elif "-house" in filename:
734                     pos = {}
735                     radius = 2.5
736                     offset = 1.0
737                     angle_step = math.pi / (dim - 1)
738
739                     for i in range(1, dim + 1):
740                         angle = math.pi - (i - 1) * angle_step
741
742                         x = radius * math.cos(angle)
743                         y = radius * math.sin(angle)
744                         pos[i] = (x, y)
745
746                         # copias u_i
747                         x_c = (radius + offset) * math.cos(angle)
748                         y_c = (radius + offset) * math.sin(angle)
749
750                         u_i = i + dim
751                         if u_i == dim + 1 or u_i == 2 * dim:
752                             y_c -= 0.8
753
754                         pos[u_i] = (x_c, y_c)
755
756                     pos[new_dim] = (0, -1.5)
757
758
759                 else:
760                     pos = nx.spring_layout(G, seed=42)
761
762                 fig, ax = plt.subplots(figsize=(5, max(dim * 0.8,
                        5)))
763                 nx.draw(
764                     G, pos, ax=ax, with_labels=True,
765                     node_color='skyblue', edge_color='gray',
```

```
766                     node_size=500, font_size=10
767                 )
768                 ax.set_axis_off()
769                 plt.tight_layout()
770                 image_file = output_filename.replace(".gra", ".png"
                        )
771                 plt.savefig(image_file, bbox_inches="tight",
                        pad_inches=0.3)
772                 plt.close()
```

TERMO DE RESPONSABILIDADE

O texto do Trabalho de Conclusão de Curso intitulado “Criação de um catálogo de grafos e soluções para problemas de dominação em grafos a partir de uma interface gráfica” é de minha inteira responsabilidade. Declaro que não há utilização indevida de texto, material fotográfico ou qualquer outro material pertencente a terceiros sem o devido referenciamento ou consentimento dos referidos autores.

João Monlevade, 14 de julho de 2025.

Documento assinado digitalmente
gov.br
ANDRE FERNANDES DO PRADO TESSARO
Data: 14/07/2025 17:30:54-0300
Verifique em https://validar.iti.gov.br

André Fernandes do Prado Tessaro