Universidade Estadual de Campinas
Instituto de Computação

Felipe de Carvalho Pereira

Um Estudo Computacional do Problema do Gnosticismo Perfeito

CAMPINAS
2021

Felipe de Carvalho Pereira

Um Estudo Computacional do Problema do Gnosticismo Perfeito

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Pedro Jussieu de Rezende
Coorientador: Prof. Dr. Cid Carvalho de Souza

Este exemplar corresponde à versão final da Dissertação defendida por Felipe de Carvalho Pereira e orientada pelo Prof. Dr. Pedro Jussieu de Rezende.

CAMPINAS
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Pereira, Felipe de Carvalho, 1997-
P414e Um estudo computacional do Problema do Gnosticismo Perfeito / Felipe de Carvalho Pereira. – Campinas, SP : [s.n.], 2021.

Orientador: Pedro Jussieu de Rezende.
Coorientador: Cid Carvalho de Souza.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Disseminação de informação. 2. Redes sociais. 3. Otimização combinatória. 4. GRASP (Meta-heurística). 5. Programação linear inteira. I. Rezende, Pedro Jussieu de, 1955-. II. Souza, Cid Carvalho de, 1963-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: A computational study of the Perfect Awareness Problem
Palavras-chave em inglês:
Information dissemination
Social networks
Combinatorial optimization
GRASP (Metaheuristic)
Integer linear programming
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Pedro Jussieu de Rezende [Orientador]
Claudio Nogueira de Meneses
Fábio Luiz Usberti
Data de defesa: 22-03-2021
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-8967-8576
- Currículo Lattes do autor: http://lattes.cnpq.br/3697972262724998

Universidade Estadual de Campinas
Instituto de Computação

Felipe de Carvalho Pereira

Um Estudo Computacional do Problema do Gnosticismo Perfeito

Banca Examinadora:

- Prof. Dr. Pedro Jussieu de Rezende
  Instituto de Computação - Unicamp
- Prof. Dr. Claudio Nogueira de Meneses
  Centro de Matemática, Computação e Cognição - UFABC
- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação - Unicamp

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 22 de março de 2021

Dedicatória

Eu dedico este trabalho ao meu amado e eterno tio Ubirajara (*in memoriam)*.

"O caminho que eu sigo é o mais lodento e o mais feliz."
Júlio Andrade

Agradecimentos

Primeiramente, gostaria de agradecer aos meus incríveis pais, Ubijacira e Marcos, que nunca mediram esforços para me oferecer boas condições de estudo e de vida. Eu amo vocês incondicionalmente e serei eternamente grato por tudo que fizeram e que fazem por mim. Saibam que tudo que conquistei foi para vocês e por causa de vocês.

Em seguida, agradeço a todos meus familiares, em especial aos meus avós maternos Wilson e Maria, e meus avós paternos João (*in memoriam*) e Maria, pelo amor, apoio e incentivo que sempre recebi. Agradeço também à minha companheira, Camila, que tem sido meu braço direito há tanto tempo que já não consigo contar. Eu te amo e te admiro por ser uma pessoa tão extraordinária.

Aos meus mentores Prof. Rezende e Prof. Cid, expresso o meu mais sincero agradecimento por todo o conhecimento que compartilharam comigo, pelo zelo e paciência e pela confiança que em mim depositaram. Também agradeço ao Prof. Usberti por suas contribuições e solicitude e ao Prof. Breno Piva por ter me conduzido nos primeiros passos da minha trajetória acadêmica.

Agradeço aos meus queridos Jônatas, Letícya, Alisson e Hugo pela amizade, companheirismo e por nossos episódios de diversão; ao senhor Natanael por ser tão prestativo e compadecido para com a minha pessoa; e aos meus colegas do `LOCo`, Elisa, Ana, Renato, Fred, Deyvisson, Alonso, Enoque, Ota, João, Vitor, Welverton e Olímpio pela parceria, ajuda e pelos bons momentos.

Agradeço aos servidores e demais professores do Instituto de Computação da Unicamp, que fazem do `IC` uma instituição acadêmica de excelência.

Finalmente, agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - `CNPq` (processo 130838/2019-5) e à Fundação de Amparo à Pesquisa do Estado de São Paulo - `FAPESP` (processo 2019/22297-1) pelo apoio financeiro que tornou possível a realização dessa pesquisa.

Resumo

O Problema do Gnosticismo Perfeito (PAP do inglês *Perfect Awareness Problem*) é um problema de otimização combinatória inserido no tópico de *marketing* viral que envolve a propagação de informações em redes sociais [21]. O problema pode ser descrito da seguinte forma. A entrada consiste em um par (G, t), onde $G = (V, E)$ é um grafo não direcionado e $t : V \to \mathbb{N}^*$ é uma *função limiar*. O conjunto de vértices V corresponde à coleção de indivíduos de uma rede social e o conjunto de arestas E representa as comunicações possíveis entre tais indivíduos.

Supõe-se que, inicialmente, todos os indivíduos da rede são *ignorantes* de uma determinada informação, exceto por um conjunto inicial de *disseminadores*, denominado *conjunto semente*. Se um vértice ignorante v é informado por um vizinho disseminador, então v se torna *conhecedor*. Além disso, assim que v é informado por uma quantidade de vizinhos disseminadores igual ou maior que $t(v)$, v passa a ser também um disseminador. O objetivo do problema é determinar um conjunto semente de tamanho mínimo, a partir do qual uma propagação da informação faz com que todos os membros da rede sejam conhecedores dela.

O PAP é um problema NP-difícil e até agora somente um método heurístico foi relatado na literatura para lidar com ele. Neste trabalho, apresentamos quatro novas heurísticas para o PAP baseadas na meta-heurística GRASP [49]. Além desses algoritmos, desenvolvemos as duas primeiras formulações de Programação Linear Inteira para o problema, com o objetivo de obtermos soluções ótimas para instâncias do PAP e podermos comparar com elas as soluções produzidas por nossas heurísticas.

As contribuições da pesquisa também incluem três abordagens de pré-processamento de instâncias para redução do tamanho das entradas e um novo *benchmark* de 840 instâncias do PAP que simulam redes sociais. Foi realizado ainda um conjunto extenso de experimentos comparativos, seguidos de análises estatísticas, mostrando a eficácia e eficiência das heurísticas propostas. Além disso, aplicamos a melhor de nossas heurísticas a 17 instâncias da literatura que representam redes sociais reais e verificamos que nosso algoritmo supera a única heurística previamente existente para o PAP.

Abstract

The Perfect Awareness Problem (PAP) is a combinatorial optimization problem within the area of viral marketing and related to the spread of information in social networks [21]. The problem can be described as follows. The input consists of a pair (G, t) where $G = (V, E)$ is an undirected graph and $t : V \to \mathbb{N}^*$ is a *threshold function*. The set of vertices V corresponds to the collection of individuals in a social network and the edges in E indicate pairs of individuals between whom communication is possible.

It is assumed that, initially, all individuals in the network are *ignorant* of a certain information, except for an initial set of *spreaders*, called *seed set*. If an ignorant vertex v is informed by a neighboring spreader, then v becomes *aware*. Moreover, as soon as v is informed by a number of neighboring spreaders greater than or equal to $t(v)$, v also becomes a spreader. The objective of the problem is to find a seed set of minimum size, from which the propagation of the information makes all members of the network aware.

The PAP is NP-hard and, so far, only one heuristic has been reported in the literature for this problem. In this work, we present four novel heuristics for the PAP, based on the metaheuristic GRASP [49]. In addition to these algorithms, we developed the first two Integer Programming formulations for the problem, in order to obtain optimal solutions for PAP instances, so as to compare them with the solutions produced by our heuristics.

The contributions of this research also include three preprocessing approaches for reducing the size of input instances and a new benchmark of 840 instances of the PAP that simulate social networks. An extensive set of comparative experiments was conducted, followed by statistical analyses, showing the effectiveness and efficiency of the proposed heuristics. Furthermore, we applied the best one of our heuristics to 17 instances from the literature that represent real social networks and found that our algorithm surpasses the only previously reported heuristic for the PAP.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Estudos de redes sociais têm sido realizados por cientistas sociais desde o século passado. Até pouco tempo, pesquisadores faziam uso de conjuntos de dados pequenos devido às limitações tecnológicas. Entretanto, o surgimento da Internet proporcionou o recente advento das redes sociais *online* como, por exemplo, Facebook, Instagram e Twitter, entre outras. Essas redes contribuíram para a eclosão da disponibilidade de grandes quantidades de dados e, consequentemente, para um crescimento do interesse por problemas que envolvem redes sociais [14].

Uma parte significativa das pesquisas desenvolvidas nessas redes *online* pertence à área de estudo denominada análise de propagação de informação e influência em redes sociais. Nessa área, existem diversos tópicos de investigação tais como: análise de mídia social, estudo de epidemias, publicidade direcionada, etc. [14]. Um tópico de grande relevância que abrange diversas pesquisas na computação é o chamado *marketing* viral.

Considere que um usuário do Facebook fez um *post* sobre uma nova música de seu artista preferido. Quando um amigo desse usuário comenta ou compartilha o *post*, a informação é passada para membros de um segundo nível de relacionamento e assim sucessivamente, com potencial de transitar por uma fração da rede. Agora, suponha que o artista deseja que sua música seja difundida até muitos usuários do Facebook. Um problema imediato consiste em determinar "bons" primeiros usuários que devem postar sobre aquela música de modo que o objetivo seja alcançado.

Uma vez que o processo de disseminação de uma informação esteja modelado, é possível formular problemas que visam otimizar o alcance da sua propagação através da rede. Esse é, essencialmente, o objetivo do *marketing* viral: "ativar" um número idealmente pequeno de indivíduos influenciadores de uma rede social de modo que um grande número de outros indivíduos sejam influenciados por uma propagação viral [14].

Nesse contexto, temos o Problema do Gnosticismo Perfeito (PAP do inglês *Perfect Awareness Problem*), um problema de otimização combinatória NP-difícil dentro do tema de *marketing* viral. No PAP, supõe-se que inicialmente todos os indivíduos da rede são *ignorantes* de uma determinada informação, exceto por um conjunto inicial de *disseminadores*. Quando um disseminador informa um indivíduo ignorante v, este passa a ser um *conhecedor*. Além disso, assim que v é informado por uma quantidade de disseminadores maior ou igual a um determinado *valor limiar*, v passa também a ser um disseminador.

O objetivo do problema é determinar um conjunto inicial de disseminadores de tama-

nho mínimo de maneira que ao fim da propagação toda a rede seja conhecedora [21]. Na prática, resolver o PAP consiste em minimizar o custo de "recrutamento" dos disseminadores iniciais da informação a ser propagada, garantindo que, em algum momento, todos os indivíduos da rede tomem conhecimento da informação.

Estudos experimentais que investigam a maneira como ocorrem as disseminações em redes sociais indicam que a informação não flui livremente na rede. Ou seja, a transmissão da informação requer que exista um compartilhamento ativo que, por sua vez, depende das convicções individuais dos membros da rede [21]. Além disso, é sabido que os indivíduos que são mais expostos à informação são significativamente mais propensos a divulgá-la e o fazem mais cedo do que aqueles que não são tão expostos [5]. Essas são as principais observações que fundamentam o modelo de propagação previsto para o PAP.

Esse modelo também pode ser visto como uma idealização do processo de difusão de memes. Um meme é uma unidade de informação cultural, como por exemplo um padrão cognitivo ou comportamental, que pode ser transmitida de um indivíduo para outro [24]. Experimentos sobre como os memes evoluem e se espalham no Facebook [2] sugerem que o modelo de difusão do PAP está em conformidade com o mecanismo de disseminação de memes: um indivíduo adquire um meme quando ouve um amigo falar dele, mas as pessoas começam a espalhar um meme apenas quando este parece ser popular ou importante, ou seja, quando foi ouvido de vários amigos.

Este trabalho de Dissertação consiste em um estudo computacional do PAP com o objetivo principal de produzir algoritmos heurísticos eficazes e eficientes para o problema. Nossas principais contribuições podem ser sumarizadas como segue:

- Projetamos quatro heurísticas baseadas na meta-heurística Greedy Randomized Adaptive Procedure (GRASP) [26];
- Desenvolvemos três técnicas de pré-processamento de instâncias que reduzem o tamanho das entradas;
- A fim de avaliar nossos algoritmos, produzimos um *benchmark* de 840 instâncias do PAP com características de redes sociais;
- Com o objetivo de obter os valores ótimos das instâncias geradas e compará-los com os valores de soluções produzidas pelas heurísticas, projetamos as duas primeiras formulações conhecidas de programação linear inteira para o PAP;
- Realizamos uma análise estatística completa das nossas heurísticas e as classificamos de acordo com seu sucesso na resolução de todas as instâncias de *benchmark*;
- Aplicamos a nossa melhor heurística a 17 instâncias da literatura que representam redes sociais reais e verificamos que, para essas instâncias, nosso melhor algoritmo supera a única heurística previamente existente para o PAP.

O texto deste documento está estruturado da seguinte forma. O Capítulo 2 apresenta conceitos, definições e notações fundamentais para a compreensão deste trabalho. Em seguida, o Capítulo 3 traz a definição formal do PAP e uma revisão bibliográfica do problema,

além de uma exposição acerca de outros problemas de otimização relacionados ao PAP. No Capítulo 4, abordamos os métodos empregados na pesquisa para resolver o PAP. Adiante, no Capítulo 5, apresentamos o projeto dos experimentos computacionais conduzidos com as heurísticas e formulações, assim como os resultados provenientes desses experimentos. Por fim, o Capítulo 6 traz as conclusões obtidas neste trabalho e perspectivas futuras.

Capítulo 2

Conceitos Básicos

Neste capítulo, introduzimos conceitos, definições e notações essenciais para a compreensão desse trabalho de dissertação. Na Seção 2.1, apresentamos noções básicas sobre otimização combinatória. Em seguida, na Seção 2.2, introduzimos a meta-heurística `GRASP`. Por fim, a Seção 2.3 trata da técnica de modelagem matemática e otimização linear denominada Programação Linear Inteira (`PLI`).

Além do conteúdo apresentado neste capítulo, o leitor deve estar familiarizado com tópicos fundamentais da Teoria da Computação, tais como projeto e análise de algoritmos, terminologia de grafos, algoritmos básicos em grafos e complexidade computacional de problemas. Estes tópicos estão cobertos pela literatura em [10, 22, 37, 40, 51].

2.1 Otimização Combinatória

A otimização combinatória é um ramo da teoria da computação que trata de problemas que envolvem a busca por um objeto ótimo em uma coleção finita de objetos. Tipicamente, o tamanho da coleção é pelo menos exponencial em relação ao tamanho da representação computacional desses objetos. Assim, ao buscarmos pelo melhor elemento da coleção, uma verificação um a um destes elementos não é uma abordagem prática e, portanto, algoritmos mais eficientes são estudados na literatura [52].

Formalmente, muitos problemas de otimização combinatória, incluindo o `PAP`, podem ser definidos como segue. Sejam n um inteiro positivo, $E = \{1, \ldots, n\}$ um conjunto finito, $F \subseteq 2^E$ um conjunto de *soluções viáveis* (também chamadas factíveis) e $f : 2^E \to \mathbb{R}$ uma *função objetivo* (ou função de custo). Se o problema é de minimização[1], então buscamos uma solução $S^* \in F$ tal que $\forall S \in F\, f(S^*) \leq f(S)$. Denominamos S^* uma *solução ótima* e dizemos que $f(S^*)$ é o *valor ótimo*. A tripla (E, F, f) é definida de forma específica para cada problema [49].

Para uma instância do `PAP`, E representa os n indivíduos da rede e F contém todos os possíveis conjuntos de disseminadores iniciais a partir dos quais uma propagação atinge toda a rede. Além disso, $f(S)$ calcula o tamanho de um conjunto de disseminadores iniciais S.

[1] Alternativamente, se o problema é de maximização, buscamos uma solução $S^* \in F$ tal que $\forall S \in F$, $f(S^*) \geq f(S)$.

Os três principais tipos de algoritmos utilizados para solução de problemas de otimização são descritos a seguir.

- Algoritmo exato: encontra uma solução ótima para qualquer instância do problema;
- Algoritmo heurístico: retorna uma solução viável sem garantia comprovada de qualidade da solução, mas que, em geral, é rápido mesmo para instâncias grandes e, na prática, frequentemente devolve boas soluções viáveis;
- Algoritmo de aproximação: retorna uma solução viável cujo valor é pelo menos tão bom quanto um fator α do valor ótimo. Neste caso, dizemos que o algoritmo é α-aproximado [56].

Um algoritmo é dito *eficiente* para resolver um problema de otimização, se o seu tempo de execução é limitado superiormente por algum polinômio em função do tamanho da entrada [52]. Os problemas de otimização para os quais existem algoritmos exatos e eficientes constituem a classe P.

Ademais, sabe-se que a classe NP contém grande parte dos problemas de otimização e que todos os problemas em NP podem ser reduzidos em tempo polinomial para qualquer problema em NP-difícil. A interseção de NP e NP-difícil é a classe denominada NP-completo. Além disso, é sabido que inúmeros problemas de otimização pertencem a P ou a NP-completo. Uma das questões em aberto mais importantes da computação é se $\mathsf{P} = \mathsf{NP}$ ou $\mathsf{P} \neq \mathsf{NP}$ [52].

A menos que $\mathsf{P} = \mathsf{NP}$, não existem algoritmos exatos e eficientes para problemas de otimização da classe NP-difícil. Na prática, para estes problemas, algoritmos exatos resolvem apenas instâncias pequenas, uma vez que, nesses casos, o tempo de execução de algoritmos exatos geralmente cresce exponencialmente em relação ao tamanho da entrada. Em contraposição, heurísticas e algoritmos de aproximação mostram-se computacionalmente rápidos em solucionar tais problemas mesmo para instâncias grandes, mas com a deficiência de não garantirem a otimalidade das soluções viáveis obtidas.

O PAP é um problema provadamente NP-difícil [21], isto é, trata-se de um problema pelo menos tão difícil quanto os demais problemas de NP. Por essa razão, buscamos atacar o PAP com heurísticas, mais especificamente com algoritmos baseados na meta-heurística GRASP. Além disso, utilizamos programação linear inteira para resolver o PAP por meio de algoritmos exatos, a fim de verificar a qualidade das soluções produzidas pelas nossas heurísticas para instâncias pequenas.

2.2 Meta-heurística GRASP

Meta-heurísticas são paradigmas genéricos de projeto de algoritmos (ou *frameworks*), a partir dos quais uma ou mais heurísticas podem ser projetadas para diversos problemas. Nas últimas décadas, várias meta-heurísticas mostraram-se alternativas bem sucedidas para solução de problemas de otimização difíceis [7].

O GRASP é uma meta-heurística iterativa, inicialmente proposta em [26]. Trata-se de um dos mais conhecidos métodos do tipo *multi-start*, ou seja, que contêm rotinas

que são inicializadas múltiplas vezes com o objetivo de alcançar diversidade nas soluções obtidas [29]. Desde a sua concepção, essa meta-heurística tem sido aplicada de maneira extensiva para os mais variados problemas de otimização. Em [27] é apresentada uma bibliografia anotada da literatura sobre o `GRASP`.

No `GRASP`, cada iteração é dividida em duas fases distintas: construção e busca local. Na fase de construção, uma solução viável é construída de maneira incremental. A cada passo, um elemento é escolhido por meio de um critério guloso e randomizado e, em seguida, é adicionado à solução.

Uma vez que a solução torna-se viável, inicia-se a fase de busca local, na qual uma região do espaço de soluções é investigada até que, em geral, uma solução *localmente ótima* seja encontrada. Dizemos que uma solução S é localmente ótima, se S possui o melhor valor entre as soluções que permeiam uma região do espaço de busca de soluções que contém S. Nesse caso, o valor de S é chamado de *ótimo local*. A melhor solução obtida entre as iterações é mantida como solução final.

O Algoritmo 2.1 apresenta o principal laço do `GRASP`. A condição de parada pode ser baseada em um número fixo de iterações ou em um limite superior no tempo de execução.

Algoritmo 2.1: Meta-heurística `GRASP` (adaptado de [49]).

Entrada: Instância I
Saída : Solução viável S

```
1 S ← ∅
2 enquanto condição de parada não for satisfeita faça
3 |   S' ← FaseDeConstrução(I)
4 |   S' ← FaseDeBuscaLocal(I, S')
5 |   se (S = ∅) ∨ (f(S') é melhor que f(S)) então
6 |   |   S ← S'
7 retorna S
```

Na fase de construção, começamos com um conjunto S vazio como solução inicial. Além disso, mantemos uma lista de candidatos, indicada pelo conjunto CL, que contém todos os elementos que não estão em S e que podem ser inseridos em S. No início de cada passo dessa fase, calculamos o benefício de se inserir cada elemento da CL em S.

Além da CL, temos uma lista restrita de candidatos, representada pelo conjunto RCL, que contém os elementos da CL que possuem os melhores benefícios. Em cada passo da construção, um elemento da RCL é escolhido para incrementar S, seguindo algum critério aleatorizado. O procedimento é repetido até que S se torne viável. O Algoritmo 2.2 apresenta as etapas da fase de construção do `GRASP`.

Uma vez que a solução S construída na primeira fase pode não ser localmente ótima, aplicamos a fase de busca local a partir de S a fim de encontrar soluções viáveis melhores que S, que estão localizadas nas proximidades de S no espaço de busca de soluções. Para esse fim, é necessário definir, para o problema a ser tratado, como representar o conjunto de soluções vizinhas de uma dada solução.

Dizemos que S' é uma solução vizinha de S, se S' é fruto de operações que modificam S. Em geral, essas operações consistem na adição, eliminação ou substituição de um ou

Algoritmo 2.2: Fase de construção do `GRASP` (adaptado de [49]).

Entrada: Instância I
Saída : Solução viável S

```
1 S ← ∅
2 CL ← ConstróiCL(I)
3 enquanto S não é viável faça
4 |   CalculaBenefícios(CL)
5 |   RCL ← ConstróiRCL(CL)
6 |   e ← SelecionaElementoAleatório(RCL)
7 |   S ← S ∪ {e}
8 |   AtualizaCL(CL)   ▷ Remove os elementos que não podem ser inseridos em S
9 retorna S
```

mais elementos em S. Baseando-se nesse conceito, temos as duas principais estratégias de busca local, chamadas de *best* e *first improvement* [49].

No *best improvement*, varremos a vizinhança da solução corrente S e determinamos o melhor vizinho S' de S. Se o valor de S' é melhor que o valor de S, S' passa a ser a solução corrente para a próxima iteração. Já na estratégia *first improvement*, interrompemos a varredura sobre os vizinhos de S assim que encontramos qualquer vizinho S' de S que seja melhor do que este e, nesse caso, S' passa a ser a solução corrente na próxima iteração.

Em ambos os casos, o critério de parada para as iterações da busca local é normalmente definido como a obtenção de um ótimo local, ou seja, quando não são encontrados vizinhos melhores do que a solução corrente. O Algoritmo 2.3 exibe os passos de uma busca local típica.

Algoritmo 2.3: Fase de busca local do `GRASP` (adaptado de [49]).

Entrada: Instância I; solução viável S
Saída : Solução viável S localmente ótima

```
1  faça
2  |   S'' ← ∅   ▷ S'' conterá o melhor vizinho de S tal que f(S'') é melhor que f(S)
3  |   b ← f(S)                                          ▷ Melhor valor de solução
4  |   para cada S' na vizinhança de S faça
5  |   |   se f(S') é melhor que b então
6  |   |   |   S'' ← S'                                  ▷ Atualiza melhor vizinho de S
7  |   |   |   b ← f(S')                                 ▷ Atualiza melhor valor de solução
8  |   |   |   se estratégia first improvement se aplica então
9  |   |   |   |   Encerre o laço mais interno
10 |   se S'' ≠ ∅ então
11 |   |   S ← S''
12 enquanto S'' ≠ ∅        ▷ Se S'' = ∅, então não há vizinho de S melhor do que S
13 retorna S
```

2.3 Programação Linear Inteira

Nesta seção, apresentamos os conceitos fundamentais de Programação Linear Inteira (PLI), seguindo a abordagem de [57, 58]. Um *programa linear*, ou *problema de programação linear*, é composto por uma função linear que deve ser minimizada (ou maximizada) e por restrições lineares na forma de desigualdades matemáticas com variáveis não negativas. Muitos problemas matemáticos e computacionais, especialmente em otimização combinatória, podem ser modelados por meio de programação linear.

Suponha que temos o seguinte programa linear:

$$\begin{aligned} \min\, & cx \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

onde A é uma matriz $m \times n$, c é um vetor linha $1 \times n$, b é um vetor coluna $m \times 1$ e x é um vetor coluna $n \times 1$ contendo variáveis reais. Os elementos de A, b e c são números racionais. Assim, o problema visa minimizar a função objetivo cx, sujeita às desigualdades $Ax \leq b$. Observe que c e A contêm os coeficientes da função linear e das restrições, respectivamente.

Diferentes tipos de variáveis determinam tipos de programas lineares distintos. Se todas as variáveis são inteiras, temos um programa inteiro da forma:

$$\begin{aligned} \min\, & cx \\ & Ax \leq b \\ & x \geq 0 \text{ e inteiro.} \end{aligned}$$

Porém, se algumas variáveis são inteiras, mas não todas, temos um programa inteiro misto da forma:

$$\begin{aligned} \min\, & cx + dy \\ & Ax + By \leq b \\ & x \geq 0 \\ & y \geq 0 \text{ e inteiro,} \end{aligned}$$

onde B é uma matriz $m \times l$, d é um vetor linha $1 \times l$ e y é um vetor coluna $l \times 1$ contendo variáveis inteiras. Os elementos de B e d são números racionais. Por fim, se todas as variáveis estão restritas aos valores 0 e 1, temos um programa inteiro binário da forma:

$$\begin{aligned} \min\, & cx \\ & Ax \leq b \\ & x \geq 0 \text{ e binário.} \end{aligned}$$

Ao modelarmos um problema de otimização combinatória por intermédio de programação linear, dizemos que o programa resultante é uma *formulação* (ou *modelo*) de programação linear para aquele problema. Os seguintes passos são fundamentais para a

construção de uma boa formulação:

(i) Identificar no problema as variáveis necessárias;

(ii) Utilizar as variáveis de (i) para criar um conjunto de desigualdades lineares, de modo que os pontos no espaço de soluções que as satisfazem correspondam a soluções viáveis do problema;

(iii) Definir uma função objetivo linear nas variáveis de (i).

Programas lineares (não inteiros) podem ser resolvidos em tempo polinomial pelo método dos elipsoides [36] ou pelo algoritmo de pontos interiores [34]. Ademais, embora o algoritmo mais popular na literatura para esse propósito, o Simplex [23], seja exponencial no pior caso, ele tem um ótimo desempenho na prática.

Por outro lado, é sabido que a programação inteira é NP-difícil [17] e, por isso, a menos que P = NP, algoritmos capazes de resolver programação inteira em tempo polinomial não são esperados surgir.

Dentre os principais algoritmos existentes na literatura que tratam problemas de PLI, destacamos as técnicas de *branch and bound* e *branch and cut*. Ambas são baseadas na realização de busca em uma árvore enraizada que enumera as soluções do problema. Essa árvore é chamada de *árvore de enumeração*.

Cada folha da árvore de enumeração corresponde a uma solução (viável ou não) do problema. As arestas representam restrições sobre os valores das variáveis. Assim, se u é a raiz da árvore de enumeração de um problema Q e v é filho de u, então v é a raiz da árvore de enumeração de um subproblema Q' obtido de Q mediante a aplicação da restrição correspondente à aresta (v, u). Vértices filhos de um mesmo vértice pai representam subproblemas complementares do problema correspondente ao vértice pai. A Figura 2.1 mostra um exemplo de árvore de enumeração.

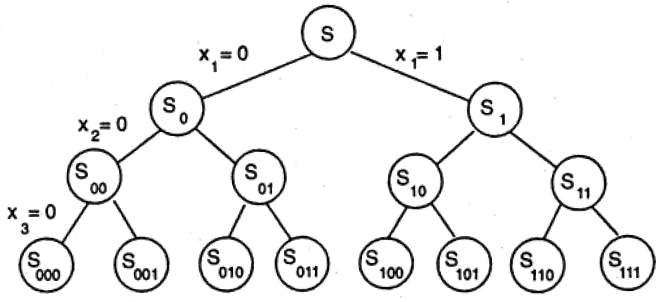


Figura 2.1: Exemplo de árvore de enumeração de um programa inteiro binário com variáveis x_1, x_2 e x_3 (extraído de [57]).

Durante a busca na árvore de enumeração, os algoritmos realizam eliminação de partes substanciais da árvore, também chamadas de *podas*. Para esse fim, usam-se os conceitos de *limitantes inferior* e *superior* do valor de uma solução ótima.

Seja S_P^* uma solução ótima para um programa inteiro P, onde $f(S_P^*)$ denota o valor de S_P^*, isto é, o valor ótimo para P. Se S_P^* e $f(S_P^*)$ são desconhecidos, muitas vezes é possível

determinar um intervalo fechado no qual $f(S_P^*)$ se encontra. Sem perda de generalidade, suponha que P é um problema de minimização. Por exemplo, qualquer solução viável S_P para P provê um limitante superior para $f(S_P^*)$, isto é, $f(S_P^*) \leq f(S_P)$. Em PLI, chamamos $f(S)$ de *limitante primal* (L_P). Bons limitantes primais podem ser obtidos por meio de heurísticas.

Para obter um limitante inferior para $f(S_P^*)$, podemos aplicar técnicas de relaxação de problemas que consistem em redefinir o problema original, removendo algumas de suas restrições o que o torna mais fácil de ser resolvido. Em seguida, resolve-se o problema relaxado de maneira exata, obtendo-se um valor que é um limitante inferior para o valor ótimo do problema original.

Uma das técnicas de relaxação mais úteis é a chamada *relaxação linear*. Nessa abordagem, removemos as restrições de integralidade das variáveis de P, obtendo um programa linear P'. Em seguida, resolvemos P' em tempo polinomial e obtemos uma solução $S_{P'}^*$ que é ótima para P'. Nesse caso, $f(S_{P'}^*) \leq f(S_P^*)$ e, dessa forma, $f(S_{P'}^*)$ é um limitante inferior de $f(S_P^*)$. Em PLI, chamamos $f(S_{P'}^*)$ de *limitante dual* (L_D).

Algoritmos de PLI se empenham em obter melhores limitantes de maneira progressiva até que L_P e L_D convirjam para o valor ótimo, isto é, até que tenhamos[2] $L_P = L_D = f(S_P^*)$.

Nos algoritmos do tipo *branch and bound*, uma subárvore da árvore de enumeração é podada sempre que a raiz dessa subárvore corresponde a um subproblema inviável, resolvido na otimalidade ou ainda se o limitante dual obtido para aquele subproblema é maior do que o valor do melhor limitante primal obtido para o problema original, considerando um problema de minimização.

Os algoritmos do tipo *branch and cut* estendem a técnica de *branch and bound*, acrescentando mecanismos para obter melhores limitantes duais para os subproblemas. Isso é feito por meio da descoberta de desigualdades válidas para os subproblemas que são violadas por soluções da relaxação. Essas desigualdades são chamadas de *planos de corte*. Ao adicionarmos tais restrições à formulação do subproblema, resolvemos novamente o subproblema relaxado, obtendo um limitante dual geralmente melhor do que o anterior.

Existem softwares comerciais voltados para a resolução de programas inteiros, sendo o CPLEX [32] um dos mais utilizados atualmente na literatura de PLI.

No próximo capítulo, apresentamos um revisão da literatura acerca de problemas que envolvem a propagação de informações em redes sociais, com ênfase no PAP e em problemas relacionados.

[2]Na verdade, é suficiente obter-se $L_P = \lceil L_D \rceil$.

Capítulo 3

Revisão Bibliográfica

Neste capítulo, apresentamos uma revisão bibliográfica sobre problemas de propagação em redes sociais relacionados ao PAP. Na Seção 3.1, introduzimos os dois principais modelos de propagação de informações existentes na literatura. Na Seção 3.2, exibimos uma definição formal do PAP. A Seção 3.3 descreve os principais resultados encontrados na literatura para o problema, com ênfase nos teóricos e nos algoritmos conhecidos para o mesmo. Na Seção 3.4, apresentamos um conjunto de problemas relativos ao PAP.

3.1 Modelos de Propagação de Informações

Nesta seção, descrevemos como que processos de propagação de informações são formalmente modelados na literatura de acordo com [14].

Representamos uma rede social como um grafo $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas que conectam pares de vértices de V. Cada indivíduo da rede corresponde a um vértice[3] $v \in V$, de modo que cada aresta $\{u, v\} \in E$ reflete um relacionamento de influência entre os indivíduos u e v. Nos casos em que se pretende representar influências unidirecionais, utilizam-se arestas direcionadas (*arcos*).

Em geral, a passagem de tempo em processos de difusão de informação é discretizada no que denominamos *rodadas*. Em cada rodada $\tau \in \mathbb{N}$, dizemos que $v \in V$ está *ativo* ou *inativo*. Se v adotou a informação que está sendo propagada e a passa adiante, então v está ativo. Caso contrário, v está inativo.

Diferentes problemas podem utilizar modelos de propagação diversos, os quais se distinguem principalmente pela forma com que indivíduos são influenciados. De acordo com [14], os dois principais modelos existentes na literatura são chamados de *independent cascade* (IC) e *linear threshold* (LT). Aqui, apresentamos as definições dos modelos IC e LT para um grafo G não direcionado. Todavia, tais definições podem ser estendidas para grafos direcionados.

No modelo IC [35], a cada aresta $\{u, v\} \in E$ é associada uma probabilidade de influência $p_{u,v} \in [0, 1]$ que indica a chance de u influenciar v e vice-versa. Assim, se u torna-se ativo em alguma rodada $\tau \geq 0$ e v é inativo nesta mesma rodada, então v se tornará ativo na rodada $\tau + 1$ por influência de u com probabilidade $p_{u,v}$. Note que a rodada τ deve ser

[3] Ao longo do texto, identificaremos um vértice v com o indivíduo ao qual ele corresponde.

o único momento em que u pode influenciar v.

No modelo LT [35], a cada aresta $\{u, v\} \in E$ é associado um peso positivo que quantifica a capacidade de u influenciar v e vice-versa. Além disso, cada vértice $v \in V$ é associado a um *valor limiar* $t(v)$ de acordo com uma *função limiar* $t : V \to \mathbb{N}^*$. Dessa forma, suponha que v está inativo em uma rodada $\tau \geq 0$ e que existem z vizinhos de v ativos nesta mesma rodada. Se as somas dos pesos das arestas que conectam v a esses z vizinhos é maior ou igual a $t(v)$, então v se tornará ativo na rodada $\tau + 1$. Como veremos na seção seguinte, o PAP é definido usando um modelo de propagação baseado no LT.

3.2 O Problema do Gnosticismo Perfeito

Inicialmente proposto em [21], o PAP possui como entrada um par (G, t), onde $G = (V, E)$ é um grafo não direcionado e $t : V \to \mathbb{N}^*$ é uma função limiar. O conjunto de vértices V corresponde à coleção de indivíduos de uma rede social e o conjunto de arestas E representa as comunicações entre os indivíduos da rede. Ou seja, se u e v pertencem a V e $\{u, v\} \in E$, então u e v podem se comunicar diretamente na rede, e portanto, se influenciam mutuamente. Denotamos por $N_G(v)$ a vizinhança[4] de v em G, i.e., $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$.

Neste problema, a passagem de tempo é descrita por rodadas, denotadas por $\tau \in \mathbb{N}$. Em toda rodada τ, cada vértice v assume ao menos um estado, entre os três descritos a seguir, em relação à informação que está sendo propagada:

- Ignorante: se v não recebeu a informação a partir de quaisquer vizinhos e, por conseguinte, desconhece a informação;
- Conhecedor: se v recebeu a informação por intermédio de ao menos um de seus vizinhos e, em consequência, já possui conhecimento da informação;
- Disseminador: se v recebeu a informação por meio de ao menos $t(v)$ vizinhos e, neste caso, já dissemina a informação.

Note que um vértice não pode ser ignorante e conhecedor ao mesmo tempo. Por outro lado, todo vértice disseminador também é conhecedor, mas a recíproca é falsa.

O processo de disseminação começa na rodada $\tau = 0$ a partir de um conjunto inicial não-vazio de disseminadores $S \subseteq V$, chamado de *conjunto semente*. Os vértices do conjunto semente são chamados de *sementes*. Em uma propagação iniciada por S, denota-se por $\mathrm{Dis}_G[S, \tau]$ o conjunto de vértices que são disseminadores na rodada τ. Dessa maneira, $\mathrm{Dis}_G[S, \tau]$ é igual a:

- S, se $\tau = 0$;
- $\mathrm{Dis}_G[S, \tau - 1] \cup \{u : |N_G(u) \cap \mathrm{Dis}_G[S, \tau - 1]| \geq t(u)\}$, se $\tau \geq 1$.

[4] Ao longo do texto, omitiremos o subscrito quando o grafo ao qual v pertence estiver evidente no contexto.

Ou seja, na rodada $\tau = 0$, o conjunto de disseminadores é o próprio conjunto semente, e para $\tau \geq 1$, esse conjunto é obtido pela união dos vértices que eram disseminadores na rodada $\tau - 1$ com todo vértice u que possuía, na rodada $\tau - 1$, uma quantidade de vizinhos disseminadores maior ou igual a $t(u)$. Note que para todo $\tau \geq 1$ vale que $\mathrm{Dis}_G[S, \tau - 1] \subseteq \mathrm{Dis}_G[S, \tau]$.

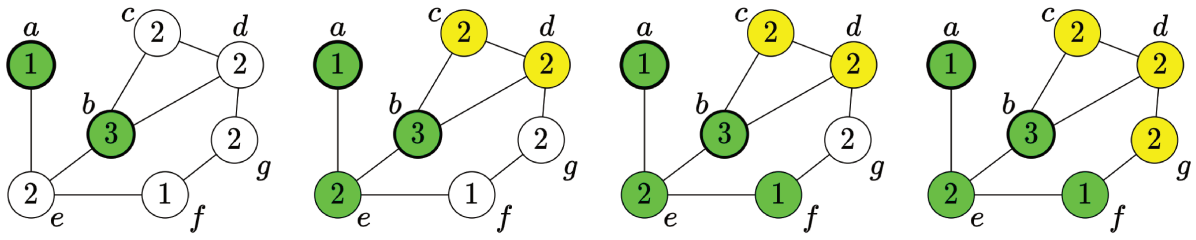
O término da propagação ocorre quando $\mathrm{Dis}_G[S, \rho] = \mathrm{Dis}_G[S, \rho - 1]$ para algum $\rho \geq 1$. Isto é, quando nenhum novo disseminador surge na transição entre rodadas consecutivas. Os conjuntos finais de disseminadores e conhecedores são denotados e descritos, respectivamente, por:

- $\mathrm{Dis}_G[S] = \mathrm{Dis}_G[S, \rho]$;
- $\mathrm{Con}_G[S] = S \cup \{u : |N_G(u) \cap \mathrm{Dis}_G[S]| \geq 1\}$.

Isto é, o conjunto final de vértices conhecedores é formado pelas sementes e por todo vértice u que tenha ao menos um vizinho disseminador na rodada ρ. Ademais, um conjunto semente S é chamado de *conjunto semente perfeito*, se $\mathrm{Con}_G[S] = V$, ou seja, se ao final da propagação todos os vértices em V são conhecedores.

Problema 1 (PAP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$ e uma função limiar $t : V \to \mathbb{N}^*$, o objetivo do* PAP *é encontrar um conjunto semente perfeito de tamanho mínimo.*

Observe que, para o PAP, uma solução é viável se, e somente se, ela é um conjunto semente perfeito. A Figura 3.1 exemplifica o processo de propagação em uma instância do PAP, onde os limiares associados aos vértices estão indicados nos círculos. Em cada rodada, os disseminadores estão representados na cor verde, os conhecedores em amarelo e os ignorantes na cor branca. Neste exemplo, o conjunto semente escolhido é formado pelos vértices a e b, e trata-se de uma solução ótima para essa instância do problema.



(a) $\tau = 0$ (b) $\tau = 1$ (c) $\tau = 2$ (d) $\tau = 3$

Figura 3.1: Exemplo de propagação do PAP.

3.3 Trabalhos Anteriores

Embora o PAP tenha sido introduzido apenas recentemente, já existem resultados teóricos e algoritmos propostos para esse problema. O primeiro resultado teórico mostra que o PAP não pode ser aproximado por um fator de $\mathcal{O}(2^{\log^{1-\varepsilon} n})$ para qualquer $\varepsilon > 0$, a menos que

$\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\text{polylog}(n)})$ [21]. Isso implica que o PAP é um problema NP-difícil. A prova é baseada em uma redução polinomial do Problema da Seleção do Conjunto de Alvos (TSSP do inglês *Target Set Selection Problem*) para o PAP, a fim de estender o resultado que fora originalmente provado para o TSSP em [13]. Em [21], é provado também que esse resultado é válido mesmo quando os limiares de todos os vértices são limitados superiormente pelo valor 2.

O Problema do Conjunto Dominante Mínimo (MDSP do inglês *Minimum Dominating Set Problem*) é um problema NP-difícil clássico [30] também sabido ser difícil de aproximar [31]. Sabe-se que o MDSP pode ser reduzido em tempo polinomial ao PAP, chegando-se a que o PAP é um problema NP-difícil por um outro caminho (vide Apêndice A). Essa redução envolve a transformação de uma instância do MDSP em uma instância do PAP tal que os valores limiares dos vértices são iguais aos seus respectivos graus.

Através da extensão de resultados existentes para o MDSP, em [21] prova-se que dado um grafo G tal que $t(v) = d(v)$ para qualquer vértice v, o PAP não pode ser aproximado por um fator de $(1 + o(1)) \ln \Delta$, a menos que $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\text{polylog}(n)})$, onde $d(v)$ denota o grau de v em G e Δ denota o maior grau em G.

Existem ainda resultados teóricos para o PAP relacionados a uma classe de grafos densos chamados *grafos de Ore*. Conforme [21], G é um grafo de Ore, se $d(u) + d(v) \geq |V|$ para todo $u, v \in V$, com $\{v, u\} \notin E$. Em [21], é provado que se G é um grafo de Ore, então existe um conjunto semente perfeito de tamanho no máximo:

$$\min \left\{ 3 + \left\lfloor \frac{\delta^2 - 5\delta + 6}{|V| - \delta} \right\rfloor, \left\lfloor \frac{|V|}{4} \right\rfloor + 1, \delta \right\},$$

onde δ denota o menor grau em G.

Dois algoritmos foram propostos em [21] para o PAP. O primeiro deles é o algoritmo que denotaremos por CGR-TREE, o qual devolve um conjunto semente perfeito de tamanho mínimo para instâncias do PAP nas quais o grafo dado na entrada é uma árvore. A seguir, descrevemos o funcionamento do CGR-TREE.

Seja $T = (V, E)$ uma árvore enraizada em algum vértice $r \in V$. O CGR-TREE computa um conjunto semente perfeito S de maneira progressiva ao visitar os vértices de T em ordem inversa àquela que seria feita através de uma busca em largura. Uma vez que um vértice v tenha sido visitado, o CGR-TREE determina se v deve ou não pertencer a S e se v deve ou não se tornar um disseminador. Em seguida, v é removido da árvore e o limiar de seu vértice pai é atualizado de acordo com a escolha feita para v.

O Algoritmo 3.1 exibe os passos do CGR-TREE. Os conjuntos A e R denotam, respectivamente, os vértices que garantidamente se tornarão conhecedores e os vértices que precisam tornar-se disseminadores para que os vértices em A tornem-se, de fato, conhecedores. Além disso, p_v denota o vértice pai do vértice v. A cada iteração, se o vértice visitado v difere da raiz r, então há três casos a analisar.

No caso 1 (linha 9), $t(v)$ atingiu o valor 0 graças aos filhos de v e, portanto, v se tornará um disseminador. Em consequência disso, p_v é inserido em A e $t(p_v)$ é decrementado em 1. Note que nada é feito a respeito dos filhos de v (caso existam), uma vez que, nesse ponto, eles já foram visitados, avaliados e removidos de T.

No caso 2 (linha 12), verifica-se que v precisa se tornar disseminador, pois $v \in R$ e

Algoritmo 3.1: Algoritmo `CGR-TREE` [21].

Entrada: Árvore $T = (V, E)$; função limiar $t : V \to \mathbb{N}^*$
Saída : Conjunto semente perfeito S de cardinalidade mínima

```
1  S: Conjunto semente em construção
2  A: Conjunto dos vértices que garantidamente se tornarão conhecedores
3  R: Conjunto dos vértices que precisam se tornar disseminadores
4  p_v: Denota o pai de v em T
5  S ← ∅; A ← ∅; R ← ∅
6  Ordene V = {v_1, v_2, ..., v_n} de acordo com a visitação numa busca em largura
7  para v = v_n até v_1 faça
8      se v ≠ r então                                          ▷ v não é a raiz de T
9          se t(v) = 0 então                                   ▷ Caso 1
10             t(p_v) ← t(p_v) − 1
11             A ← A ∪ {p_v}
12         senão se (v ∈ R) ∧ (t(v) ≥ 2) então                 ▷ Caso 2
13             S ← S ∪ {v}
14             t(p_v) ← t(p_v) − 1
15             A ← A ∪ {p_v}
16         senão se (v ∉ A) ∨ (v ∈ R ∧ t(v) = 1) então         ▷ Caso 3
17             R ← R ∪ {p_v}                                   ▷ p_v precisa disseminar
18     senão se (t(v) > 0) ∧ (v ∉ A \ R) então                 ▷ v é raiz de T
19         S ← S ∪ {v}
20 retorna S
```

$t(v) \geq 2$. Nesse ponto, o único vizinho de v em T é p_v. Logo, não há vizinhos de v suficientes para tornar v disseminador. Portanto, v deve pertencer ao conjunto semente. Em consequência, v é inserido em S, p_v é inserido em A e $t(p_v)$ é decrementado em 1.

No caso 3 (linha 16), ocorre pelo menos um dos dois seguintes cenários: não há garantias de que v se torne conhecedor, pois $v \notin A$, ou v precisa ser disseminador, pois $v \in R$ e $t(v) = 1$. Note que, nesse ponto, o único vizinho de v em T é p_v e a disseminação de p_v é suficiente para que v também dissemine (se tornando conhecedor também). Portanto, v deve receber a disseminação de p_v, mas não deve pertencer ao conjunto semente. Em consequência disso, p_v é inserido em R.

Quando a raiz r é visitada (linha 18), r é o único vértice restante em T. Assim, se $t(r) > 0$ e não há garantias de que r se torne conhecedor por meio de seus filhos, então r deve pertencer ao conjunto semente e é adicionado a S.

É fácil ver que a complexidade de tempo do `CGR-TREE` é linear no tamanho da entrada. Uma prova de sua corretude pode ser vista em [21].

O segundo algoritmo, a ser denotado por `CGR`, consiste em uma generalização do `CGR-TREE` e devolve um conjunto semente perfeito (não necessariamente de cardinalidade mínima) para um grafo arbitrário, ou seja, trata-se de uma heurística para o `PAP`. O algoritmo opera de maneira gulosa e iterativamente rejeita a inserção de vértices no conjunto semente S, até que uma determinada condição ocorra e faça com que algum vértice seja inserido em S. O algoritmo é encerrado quando todo vértice foi descartado

ou selecionado como semente. A seguir, descrevemos os conjuntos mantidos pelo `CGR`:

- S denota o conjunto semente em construção;
- U representa o conjunto de vértices que ainda não foram removidos do grafo original;
- *Temp* denota o conjunto de vértices movidos para um estado temporário de espera. Esses vértices ainda pertencem a U, mas não podem contribuir para que seus respectivos vizinhos tornem-se disseminadores;
- R representa o conjunto de vértices que precisam se tornar disseminadores, mas que, a partir do S corrente, não se tornariam;
- A é o conjunto dos vértices que tornam-se conhecedores assumindo que todos os vértices em R se tornarão disseminadores.

O `CGR`, exibido no Algoritmo 3.2, funciona da seguinte maneira. Enquanto existir ao menos um vértice que seja ignorante ou que pertença a R, o caso 3 (linha 24) ocorre e um vértice v é selecionado de acordo com uma determinada função. Em seguida, v é movido para *Temp*. Como consequência de v pertencer a *Temp*, todos os vizinhos de v não contarão com v para tornarem-se disseminadores. Para esse efeito, para cada $u \in N(v)$, o valor do grau de u restrito aos vértices em $U \setminus Temp$, denotado por $\delta(u)$, é decrementado em 1.

Devido à atualização do grafo feita no caso 3, podemos ter a ocorrência de ao menos um dos dois seguintes cenários: existe $v \notin A$ tal que $\delta(v) = 0$ ou existe $v \in (U - Temp) \cap R$ tal que $\delta(v) < t(v)$. Se pelo menos um desses cenários ocorre, então o caso 2 se aplica (linha 18) e v é adicionado ao conjunto semente e removido do grafo. Além disso, se u é vizinho de v, então temos que u possui agora um vizinho disseminador a mais do que tinha antes. Assim, o valor de $k(u)$ é decrementado em 1, onde $k(u)$ denota o número restante de novos vizinhos disseminadores que u precisa ter para se tornar disseminador.

Por outro lado, pode ser que o grafo atualizado contenha algum vértice v tal que $k(v)$ foi progressivamente decrementado até atingir 0. Isso significa que o conjunto semente corrente é suficiente para que a propagação faça com que v se torne disseminador. Se existe v nessas condições, então o caso 1 se aplica e v é removido do grafo. Além disso, para cada vizinho u de v, decrementa-se $k(u)$ em 1.

Observe que o caso 1 também se aplica aos vértices que estão em *Temp*. Nesse cenário, o valor δ dos vizinhos do vértice v já foram reduzidos em 1 quando v foi movido para *Temp* e, portanto, nenhuma redução adicional é feita. Por construção, uma vez que um vértice é movido para *Temp*, este será removido do grafo apenas pelo caso 1. De fato, os casos 2 e 3 aplicam-se somente para os vértices que não estão em *Temp*. Isso significa que se um vértice foi movido para *Temp*, então ele não pertencerá ao conjunto semente.

Quando o caso 3 se aplica, a ideia é identificar vértices que não devem pertencer ao conjunto semente. Assim, dois subcasos são considerados. Se o grafo ainda contém vértices que não pertencem a R, então dentre esses vértices, aquele que possui valor mínimo de δ é movido para *Temp*. Do contrário, todos os vértices no grafo devem se

Algoritmo 3.2: Algoritmo `CGR` [21].

Entrada: Grafo $G = (V, E)$; função limiar $t : V \to \mathbb{N}^*$
Saída : Conjunto semente perfeito S

S: Conjunto semente em construção
U: Conjunto de vértices não removidos do grafo
Temp: Conjunto de vértices em estado temporário de espera
A: Conjunto dos vértices que garantidamente se tornarão conhecedores
R: Conjunto dos vértices que precisam se tornar disseminadores
$S \leftarrow \emptyset$; $A \leftarrow \emptyset$; $R \leftarrow \emptyset$; $Temp \leftarrow \emptyset$; $U \leftarrow V$
para cada $v \in V$ **faça**
    $k(v) \leftarrow t(v)$
    $\delta(v) \leftarrow |N(v)|$
enquanto $(A \neq V) \vee (R \neq \emptyset)$ **faça**
    se $\exists v \in U$ *tal que* $k(v) = 0$ **então** ▷ Caso 1
        para cada $u \in N(v) \cap U$ **faça**
            $k(u) \leftarrow \max\{k(u) - 1, 0\}$
            $A \leftarrow A \cup \{u\}$
            se $v \notin Temp$ **então**
                $\delta(u) \leftarrow \delta(u) - 1$
        $U \leftarrow U \setminus \{v\}$; $R \leftarrow R \setminus \{v\}$; $A \leftarrow A \cup \{v\}$
    senão se $(\exists v \in (U \setminus Temp) \cap R \ t.q. \ \delta(v) < k(v)) \vee (\exists v \notin A \ t.q. \ \delta(v) = 0)$ **então** ▷ Caso 2
        $S \leftarrow S \cup \{v\}$
        para cada $u \in N(v) \cap U$ **faça**
            $k(u) \leftarrow k(u) - 1$
            $\delta(u) \leftarrow \delta(u) - 1$
        $U \leftarrow U \setminus \{v\}$; $R \leftarrow R \setminus \{v\}$; $A \leftarrow A \cup \{v\}$
    senão ▷ Caso 3
        se $U \setminus Temp \setminus R \neq \emptyset$ **então**
            $v \leftarrow \text{argmin}_{w \in U \setminus (Temp \cup R)}\{\delta(w)\}$
            se $v \notin A$ **então**
                $R \leftarrow R \cup \{x\}$, onde $x = \text{argmax}_{w \in N(v) \cap (U \setminus Temp)}\{\delta(w)\}$
                para cada $z \in N(x) \cap U$ **faça**
                    $A \leftarrow A \cup \{z\}$
        senão
            $v \leftarrow \text{argmax}_{w \in R} \left\{ \dfrac{k(w)}{\delta(w)(\delta(w)+1)} \right\}$
        para cada $u \in N(v)$ **faça**
            $\delta(u) \leftarrow \delta(u) - 1$
        $Temp \leftarrow Temp \cup \{v\}$; $R \leftarrow R \setminus \{v\}$; $A \leftarrow A \cup \{v\}$
retorna S

tornar disseminadores e a escolha do vértice que não deve pertencer ao conjunto semente

é feita a partir de uma métrica descrita em [20].

A complexidade de tempo do CGR é $\mathcal{O}(|E| \log |V|)$. Uma prova de sua corretude pode ser vista em [21].

Em [21] foram conduzidos experimentos com o CGR em 17 instâncias do PAP provenientes de redes sociais reais. Além do CGR, foram testadas duas heurísticas chamadas MTS [20] e DOM [12], as quais foram originalmente projetadas para os problemas TSSP e MDSP, respectivamente. Esses três algoritmos possuem a mesma complexidade assintótica de tempo. Os resultados mostraram que os tamanhos dos conjuntos sementes perfeitos obtidos via CGR, foram, em geral, menores que aqueles obtidos pelos outros dois algoritmos.

Até o momento da escrita desta Dissertação, não foram encontradas outras publicações que tratassem diretamente do PAP. Uma versão preliminar do artigo [21] foi citada em [15], mas apenas como um dos trabalhos ligados ao mesmo tema.

3.4 Problemas Relacionados

Nesta seção, apresentamos quatro problemas problemas de otimização que estão relacionados ao PAP. Todos eles pertencem ao tópico de *marketing* viral e são baseados no modelo de propagação LT.

Um dos problemas de *marketing* viral mais famosos da literatura é chamado de Problema da Maximização de Influência (IMP do inglês *Influence Maximization Problem*), inicialmente formalizado em [35]. Nesse problema, cada vértice pode assumir apenas um dos seguintes estados durante a propagação: ativo ou inativo.

Problema 2 (IMP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$, uma função limiar $t : V \to \mathbb{N}^*$ e um inteiro positivo k, o objetivo do IMP é encontrar um conjunto semente de tamanho k que maximiza a quantidade de vértices ativos ao fim da propagação.*

O IMP é provadamente NP-difícil [35] e tem sido bastante estudado na literatura, dando origem a diversas versões [48, 53].

Para o TSSP (abaixo), citado na seção anterior, também vale que cada vértice só pode assumir dois estados: ativo ou inativo.

Problema 3 (TSSP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$, uma função limiar $t : V \to \mathbb{N}^*$, o objetivo do TSSP é encontrar um conjunto semente de tamanho mínimo de modo que todos os vértices sejam ativos ao fim da propagação.*

Percebe-se que qualquer solução viável do TSSP pode ser transformada em uma solução viável do PAP, mas a recíproca é falsa.

Uma formulação de PLI foi proposta em [1] para uma versão do TSSP onde o grafo G fornecido é direcionado. Entre outros resultados, os autores apresentam limitantes superiores e inferiores para esse problema. Existem diversas variações do TSSP na literatura, incluindo versões do problema com restrições de tempo e orçamento [16].

A seguir, temos o Prolema do Conjunto Evangelizador Perfeito (PESP do inglês *Perfect Evangelizing Set Problem*), proposto em [19]. O PESP consiste em uma generalização do

PAP. Na entrada do PESP são fornecidos um grafo não direcionado $G = (V, E)$ e duas funções de valor limiar $t_E : V \to \mathbb{N}^*$ e $t_C : V \to \mathbb{N}^*$ tais que $\forall v \in V\, t_C(v) \leq t_E(v) \leq d(v) + 1$. Para toda rodada $\tau \in \mathbb{N}$, cada vértice v assume ao menos um estado em relação à informação que está sendo propagada:

- Agnóstico: se até $t_C(v) - 1$ vizinhos passaram a informação a v e, por conseguinte, este ainda não foi convertido;
- Convertido: se v recebeu a informação por meio de no mínimo $t_C(v)$ vizinhos e, portanto, já foi convertido;
- Evangelizador: se v recebeu a informação a partir de ao menos $t_E(v)$ vizinhos e, neste caso, já a dissemina.

Note que um vértice não pode ser agnóstico e convertido ao mesmo tempo. Por outro lado, todo vértice evangelizador também é convertido, mas a recíproca é falsa. O processo de disseminação começa na rodada $\tau = 0$ a partir de um conjunto inicial de evangelizadores $S \subseteq V$, que é análogo a um conjunto semente. Em uma propagação iniciada por S, denota-se por $\text{Evg}_G[S, \tau]$ o conjunto de vértices que são evangelizadores na rodada τ. Dessa maneira, $\text{Evg}_G[S, \tau]$ é igual a:

- S, se $\tau = 0$
- $\text{Evg}_G[S, \tau - 1] \cup \{u : |N_G(u) \cap \text{Evg}_G[S, \tau - 1]| \geq t_E(u)\}$, se $\tau \geq 1$.

Analogamente, denota-se por $\text{Conv}_G[S, \tau]$ o conjunto de vértices que estão no estado de convertido na rodada τ. Assim, $\text{Conv}_G[S, \tau]$ é igual a:

- S, se $\tau = 0$
- $\text{Conv}_G[S, \tau - 1] \cup \{u : |N_G(u) \cap \text{Evg}_G[S, \tau - 1]| \geq t_C(u)\}$, se $\tau \geq 1$.

O fim da propagação ocorre quando $\text{Evg}_G[S, \rho] = \text{Evg}_G[S, \rho - 1]$ para algum $\rho \geq 1$. Isto é, quando nenhum novo evangelizador surge na transição entre rodadas. Os conjuntos finais de evangelizadores e convertidos são denotados e descritos, respectivamente, por $\text{Evg}_G[S] = \text{Evg}_G[S, \rho]$ e $\text{Conv}_G[S] = \text{Conv}_G[S, \rho]$.

Problema 4 (PESP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$ e duas funções de valor limiar $t_E : V \to \mathbb{N}^*$ e $t_C : V \to \mathbb{N}^*$ tais que $\forall v \in V\, t_C(v) \leq t_E(v)$, o objetivo do* PESP *é encontrar um conjunto semente de tamanho mínimo tal que $Conv_G[S] = V$.*

Note que uma solução S é viável para o PESP se, e somente se, $\text{Conv}_G[S] = V$. Além disso, o PAP é um caso especial do PESP em que $t_C(v) = 1$ para todo $v \in V$.

Em [19], os autores utilizam os conceitos de *decomposição em árvore* e de *largura de árvore* de um grafo. Sejam $G = (V, E)$ um grafo e $(\mathcal{T}, \mathcal{X})$, onde $\mathcal{X}$ é uma família de subconjuntos de V e $\mathcal{T}$ é uma árvore cujos vértices correspondem aos elementos em $\mathcal{X}$. O par $(\mathcal{T}, \mathcal{X})$ é uma decomposição em árvore de G, se satisfaz as seguintes condições:

(i) A união dos subgrafos de G induzidos por cada $X \in \mathcal{X}$ é igual a G;

(ii) Para todo $v \in V$, os vértices pertencentes a $\{X \in \mathcal{X} \mid v \in X\}$ são conectados em $\mathcal{T}$.

A *largura* de $\mathcal{T}$ é igual a $\max_{X \in \mathcal{X}}\{|X| - 1\}$ e a largura de árvore de G é igual a menor das larguras de todas as decomposições em árvore de G que satisfazem certas restrições descritas em [19].

Um algoritmo paramétrico para o PESP em função da largura de árvore de G é fornecido em [19]. Os autores mostram que se a largura de árvore de G é igual a w, então o PESP pode ser solucionado em tempo $|V|^{O(w)}$ por meio desse algoritmo.

Nesse mesmo artigo, foi proposto um algoritmo para o PESP para grafos densos que atendem à seguinte propriedade: para todo $v \in V$, $d(v) \geq (|V| + \alpha + \beta)/2 - 2$, onde α e β são constantes tais que $t_E(v) \leq \alpha$, $t_C(v) \leq \beta$, e $\alpha + \beta \leq |V| + 2$. Para esse tipo de grafo, o algoritmo retorna uma solução viável de tamanho menor ou igual a $2\alpha - 2$. Além disso, se $d(v) \geq |V|/2$ para todo $v \in V$ e $\alpha = \beta = 2$, então este mesmo algoritmo provê uma solução ótima de tamanho 2.

Até o momento da escrita desta Dissertação, não foram encontradas outras publicações que tratassem diretamente do PESP. O artigo [19] foi citado por [21], ou seja, pelo artigo que propôs o PAP, e pelo artigo [18], mas nesse caso apenas como trabalho relacionado.

Por fim, temos o Problema do Conjunto de Evangelização Máxima (MESP do inglês *Maximally Evangelizing Set Problem*), proposto em [19]. O MESP consiste na versão de maximização do PESP. A entrada do MESP é quase idêntica à do PESP e o processo de propagação é o mesmo para ambos os problemas.

Problema 5 (MESP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$, uma constante B e duas funções de valor limiar $t_E : V \to \mathbb{N}^*$ e $t_C : V \to \mathbb{N}^*$ tais que $\forall v \in V\, t_C(v) \leq t_E(v)$, o objetivo do MESP é encontrar um conjunto semente de tamanho no máximo B, tal que o conjunto final de vértices convertidos seja máximo.*

Uma solução S é viável para o MESP se, e somente se, $|S| \leq B$. Observe que diferentemente do PESP, uma solução viável para o MESP não necessariamente faz com que todos os vértices tornem-se convertidos.

Em [19], é provado que é NP-difícil aproximar o MESP por um fator de $n^{1-\varepsilon}$ para qualquer $\varepsilon > 0$, mesmo quando $t_C(v) = 1$ para cada $v \in V$. Além disso, os autores apresentam algoritmos polinomiais para o MESP para os casos em que o grafo dado na entrada é completo ou é uma árvore.

Adicionalmente, o artigo contempla o estudo de uma parametrização do MESP que utiliza o conceito de *diversidade de vizinhança*. Um grafo $G = (V, E)$ possui diversidade de vizinhança σ, se V pode ser particionado em até σ conjuntos $V_1, V_2, \ldots, V_\sigma$, tais que para quaisquer dois vértices $u, v \in V_i$, com $i \in \{1, 2, \ldots, \sigma\}$, $N(v) \setminus \{u\} = N(u) \setminus \{v\}$. Os autores apresentam um algoritmo de parâmetro fixo tratável (FPT) para o MESP com parâmetros σ e B.

No próximo capítulo, apresentamos as abordagens empregadas neste trabalho para solucionar o PAP.

Capítulo 4

Metodologia

Neste capítulo, apresentamos os métodos utilizados neste trabalho para solucionar o PAP. Na Seção 4.1, introduzimos técnicas de pré-processamento de instâncias que visam a diminuição do tamanho da entrada. A Seção 4.2 apresenta duas formulações de PLI para obtenção de soluções ótimas para o PAP. Por fim, na Seção 4.3, descrevemos quatro heurísticas baseadas na meta-heurística GRASP que resolvem o PAP eficientemente, mas sem garantia de otimalidade.

Ao longo do texto, sempre que nos referirmos a uma solução S para uma instância do PAP, significa que S pode ser viável ou não, ou seja, S é um conjunto semente que pode ou não ser perfeito, já que S é viável se, e somente se, S é um conjunto semente perfeito. Ademais, S ser uma solução ótima é equivalente a S ser viável e $|S|$ ser mínima.

Além disso, dada uma formulação de PLI para o PAP definida sobre um conjunto C de variáveis inteiras, dizemos que uma função $F : C \to \mathbb{Z}$ de atribuição de valores às variáveis em C é uma *solução do modelo* somente se $F(C)$ não viola as restrições da formulação. Se F é uma solução que minimiza a função objetivo, então F é uma *solução ótima* do modelo.

4.1 Pré-processamento

Realizar procedimentos de pré-processamento de instâncias é uma prática comum na área de otimização combinatória. Aqui, apresentamos três diferentes técnicas úteis para reduzir o tamanho de instâncias do PAP.

4.1.1 Decomposição em Componentes Conexas

Uma abordagem simples, mas frequentemente utilizada na literatura, consiste em fragmentar instâncias grandes em subinstâncias menores, a fim de permitir a resolução do problema mediante uma estratégia de divisão e conquista. Esta é a ideia por trás da nossa primeira técnica de pré-processamento, apresentada a seguir.

Seja $I = (G = (V, E), t)$ uma instância qualquer do PAP. Suponha que G possui $c \geq 2$ componentes conexas G_k para $k \in \{1, 2, \ldots, c\}$ e que S seja um conjunto semente perfeito para I. Durante o processo de propagação do PAP em G a partir de S, a propagação que ocorre em G_k depende exclusivamente das sementes em $G_k \cap S$.

Dito isto, nossa primeira estratégia de pré-processamento consiste em decompor I em c subinstâncias $I_1, I_2, \ldots, I_c$, tais que $I_k = \left(G_k, t\big|_{G_k}\right)$, onde $t\big|_{G_k}$ denota a função t com domínio restrito ao conjunto de vértices de G_k. Em seguida, resolvemos separadamente cada instância I_k e utilizamos a união das soluções viáveis obtidas para construir uma solução viável para I. O Teorema 4.1 e o Corolário 4.1 garantem a corretude da técnica.

Teorema 4.1. *Se $S_1, S_2, \ldots, S_c$ são soluções viáveis de $I_1, I_2, \ldots, I_c$, respectivamente, então $\bigcup_{k=1}^{c} S_k$ é solução viável para I.*

Demonstração. Suponha que $S_1, S_2, \ldots, S_c$ sejam soluções viáveis de $I_1, I_2, \ldots, I_c$. Então para cada I_k, com $k \in \{1, 2, \ldots, c\}$, existe $\rho_k > 0$, tal que a propagação do `PAP` com ρ_k rodadas em G_k a partir de S_k termina com todos os vértices de G_k conhecedores. Seja $\rho = \max\{\rho_1, \rho_2, \ldots, \rho_c\}$. É evidente que uma propagação em G com ρ rodadas a partir de $\bigcup_{k=1}^{c} S_k$ termina com todos os vértices de G conhecedores. Logo $\bigcup_{k=1}^{c} S_k$ é uma solução viável para I. □

Corolário 4.1. *Se $S_1, S_2, \ldots, S_c$ são soluções ótimas de $I_1, I_2, \ldots, I_c$, respectivamente, então $\bigcup_{k=1}^{c} S_k$ é solução ótima para I.*

Demonstração. Sejam $S_1, S_2, \ldots, S_c$ soluções ótimas de $I_1, I_2, \ldots, I_c$. Como tais soluções são viáveis, pelo Teorema 4.1, $S = \bigcup_{k=1}^{c} S_k$ é uma solução viável para I.

Suponha com o propósito de obter uma contradição que S não é solução ótima para I. Então, existe uma solução viável S' para I tal que $|S'| < |S|$. Como S' é viável, então existe $\rho > 0$, tal que a propagação do `PAP` com ρ rodadas em G a partir de S' termina com todos os vértices de G conhecedores. Assim, para todo $k \in \{1, 2, \ldots, c\}$, uma propagação do `PAP` em G_k com ρ rodadas a partir de $S'_k = S' \cap G_k$ termina com todos os vértices de G_k conhecedores. Logo, S'_k é uma solução viável para I_k, $\forall\, k \in \{1, 2, \ldots, c\}$. Mas, $\sum_{k=1}^{c} |S'_k| = |S'| < |S| = \sum_{k=1}^{c} |S_k|$ implica que existe $k \in \{1, 2, \ldots, c\}$ tal que $|S'_k| < |S_k|$, o que contradiz o fato de que S_k é ótima para I_k para todo $k \in \{1, 2, \ldots, c\}$. □

A identificação das componentes conexas de G e a decomposição de I em subinstâncias podem ser feitas em tempo linear no tamanho de G. Já a obtenção da solução da instância original pode ser feita em tempo linear na soma dos tamanhos das soluções das subinstâncias.

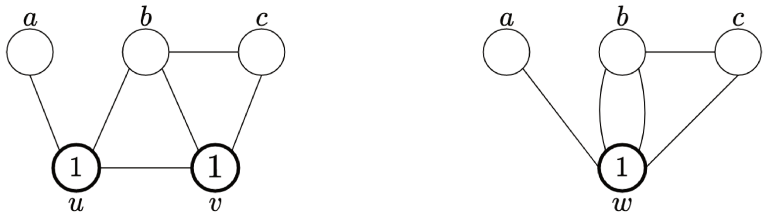
4.1.2 Contração de Arestas

Nossa segunda técnica de pré-processamento realiza a contração de arestas cujos extremos possuem limiares iguais a 1. Seja $I = (G = (V, E), t)$ uma instância qualquer do `PAP`. Suponha que existem $u, v \in V$, tais que $t(u) = t(v) = 1$ e $\{u, v\} \in E$. Nesse caso, se u é disseminador em uma rodada τ, mas v não o é, então, na rodada $\tau + 1$, v se tornará disseminador, pois $t(v) = 1$.

Por isso, podemos representar u e v através de um único vértice com limiar igual a 1, ou seja, podemos construir uma instância $I' = (G', t')$, tal que $G' = (V', E')$ é obtido a partir de G pela contração da aresta $\{u, v\}$, o que resulta em $V' = (V \setminus \{u, v\}) \cup \{w\}$. Além disso, $t'(w) = 1$ e para qualquer vértice x em G' com $x \neq w$, $t'(x) = t(x)$.

Ademais, w torna-se extremo das arestas em G' correspondentes àquelas em G que possuíam u ou v como extremo (exceto a aresta $\{u, v\}$) e, portanto, podemos ter a ocorrência de arestas múltiplas em G'.

A Figura 4.1 apresenta um exemplo do procedimento de contração, onde os limiares de u, v e w estão indicados dentro de seus respectivos círculos e os limiares dos demais vértices foram omitidos.



(a) Grafo G. (b) Grafo G' obtido de G.

Figura 4.1: Exemplo de contração da aresta $\{u, v\}$ de G.

Dessa maneira, podemos obter uma solução viável S' para a instância I' e, a partir de S' construir uma solução viável S para I, de modo que se $w \in S'$, então $u \in S$ ou $v \in S$. O Teorema 4.2 e o Corolário 4.2 garantem a corretude da técnica.

Teorema 4.2. *Seja $I = (G = (V, E), t)$ com $\{u, v\} \in E$ e $t(u) = t(v) = 1$. Seja $I' = (G', t')$ tal que G' é obtido de G por contração de $\{u, v\}$ resultante num vértice w e que $t'(w) = 1$ e $t'(x) = t(x)$ para todo vértice x de G' diferente de w. Se S' é uma solução viável para I', então podemos construir uma solução viável S para I com $|S| = |S'|$.*

Demonstração. Seja S' uma solução viável para I'. Iremos construir uma solução S para I a partir de S'. Se $w \notin S'$, então podemos construir S tal que cada vértice de S' é mapeado ao vértice de S de mesmo identificador. Se $w \in S'$, então podemos construir S tal que w é mapeado a u ou a v e cada um dos demais vértices de S' é mapeado ao vértice de S com mesmo identificador. Em ambos os casos, é evidente que uma propagação em G' a partir de S' torna w disseminador se, e somente se, uma propagação em G a partir de S torna u e v disseminadores. Logo, a propagação em G' é análoga à propagação em G e, portanto S é viável para I. Além disso, por construção de S, vale que $|S| = |S'|$. □

Corolário 4.2. *Seja $I = (G = (V, E), t)$ com $\{u, v\} \in E$ e $t(u) = t(v) = 1$. Seja $I' = (G', t')$ tal que G' é obtido de G por contração de $\{u, v\}$ resultante num vértice w e que $t'(w) = 1$ e $t'(x) = t(x)$ para todo vértice x de G' diferente de w. Se S' é uma solução ótima para I', então podemos construir uma solução ótima S para I com $|S| = |S'|$.*

Demonstração. Seja S' uma solução ótima para I'. Como S' é viável para I', então de acordo com o Teorema 4.2, podemos construir uma solução S viável para I tal que $|S| = |S'|$.

Suponha com o propósito de obter uma contradição que S não é ótima. Então, existe uma solução viável X para I tal que $|X| < |S|$. Logo, temos apenas dois casos: u e v não pertencem a X ou ao menos um dos dois vértices pertence a X. Agora, iremos construir uma solução X' para I' a partir de X.

Se $u \notin X$ e $v \notin X$, então podemos construir X' de modo que cada vértice de X é mapeado ao vértice de X' de mesmo identificador. Por outro lado, suponha, sem perda de generalidade, que $u \in X$. Então, podemos construir X' de modo que u é mapeado em w e cada um dos demais vértices de X é mapeado ao vértice de X' de mesmo identificador.

Em ambos os casos, é evidente que uma propagação em G a partir de X torna u e v disseminadores se, e somente se, uma propagação em G' a partir de X' torna w disseminador. Logo, a propagação em G é análoga à propagação em G' e, portanto, X' é viável para I'. Mas por construção de X', vale que $|X'| = |X| < |S| = |S'|$, o que contradiz o fato de que S' é ótima para I'. □

Nossa abordagem consiste em realizar sucessivas contrações até que não exista um par de vértices que atenda às pré-condições para a aplicação da técnica. A instância obtida ao fim das sucessivas contrações é resolvida e a solução viável obtida é transformada em uma solução viável da instância original em tempo linear[5] no tamanho de V. Observe que a contração de uma aresta $\{u, v\}$ pode ser realizada em tempo[6] $\mathcal{O}\left(d(u) + d(v) + \sum_{y \in N(u) \cup N(v)} d(y)\right)$.

4.1.3 Colapsação de Vértices

A nossa terceira abordagem de pré-processamento é baseada em colapsação de vértices. Seja $I = (G = (V, E), t)$ uma instância qualquer do PAP. Suponha que um vértice $u \in V$ seja extremidade de uma ou mais arestas (múltiplas[7]) e que tenha apenas um vizinho v. Se $t(v) = 1$, então podemos colapsar u e v em um novo vértice w com valor limiar igual a 1.

A justificativa é que se u é disseminador em uma rodada τ, mas v não o é, então na rodada $\tau + 1$, v se tornará disseminador, pois $t(v) = 1$. Além disso, se v é disseminador em uma rodada τ, mas u não o é, então na rodada $\tau + 1$, u se tornará disseminador, pois $t(u) \leq d(u)$ e todas as arestas que incidem em u também incidem em v. Assim, podemos construir uma instância $I' = (G', t')$, tal que G' é obtido a partir de G através da colapsação de u e v em um novo vértice w. Além disso, $t'(w) = 1$ e $t'(x) = t(x)$ para todo vértice x de G' diferente de w. Note que as únicas arestas das quais w é extremo em G' correspondem àquelas que possuíam v como extremo em G (exceto as que conectavam u e v).

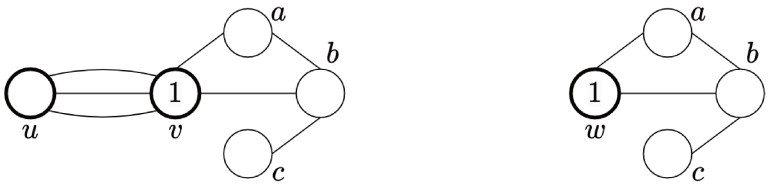
A Figura 4.2 apresenta um exemplo do procedimento de colapsação, onde os limiares de v e w estão indicados dentro de seus respectivos círculos e os limiares dos demais vértices foram omitidos.

Dessa maneira, podemos obter uma solução viável S' para a instância I' e, a partir de S' construir uma solução viável S para I, de modo que se $w \in S'$, então $u \in S$ ou $v \in S$. O Teorema 4.3 e o Corolário 4.3 garantem a corretude da técnica.

[5]Para isso, basta que para cada vértice fruto da contração de uma aresta, associemos uma lista contendo os vértices que eram extremos da aresta que foi contraída.

[6]Assumindo-se uma implementação do grafo com listas de adjacência.

[7]Um cenário com arestas múltiplas pode ocorrer após a aplicação da segunda técnica de pré-processamento, apresentada na Seção 4.1.2.



(a) Grafo G. (b) Grafo G' obtido de G.

Figura 4.2: Exemplo de colapsação dos vértices u e v de G.

Teorema 4.3. *Seja $I = (G = (V, E), t)$ com $\{u, v\} \in E$ e $N(u) = \{v\}$ e $t(v) = 1$. Seja $I' = (G', t')$ tal que G' é obtido de G por colapsação de u e v resultante num vértice w e que $t'(w) = 1$ e $t'(x) = t(x)$ para todo vértice x de G' diferente de w. Se S' é uma solução viável para I', então podemos construir uma solução viável S para I tal que $|S'| = |S|$.*

Demonstração. Semelhante à demonstração do Teorema 4.2. □

Corolário 4.3. *Seja $I = (G = (V, E), t)$ com $\{u, v\} \in E$ e $N(u) = \{v\}$ e $t(v) = 1$. Seja $I' = (G', t')$ tal que G' é obtido de G por colapsação de u e v resultante num vértice w e que $t'(w) = 1$ e $t'(x) = t(x)$ para todo vértice x de G' diferente de w. Se S' é uma solução ótima para I', então podemos construir uma solução ótima S para I tal que $|S'| = |S|$.*

Demonstração. Semelhante à demonstração do Corolário 4.2. □

Para essa estratégia, nossa abordagem consiste em realizar sucessivas colapsações até que não exista um par de vértices que atenda às pré-condições para a aplicação da técnica. A instância obtida ao fim das sucessivas colapsações é resolvida e a solução viável obtida é transformada em uma solução viável da instância original em tempo linear no tamanho de V. Observe que a colapsação dos vértices u e v pode ser realizada em tempo $\mathcal{O}\left(d(u) + d(v) + \sum_{y \in N(u) \cup N(v)} d(y)\right)$.

4.2 Formulações de Programação Linear Inteira

Nesta seção, apresentamos duas formulações de programação linear inteira para o `PAP`. Como veremos no Capítulo 5, tais formulações foram criadas com o objetivo de obter valores ótimos para as instâncias utilizadas nos experimentos, a fim de compararmos os valores das soluções produzidas pelas nossas heurísticas com esses valores.

4.2.1 Modelo `PLI-RODADAS`

Seja $I = (G = (V, E), t)$ uma instância do `PAP` tal que $|V| = n$. A seguir, descrevemos a nossa primeira formulação de `PLI` para o `PAP`, denotada por `PLI-RODADAS`. Considere o conjunto de variáveis binárias $C = \{s_{v,\tau} : v \in V, \tau \in \{0, 1, \ldots, n\}\}$ do modelo `PLI-RODADAS` de I. Mostraremos que é possível obter uma solução ótima para I, a partir de uma solução ótima da formulação do modelo `PLI-RODADAS` de I. De modo resumido, podemos tomar uma solução ótima da formulação e construir um conjunto semente $S = \{v \in V \mid s_{v,0} = 1\}$.

Como veremos mais à frente, é verdade que na propagação do PAP a partir de S, v é disseminador na rodada τ se, e somente se, $s_{v,\tau} = 1$. Usaremos esse fato para mostrar que S é solução ótima para I. Eis a formulação:

$$\min z = \sum_{v \in V} s_{v,0} \tag{4.1}$$

sujeito a:

$$s_{v,\tau} - s_{v,\tau-1} \geq 0 \qquad \forall v \in V \, \forall \tau \in [1,n] \tag{4.2}$$

$$\sum_{u \in N(v)} (s_{u,\tau-1}) - t(v)(s_{v,\tau} - s_{v,0}) \geq 0 \qquad \forall v \in V \, \forall \tau \in [1,n] \tag{4.3}$$

$$\sum_{u \in N(v)} (s_{u,\tau-1}) - (d(v) - t(v) + 1)s_{v,\tau} \leq t(v) - 1 \qquad \forall v \in V \, \forall \tau \in [1,n] \tag{4.4}$$

$$s_{v,0} + \sum_{u \in N(v)} s_{u,n-1} \geq 1 \qquad \forall v \in V \tag{4.5}$$

$$s_{v,\tau} \in \{0,1\} \qquad \forall v \in V \, \forall \tau \in [0,n] \tag{4.6}$$

A função objetivo (4.1) minimiza a quantidade de disseminadores na rodada 0, isto é, o tamanho do conjunto semente. A restrição (4.2) garante que um vértice v que tenha se tornado disseminador permaneça nesse estado até o fim da propagação. A restrição (4.3) impede que um vértice v que não é semente torne-se disseminador enquanto a quantidade de seus vizinhos disseminadores for menor do que o seu limiar.

De maneira complementar a (4.3), a restrição (4.4) força um vértice v a se tornar disseminador quando a quantidade de vizinhos disseminadores atinge seu limiar. Ademais, a restrição (4.5) garante que a solução só é válida se todos os vértices são conhecedores ao fim da propagação. Por fim, a restrição (4.6) garante que as variáveis $s_{v,\tau}$ sejam binárias. O número total de variáveis é da ordem de $|V|^2$. A seguir, justificamos a corretude da formulação.

Lema 4.1. *Seja $I = (G = (V,E), t)$ uma instância do PAP. Se P é a propagação do PAP a partir de uma solução viável S de I, então a função $F_P : C \to \{0,1\}$ de atribuição de valores às variáveis do modelo PLI-RODADAS de I definida por $F_P(s_{v,\tau}) = 1 \leftrightarrow v \in Dis_G[S,\tau]$ é uma solução do modelo PLI-RODADAS de I.*

Demonstração. Seja $I = (G = (V,E), t)$ uma instância do PAP. Suponha que P é a propagação do PAP a partir de uma solução viável S de I. Seja $F_P : C \to \{0,1\}$ uma atribuição de valores às variáveis do modelo PLI-RODADAS de I definida por $F_P(s_{v,\tau}) = 1 \leftrightarrow v \in \mathrm{Dis}_G[S,\tau]$. Mostraremos que F_P é uma solução do modelo PLI-RODADAS de I.

Pela definição de propagação do PAP, temos que se $v \in \mathrm{Dis}_G[S,\tau-1]$, então $v \in \mathrm{Dis}_G[S,\tau]$, para todo $\tau \geq 1$. Assim, se $F_P(s_{v,\tau-1}) = 1$, então $F_P(s_{v,\tau}) = 1$ e por conseguinte, $F_P(s_{v,\tau}) \geq F_P(s_{v,\tau-1})$ para todo $\tau \geq 1$. Logo, $F_P(C)$ não viola a restrição (4.2).

Agora, suponha com o propósito de obter uma contradição que $F_P(C)$ viola a restrição (4.3). Então, existe $(v,\tau) \in V \times \mathbb{N}^*$ t.q. $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) < t(v)(F_P(s_{v,\tau}) - F_P(s_{v,0}))$. Note que como $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) \geq 0$ e $t(v) > 0$, a violação só ocorre se

$F_P(s_{v,\tau}) - F_P(s_{v,0}) > 0$ e, portanto, $F_P(s_{v,\tau}) = 1$ e $F_P(s_{v,0}) = 0$. Assim, chegamos que $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) < t(v)$. Mas $v \notin \text{Dis}_G[S, 0]$, pois $F_P(s_{v,0}) = 0$ e $|N(v) \cap \text{Dis}_G[S, \tau - 1]| < t(v)$, pois $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) < t(v)$. Logo, pela definição de propagação do PAP, $v \notin \text{Dis}_G[S, \tau]$. Todavia, $F_P(s_{v,\tau}) = 1$, logo $v \in \text{Dis}_G[S, \tau]$, contradição. Portanto, $F_P(C)$ não viola a desigualdade (4.3).

Suponha com o propósito de obter uma contradição que $F_P(C)$ viola a desigualdade (4.4). Então, existe $(v, \tau) \in V \times \mathbb{N}^*$ t.q. $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) > (d(v) - t(v) + 1)F_P(s_{v,\tau}) + t(v) - 1$. Se $F_P(s_{v,\tau}) = 1$, então $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) > d(v)$, contradição, pois $|N(v)| = d(v)$. Por outro lado, se $F_P(s_{v,\tau}) = 0$, então $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) \geq t(v)$. Como $\sum_{u \in N(v)} F_P(s_{u,\tau-1}) \geq t(v)$, então $|N(v) \cap \text{Dis}_G[S, \tau - 1]| \geq t(v)$ e, portanto, $v \in \text{Dis}_G[S, \tau]$. Mas também é verdade que $v \notin \text{Dis}_G[S, \tau]$, pois $F_P(s_{v,\tau}) = 0$, contradição. Assim, $F_P(C)$ não viola a desigualdade (4.4).

Como S é viável e $|V| = n$, então é evidente que na $(n+1)$-ésima rodada todos os vértices são conhecedores. Isso implica que se $\tau = n$, então $v \in \text{Dis}_G[S, 0]$ ou $|N(v) \cap \text{Dis}_G[S, n-1]| \geq 1$. Logo, $F_P(s_{v,0}) + \sum_{u \in N(v)} F_P(s_{u,n-1}) \geq 1$ e, portanto, $F_P(C)$ não viola a desigualdade (4.5). □

Lema 4.2. *Seja $I = (G = (V, E), t)$ uma instância do PAP. Se $F : C \to \{0, 1\}$ é uma solução do modelo PLI-RODADAS de I, então existe uma solução S de I a partir da qual a propagação P do PAP é tal que $v \in Dis_G[S, \tau] \leftrightarrow F(s_{v,\tau}) = 1$, e S é viável.*

Demonstração. Seja $I = (G = (V, E), t)$ uma instância do PAP. Seja $F : C \to \{0, 1\}$ uma atribuição de valores às variáveis do modelo PLI-RODADAS de I que atende às restrições do modelo. A restrição (4.3) não é violada por $F(C)$, logo vale que se $F(s_{v,\tau}) = 1$ para algum $\tau \geq 1$, então a desigualdade só é satisfeita se $F(s_{v,0}) = 1$ ou $\sum_{u \in N(v)} F(s_{u,\tau-1}) \geq t(v)$.

Ademais, como $F(C)$ atende à restrição (4.4), vale que se $\sum_{u \in N(v)} F(s_{u,\tau-1}) \geq t(v)$, então $F(s_{v,\tau}) = 1$ para todo $\tau \geq 1$. Além disso, como $F(C)$ atende à restrição (4.2), então $F(s_{v,\tau}) \geq F(s_{v,\tau-1})$ para todo $\tau \geq 1$. Isso implica que se $F(s_{v,0}) = 1$, então $F(s_{v,\tau}) = 1$. Logo, se $\sum_{u \in N(v)} F(s_{u,\tau-1}) \geq t(v)$ ou $F(s_{v,0}) = 1$, então $F(s_{v,\tau}) = 1$.

Assim, temos que para todo $\tau \geq 1$, $F(s_{v,\tau}) = 1$ se, e somente se $F(s_{v,0}) = 1$ ou $\sum_{u \in N(v)} F(s_{u,\tau-1}) \geq t(v)$.

Tome $S = \{v \in V \mid F(s_{v,0}) = 1\}$ e seja P a propagação do PAP iniciada a partir de S. Vamos mostrar que $v \in \text{Dis}_G[S, \tau] \leftrightarrow F(s_{v,\tau}) = 1$ para todo $\tau \geq 0$. O caso $\tau = 0$ decorre da construção de S. Agora, assuma como hipótese de indução que $v \in \text{Dis}_G[S, \tau] \leftrightarrow F(s_{v,\tau}) = 1$ para algum $\tau \geq 0$.

Como $\tau + 1 \geq 1$, pela definição de propagação do PAP, $v \in \text{Dis}_G[S, \tau + 1]$ se, e somente se, $v \in \text{Dis}_G[S, 0]$ ou $|N(v) \cap \text{Dis}_G[S, \tau]| \geq t(v)$. Pela construção de S, $v \in \text{Dis}_G[S, 0] \leftrightarrow F(s_{v,0}) = 1$. Além disso, por hipótese de indução, $v \in \text{Dis}_G[S, \tau] \leftrightarrow F(s_{v,\tau}) = 1$. Logo, $|N(v) \cap \text{Dis}_G[S, \tau]| \geq t(v) \leftrightarrow \sum_{u \in N(v)} F(s_{u,\tau}) \geq t(v)$. Portanto, $v \in \text{Dis}_G[S, 0]$ ou $|N(v) \cap \text{Dis}_G[S, \tau]| \geq t(v)$ se, e somente se, $F(s_{v,0}) = 1$ ou $\sum_{u \in N(v)} F(s_{u,\tau}) \geq t(v)$.

Mas provamos anteriormente que para todo $\tau \geq 1$, $F(s_{v,0}) = 1$ ou $\sum_{u \in N(v)} F(s_{u,\tau-1}) \geq t(v)$ se, e somente se $F(s_{v,\tau}) = 1$. Logo, $F(s_{v,0}) = 1$ ou $\sum_{u \in N(v)} F(s_{u,\tau}) \geq t(v)$ se, e somente se, $F(s_{v,\tau+1}) = 1$. Concluímos que $v \in \text{Dis}_G[S, \tau + 1] \leftrightarrow F(s_{v,\tau+1}) = 1$.

Por fim, mostraremos que S é viável. Como $F(C)$ não viola a desigualdade (4.5), vale que $F(s_{v,0}) + \sum_{u \in N(v)} F(s_{u,n-1}) \geq 1$. Isso implica que $v \in \text{Dis}_G[S, 0]$ ou $|N(v) \cap$

$\text{Dis}_G[S, n-1]| \geq 1$ e, portanto, ao fim da $(n+1)$-ésima rodada de P, v é conhecedor. Portanto, S é viável para I. □

Teorema 4.4. *Seja $I = (G = (V, E), t)$ uma instância do PAP. S é uma solução viável de I se, e somente se, existe uma solução $F : C \to \{0, 1\}$ do modelo PLI-RODADAS de I tal que $S = \{v \in V \mid F(s_{v,0}) = 1\}$.*

Demonstração. Seja S uma solução de I e seja P a propagação do PAP a partir de S. Pelo Lema 4.1, temos que se S é viável, então existe uma solução $F_P : C \to \{0, 1\}$ do modelo PLI-RODADAS de I tal que $F_P(s_{v,\tau}) = 1 \leftrightarrow v \in \text{Dis}_G[S, \tau]$. Logo, $F_P(s_{v,0}) = 1 \leftrightarrow v \in \text{Dis}_G[S, 0]$ e como $S = \text{Dis}_G[S, 0]$, então $S = \{v \in V \mid F_P(s_{v,0}) = 1\}$.

Reciprocamente, seja $F : C \to \{0, 1\}$ uma solução do modelo PLI-RODADAS de I. Pelo Lema 4.2, existe uma solução S de I a partir da qual a propagação P do PAP é tal que $v \in \text{Dis}_G[S, \tau] \leftrightarrow F(s_{v,\tau}) = 1$. Em particular, $F(s_{v,0}) = 1 \leftrightarrow v \in \text{Dis}_G[S, 0]$ e como $S = \text{Dis}_G[S, 0]$, então $S = \{v \in V \mid F(s_{v,0}) = 1\}$. Mas, pelo Lema 4.2, também temos que S é viável. □

Corolário 4.4. *Seja $I = (G = (V, E), t)$ uma instância do PAP. Se $F : C \to \{0, 1\}$ é uma solução ótima do modelo PLI-RODADAS de I, então $S = \{v \in V \mid F(s_{v,0}) = 1\}$ é uma solução ótima para I.*

Demonstração. Seja $I = (G = (V, E), t)$ uma instância do PAP e seja $F : C \to \{0, 1\}$ uma solução ótima do modelo PLI-RODADAS de I. Tome $S = \{v \in V \mid F(s_{v,0}) = 1\}$. O Teorema 4.4 garante que S é solução viável de I.

Suponha com o propósito de obter uma contradição que S não é ótima para I. Então existe solução viável S' de I tal que $|S'| < |S|$. Assim, pelo Teorema 4.4, existe uma solução $F' : C \to \{0, 1\}$ do modelo PLI-RODADAS de I tal que $F'(s_{v,0}) = 1 \leftrightarrow v \in S'$. Mas $\sum_{v \in V} F'(s_{v,0}) = |S'| < |S| = \sum_{v \in V} F(s_{v,0})$ e isso contradiz o fato de que F é solução ótima do modelo PLI-RODADAS de I. Portanto, S é uma solução ótima para I. □

Desigualdade adicionais A seguir, descrevemos um conjunto de desigualdades adicionais para o modelo PLI-RODADAS. A adição dessas restrições ao PLI-RODADAS mostrou-se útil na prática ao cortar soluções da relaxação linear, acelerando o processo de obtenção de soluções inteiras.

Dado um grafo $G = (V, E)$, dizemos que $D \subseteq V$ é um *conjunto dominante* se todo vértice em $V \setminus D$ é adjacente a ao menos um vértice em D [54]. Assim, em uma propagação do PAP a partir de uma solução viável, sabemos que o conjunto final de disseminadores é um conjunto dominante, pois ao final da propagação, todo vértice que não é disseminador deve possuir ao menos um vizinho que é disseminador.

Recordemo-nos que a propagação termina na rodada $\tau \geq 1$ se nenhum vértice se torna um novo disseminador na transição entre as rodadas $\tau - 1$ e τ. Decorre diretamente daí a restrição (4.7), cuja validade pode ser facilmente verificada, uma vez que se nenhum novo vértice se torna disseminador na transição da rodada $\tau - 1$ para τ, então $\sum_{w \in V}(s_{w,\tau} - s_{w,\tau-1}) = 0$, e portanto, a restrição só é satisfeita se $s_{v,\tau} + \sum_{u \in N(v)} s_{u,\tau} \geq 1$, isto é, v é disseminador ou possui ao menos um vizinho disseminador na rodada τ.

$$s_{v,\tau} + \sum_{u \in N(v)} s_{u,\tau} + \sum_{w \in V} (s_{w,\tau} - s_{w,\tau-1}) \geq 1 \qquad \forall v \in V \; \forall \tau \in [1, n] \qquad (4.7)$$

Note que não existe solução viável do PAP cuja propagação seja representada por uma solução do modelo PLI-RODADAS que viola a desigualdade (4.7). Portanto, a restrição (4.7) não corta soluções inteiras e sua inclusão no modelo se justifica por acelerar a resolução, na prática.

4.2.2 Modelo PLI-ARCOS

Nesta seção, descrevemos uma segunda formulação de PLI para o PAP, denotada por PLI-ARCOS[8]. Como veremos mais adiante, dada uma solução ótima do modelo PLI-ARCOS, os valores atribuídos às variáveis da formulação determinam um grafo dirigido que representa uma propagação do PAP a partir de um conjunto semente perfeito de tamanho mínimo.

Sejam $I = (G = (V, E), t)$ uma instância do PAP e $S \subseteq V$ uma solução para I. Vamos mostrar que S é viável se, e somente se, existe um grafo dirigido $G' = (V, E')$ com $E' \subseteq \{(u, v) \mid \{u, v\} \in E\}$ que, em conjunto com S, atende às seguintes propriedades:

(i) Se $v \in S$, então não existem arcos que incidem em v;

(ii) Se $(u, v) \in E'$, então $u \in S$ ou existem ao menos $t(u)$ arcos que incidem em u;

(iii) Se $v \in V$, então $v \in S$ ou existe um arco que incide em v;

(iv) G' é acíclico.

Lema 4.3. *Sejam $I = (G = (V, E), t)$ uma instância do PAP e S uma solução para I. Se S é viável para I, então existe $G' = (V, E')$ com $E' \subseteq \{(u, v) \mid \{u, v\} \in E\}$ que, em conjunto com S, atende às propriedades* (i), (ii), (iii) *e* (iv).

Demonstração. Sejam $I = (G = (V, E), t)$ uma instância do PAP e $S \subseteq V$ uma solução para I. Vamos construir $G' = (V, E')$ com $E' \subseteq \{(u, v) \mid \{u, v\} \in E\}$ inicialmente vazio. Agora, considere a propagação do PAP a partir de S. Para cada $\{u, v\} \in E$, se existe uma rodada $\tau \geq 0$ na qual u é disseminador, mas v não é, então adicione o arco (u, v) em E'. Vamos mostrar que G' construído dessa maneira, em conjunto com S, atende às propriedades (i), (ii), (iii) e (iv).

De acordo com a definição do PAP, se v é semente, então v é disseminador desde a rodada inicial e, portanto, por construção de G', não há arco que incide em v. Logo, a propriedade (i) é atendida.

Se $(u, v) \in E'$, então $\{u, v\} \in E$ e, além disso, por construção de G', existe $\tau \geq 0$ tal que u é disseminador na rodada τ, mas v não é disseminador na rodada τ. Mas se u é disseminador na rodada τ, então temos somente duas possibilidades. A primeira é que u é

[8] Agradecemos a F. Usberti pelas profícuas discussões sobre o PAP e por sugerir uma formulação inicial baseada em modelagem por grafos.

semente e, portanto, $u \in S$. A segunda é que u se tornou disseminador em alguma rodada $\sigma \geq 1$ e, portanto, na rodada $\sigma - 1$, u não era disseminador, mas existiam ao menos $t(u)$ vértices vizinhos de u em G que são disseminadores. Nesse caso, por construção de G', sabemos que existem ao menos $t(u)$ arcos que incidem em u. Assim, chegamos que a propriedade (ii) é atendida.

Ademais, como S é viável, sabemos que ao final da propagação todos os vértices são conhecedores. Isso implica que para todo vértice $v \in V$, vale que $v \in S$ ou existe alguma rodada $\tau \geq 0$ em que algum vizinho u de v em G é disseminador mas v não o é, de modo que na rodada $\tau + 1$, v torna-se conhecedor. No segundo caso, temos que o arco (u, v) pertence a E'. Logo, a propriedade (iii) é atendida.

Por fim, suponha com o propósito de obter uma contradição, que a propriedade (iv) não é atendida. Dessa forma, existe um ciclo em G' descrito pela sequência de arcos $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)$, com $k \geq 2$. Seja v_i o primeiro vértice que se tornou disseminador, entre os vértices desse ciclo, ao longo da propagação. Isso implica que[9] v_{i-1} não pode ter sido disseminador antes de v_i, o que contraria o fato de que $(v_{i-1}, v_i) \in E'$ de acordo com a construção de G'. Portanto, a propriedade (iv) é atendida. □

Lema 4.4. *Sejam $I = (G = (V, E), t)$ uma instância do* PAP *e S uma solução para I. Se existe $G' = (V, E')$ com $E' \subseteq \{(u, v) \mid \{u, v\} \in E\}$ que, em conjunto com S, atende às propriedades* (i), (ii), (iii) *e* (iv)*, então S é viável para I.*

Demonstração. Sejam $I = (G = (V, E), t)$ uma instância do PAP e $S \subseteq V$ uma solução para I. Suponha que existe $G' = (V, E')$ com $E' \subseteq \{(u, v) \mid \{u, v\} \in E\}$ que, em conjunto com S, atende às propriedades (i), (ii), (iii) e (iv).

Como a propriedade (ii) é atendida, então o fato de que $t(u) \geq 1$ para todo vértice u implica que se $(u, v) \in E'$, então existe pelo menos um arco em E' que incide em u, a menos que $u \in S$. Além disso, como G' é acíclico (propriedade (iv)), sabemos que qualquer caminho em G' é simples.

A partir disso, podemos concluir que se um caminho C começa em algum vértice $u \notin S$ e termina num vértice v, então existe um caminho C' de comprimento ainda maior tal que C' inclui C e que começa em algum vértice $w \in S$ e termina em v. Mas G', em conjunto com S, também atende à propriedade (i) e, portanto, se C é o maior caminho que termina em um vértice v, então C começa em algum vértice pertencente a S.

Dito isto, considere a propagação do PAP a partir de S. Se $v \in S$, então v é disseminador desde a rodada inicial. Agora, tome um vértice $v \notin S$, e seja C o caminho mais longo em G' que termina em v. Vamos mostrar que se o comprimento de C é $\tau \geq 1$ e existem ao menos $t(v)$ arcos que incidem em v, então v é disseminador na rodada τ.

Seja $v \notin S$ um vértice tal que o caminho mais longo que termina em v tem comprimento $\tau = 1$ e que existem ao menos $t(v)$ arcos que incidem em v. Então, cada um dos $t(v)$ arcos tem como origem uma semente e, portanto, v é vizinho de $t(v)$ sementes em G. Logo, v é disseminador na rodada $\tau = 1$.

Assuma, por hipótese de indução, que a proposição vale para algum $\tau \geq 1$. Seja $v \notin S$ um vértice tal que o caminho mais longo C que termina em v tem comprimento

[9] Se $i = 1$, tome v_k no lugar de v_{i-1}.

$\tau + 1 \geq 2$ e que existem ao menos $t(v)$ arcos que incidem em v. Se (u, v) é um desses $t(v)$ arcos, então, pela propriedade (ii), $u \in S$ ou ao menos $t(u)$ arcos incidem em u. No primeiro caso, u é disseminador desde a rodada inicial, pois é semente. No segundo caso, sabemos que o caminho mais longo que termina em u tem comprimento menor ou igual a τ; do contrário, C não seria um caminho mais longo que termina em v. Neste caso, por hipótese de indução, u é disseminador na rodada τ. Em ambos os casos u é disseminador na rodada τ e, portanto, v possui $t(v)$ vizinhos em G que são disseminadores na rodada τ. Isso implica que v é disseminador na rodada $\tau + 1$.

Por fim, como a propriedade (iii) é atendida, para todo vértice $v \in V$ vale que $v \in S$ ou existe um arco $(u, v) \in E'$. Se $v \in S$, então v é conhecedor ao fim da propagação, pois é semente. Se $(u, v) \in E'$, então pela propriedade (ii), $u \in S$ ou existem $t(u)$ arcos que incidem em u. No primeiro caso, v é conhecedor ao fim da propagação, pois é vizinho de uma semente em G. No segundo caso, sabemos que u é disseminador na rodada τ, onde τ é o comprimento do maior caminho em G' que termina em u e, portanto, v é conhecedor na rodada $\tau + 1$. Em todos os casos, v é conhecedor. Portanto, S é viável. □

A seguir, definimos a formulação `PLI-ARCOS`. Sejam $I = (G = (V, E), t)$ uma instância do `PAP` e $G' = (V, E')$ um grafo dirigido com $E' = \{(u, v) \mid \{u, v\} \in E\}$.

Denotamos as vizinhanças de entrada e de saída de v em G' por $N_{\text{in}}(v) = \{u \in V : (u, v) \in E'\}$ e $N_{\text{out}}(v) = \{u \in V : (v, u) \in E'\}$, respectivamente. Os graus de entrada e de saída de v são denotados por $d_{\text{in}}(v)$ e $d_{\text{out}}(v)$, respectivamente. Ademais, Ξ será a notação usada para o conjunto de todos os ciclos orientados de G'.

Considere os seguintes conjuntos de variáveis binárias $C = \{s_v : v \in V\}$ e $A = \{a_{u,v} : (u, v) \in E'\}$ do modelo `PLI-ARCOS`. Mostraremos que dada uma solução do modelo `PLI-ARCOS`, é possível construir um grafo $G'' = (V, E'')$ subgrafo de G', onde $E'' = \{(u, v) \in E' \mid a_{u,v} = 1\}$, e um conjunto semente $S = \{v \in V \mid s_v = 1\}$ para I, tais que G'', em conjunto com S, atendem às propriedades (i), (ii), (iii) e (iv). A formulação `PLI-ARCOS` segue.

$$\min z = \sum_{v \in V} s_v \tag{4.8}$$

sujeito a:

$$\sum_{u \in N_{\text{in}}(v)} (a_{u,v}) - d_{\text{in}}(v)(1 - s_v) \leq 0 \qquad \forall v \in V \tag{4.9}$$

$$\sum_{u \in N_{\text{in}}(v)} (a_{u,v}) - t(v)(a_{v,w} - s_v) \geq 0 \qquad \forall (v, w) \in E' \tag{4.10}$$

$$s_v + \sum_{u \in N_{\text{in}}(v)} a_{u,v} \geq 1 \qquad \forall v \in V \tag{4.11}$$

$$\sum_{(u,v) \in \xi} a_{u,v} \leq |\xi| - 1 \qquad \forall \xi \in \Xi \tag{4.12}$$

$$s_v, a_{u,v} \in \{0, 1\} \qquad \forall v \in V \; \forall u \in V \tag{4.13}$$

A função objetivo (4.8) minimiza a quantidade de vértices que pertencerão ao conjunto

S. A restrição (4.9) determina que se $v \in S$, então não existe arco em E'' que incide em v. A restrição (4.10) garante que se $(v, w) \in E''$, então $v \in S$ ou existem ao menos $t(v)$ arcos em E'' que incidem em v. A desigualdade (4.11) estabelece que se $v \in V$, então $v \in S$ ou existe ao menos um arco em E'' que incide em v.

A família de restrições (4.12) corresponde à eliminação de subciclos orientados em G''. Observe que diferentemente das restrições anteriores, existe um número exponencial de desigualdades em (4.12). Nesse caso, ao utilizarmos algoritmos do tipo *branch and bound* com a formulação `PLI-ARCOS`, definimos as desigualdades de (4.12) como *lazy constraints*. Isso significa que no início do algoritmo, as restrições (4.12) não são consideradas pelo algoritmo, mas são adicionadas ao longo da execução da seguinte maneira: sempre que o algoritmo encontrar uma solução inteira, verifica-se se tal solução viola alguma das desigualdades em (4.12). Se sim, então a desigualdade violada é adicionada ao conjunto de desigualdades consideradas pelo algoritmo. Para checar se uma solução inteira viola alguma desigualdade em (4.12), basta executarmos uma busca em profundidade em G'' com o objetivo de encontrar ciclos orientados.

Por fim, a restrição (4.13) garante que todas as variáveis do modelo sejam binárias. A quantidade total de variáveis é da ordem de $|V| + |E|$.

A seguir, justificamos a corretude da formulação. Sejam $I = (G = (V, E), t)$ uma instância do `PAP` e $G' = (V, E')$ um grafo dirigido com $E' = \{(u, v) \mid \{u, v\} \in E\}$.

Lema 4.5. *Sejam $G'' = (V, E'')$ um subgrafo de G' e S uma solução para I. Se G'', em conjunto com S, atende às propriedades* (i), (ii), (iii) *e* (iv)*, então a função $F : C \cup A \to \{0, 1\}$ de atribuição de valores às variáveis do modelo `PLI-ARCOS` de I definida por $F(s_v) = 1 \leftrightarrow v \in S$ e $F(a_{u,v}) = 1 \leftrightarrow (u, v) \in E''$ é solução do modelo `PLI-ARCOS` de I.*

Demonstração. Assuma que G'', em conjunto com S, atende às propriedades (i), (ii), (iii) e (iv). Seja $F : C \cup A \to \{0, 1\}$ uma função de atribuição de valores às variáveis do modelo `PLI-ARCOS` de I definida por $F(s_v) = 1 \leftrightarrow v \in S$ e $F(a_{u,v}) = 1 \leftrightarrow (u, v) \in E''$. Vamos mostrar que F é solução do modelo `PLI-ARCOS` de I.

Suponha, com o propósito de obter uma contradição, que F viola a restrição (4.9). Então, existe $v \in V$ tal que $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) > d_{\text{in}}(v)(1 - F(s_v))$. Essa desigualdade só pode ser satisfeita se $F(s_v) = 1$ e $F(a_{u,v}) = 1$ para algum $u \in N_{\text{in}}(v)$. Mas como G'', em conjunto com S, atende à propriedade (i), então para qualquer $v \in S$, não existem arcos que incidem em v. Isso implica que se $F(s_v) = 1$, então $F(a_{u,v}) = 0$ para todo $u \in N_{\text{in}}(v)$, contradição.

Suponha, com o propósito de obter uma contradição, que F viola a restrição (4.10). Então, existe $(v, w) \in E'$ tal que $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) < t(v)(F(a_{v,w}) - F(s_v))$. Essa desigualdade só pode ser satisfeita se $F(a_{v,w}) = 1$ e $F(s_v) = 0$. Logo, $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) < t(v)$. Mas como G'', em conjunto com S, atende à propriedade (ii), se $(v, w) \in E''$ com $v \notin S$, então existem ao menos $t(v)$ arcos que incidem em v e, portanto, $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) \geq t(v)$, contradição.

Suponha, com o propósito de obter uma contradição, que F viola a restrição (4.11). Então, existe $v \in V$ tal que $F(s_v) + \sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) < 1$. Isso implica que $F(s_v) = 0$ e $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) = 0$. Mas, como G'', em conjunto com S, atende à propriedade (iii), vale

que $v \in S$ ou ao menos um arco incide em v, e portanto, $F(s_v) = 1$ ou $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) \geq 1$, contradição.

Por fim, como G'', em conjunto com S, atende à propriedade (iv), temos que G'' é acíclico. Isso implica que F não viola a restrição (4.12). □

Lema 4.6. *Se $F : C \cup A \to \{0,1\}$ é uma solução do modelo `PLI-ARCOS` de I, então existe $G'' = (V, E'')$ subgrafo de G' definido por $(u,v) \in E'' \leftrightarrow F(a_{u,v}) = 1$ que atende às propriedades* (i), (ii), (iii) *e* (iv)*, em conjunto com uma solução S de I definida por $v \in S \leftrightarrow F(s_v) = 1$.*

Demonstração. Assuma que $F : C \cup A \to \{0,1\}$ é uma solução do modelo `PLI-ARCOS` de I. Seja $G'' = (V, E'')$ subgrafo de G' tal que $(u,v) \in E'' \leftrightarrow F(a_{u,v}) = 1$ e seja S uma solução de I tal que $v \in S \leftrightarrow F(s_v) = 1$. Mostraremos que G'', em conjunto com S, atende às propriedades (i), (ii), (iii) e (iv).

Suponha, com o propósito de obter uma contradição, que a propriedade (i) não é atendida. Isso implica que existe $v \in S$ com algum arco incidente em v. Logo, existe v tal que $F(s_v) = 1$ e $F(a_{u,v}) = 1$ para algum $u \in N_{\text{in}}(v)$, o que contradiz o fato de que F não viola a restrição (4.9).

Suponha, com o propósito de obter uma contradição, que a propriedade (ii) não é atendida. Isso implica que existe $(v,w) \in E''$ tal que $v \notin S$ e existem no máximo $t(v) - 1$ arcos incidentes em v. Logo, temos que $F(a_{v,w}) = 1$, $F(s_v) = 0$ e $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) < t(v)$, o que contradiz o fato de que F não viola a restrição (4.10).

Suponha, com o propósito de obter uma contradição, que a propriedade (iii) não é atendida. Isso implica que existe v tal que $v \notin S$ e nenhum arco incide em v. Logo, vale que $F(s_v) = 0$ e que $\sum_{u \in N_{\text{in}}(v)} F(a_{u,v}) \geq 0$. Mas isso contradiz o fato de que F não viola a restrição (4.11).

Por fim, suponha com o propósito de obter uma contradição, que a propriedade (iv) não é atendida. Isso significa que existe ao menos um ciclo orientado no grafo G'', o que contradiz o fato de que F não viola a desigualdade (4.12). □

Teorema 4.5. *Seja S uma solução para I. S é viável se, e somente se, existe uma solução $F : C \cup A \to \{0,1\}$ do modelo `PLI-ARCOS` de I tal que $S = \{v \in V \mid F(s_v) = 1\}$.*

Demonstração. Seja S uma solução para I. Pelos Lemas 4.3 e 4.5, temos que se S é viável, então existe uma solução $F : C \cup A \to \{0,1\}$ do modelo `PLI-ARCOS` de I tal que $F(s_v) = 1 \leftrightarrow v \in S$. Por outro lado, pelos Lemas 4.4 e 4.6, temos que se $F : C \cup A \to \{0,1\}$ é solução do modelo `PLI-ARCOS` de I, então existe uma solução viável S para I tal que $v \in S \leftrightarrow F(s_v) = 1$. □

Corolário 4.5. *Se $F : C \cup A \to \{0,1\}$ é uma solução ótima do modelo `PLI-ARCOS` de I, então $S = \{v \in V \mid F(s_v) = 1\}$ é uma solução ótima para I.*

Demonstração. Seja $F : C \cup A \to \{0,1\}$ uma solução ótima do modelo `PLI-RODADAS` de I. Tome $S = \{v \in V : F(s_v) = 1\}$. O Teorema 4.5 garante que S é solução viável de I.

Suponha, com o propósito de obter uma contradição, que S não é ótima para I. Então existe uma solução viável S' de I tal que $|S'| < |S|$. Pelo Teorema 4.5, existe uma solução $F' : C \cup A \to \{0,1\}$ do modelo `PLI-ARCOS` de I tal que $F'(s_v) = 1 \leftrightarrow v \in S'$. Mas

$\sum_{v \in V} F'(s_v) = |S'| < |S| = \sum_{v \in V} F(s_v)$ e isso contradiz o fato de que F é solução ótima do modelo PLI-ARCOS de I. Portanto, S é uma solução ótima para I. □

4.3 Heurísticas Baseadas em GRASP

Nesta seção, apresentamos quatro heurísticas para o PAP, as quais são baseadas na metaheurística GRASP. Primeiramente, descrevemos uma rotina que simula o processo de propagação e que é utilizada tanto na fase de construção quanto na de busca local de cada uma das heurísticas. Em seguida, introduzimos as heurísticas descrevendo suas respectivas fases de construção. As abordagens empregadas nessa fase são o que principalmente diferencia essas heurísticas. Por fim, apresentamos o procedimento de busca local, que se mostra semelhante entre as heurísticas, com exceção de alguns critérios.

Considere uma instância $I = (G = (V, E), t)$ do PAP. O mecanismo de simulação utilizado nas heurísticas considera, para cada vértice $v \in V$, dois atributos:

- $estado(v)$, que indica o atual estado de v durante o processo de propagação (ignorante, conhecedor ou disseminador);
- $n_d(v)$, que mantém o número de vizinhos de v que são disseminadores.

O Algoritmo 4.1 descreve a rotina de simulação ao longo da qual mantemos uma fila Q com os vértices que tornaram-se disseminadores, mas que ainda não propagaram a informação para os seus vizinhos. O laço mais externo é repetido até que Q torne-se vazio, isto é, até que todos os vértices disseminadores tenham de fato propagado a informação para as suas respectivas vizinhanças.

Algoritmo 4.1: Processo de propagação.

```
   Entrada: Fila de disseminadores Q
1  enquanto Q não está vazia faça
2      u ← Desenfileira(Q)
3      para cada v ∈ N(u) faça
4          n_d(v) ← n_d(v) + 1
5          se estado(v) = ignorante então
6              estado(v) ← conhecedor
7          se estado(v) ≠ disseminador ∧ n_d(v) ≥ t(v) então
8              estado(v) ← disseminador
9              Enfileira(Q, v)
```

Se u é o próximo vértice da fila Q, então para cada $v \in N(u)$ incrementamos $n_d(v)$ em 1. Qualquer $v \in N(u)$ que se torna conhecedor ou disseminador pelo incremento de $n_d(v)$ tem o seu estado alterado nas linhas 6 e 8, respectivamente. Quando v se torna disseminador, ele é imediatamente inserido em Q.

Para simular o processo de propagação desde a sua origem a partir de um conjunto semente S até o fim da propagação, usamos o Algoritmo 4.2. Ele inicializa todos os vértices

como ignorantes, exceto aqueles em S, insere as sementes na fila de disseminadores Q, e em seguida chama o Algoritmo 4.1 para dar início à propagação, que é conduzida até seu estado final.

Algoritmo 4.2: Propagação completa.

Entrada: Instância $(G = (V, E), t)$; conjunto semente S

```
1 Q ← fila vazia
2 para cada v ∈ V faça
3 |  n_d(v) ← 0
4 |  se v ∈ S então
5 |  |  estado(v) ← disseminador
6 |  |  Enfileira(Q, v)
7 |  senão
8 |  |  estado(v) ← ignorante
9 ProcessoDePropagação(Q)
```

É evidente que ao final do Algoritmo 4.2, o estado de cada vértice é igual ao estado final que ele teria ao fim de uma propagação do `PAP` iniciada por S. Observe que ao longo do processo de propagação, cada aresta só é visitada quando algum de seus extremos se torna disseminador, ou seja, cada aresta é visitada não mais do que duas vezes ao longo da propagação. Portanto, o Algoritmo 4.2 tem complexidade de tempo $\mathcal{O}(|V| + |E|)$. Ao final do Algoritmo 4.2, podemos verificar a viabilidade de S simplesmente contando o número de vértices conhecedores.

Na fase de construção do `GRASP`, construímos uma solução S (inicialmente vazia) de maneira incremental, até que S torna-se viável. Uma maneira de proceder com essa fase seria, toda vez que adicionamos um vértice em S, executar o Algoritmo 4.2 a fim de verificar a viabilidade de S. Isso implicaria executar o processo de propagação para cada semente inserida em S. Todavia, mostraremos a seguir que é possível conduzir a fase de construção com apenas uma execução do processo de propagação.

A ideia consiste em salvar o estado final da propagação a partir de S e, ao adicionarmos uma nova semente u, continuamos a propagação em decorrência da disseminação gerada por u. O Teorema 4.6 garante que este procedimento é válido, isto é, que o estado final da propagação a partir do conjunto semente $S \cup \{u\}$ é o mesmo que o estado final da propagação a partir do conjunto semente $\mathrm{Dis}_G[S] \cup \{u\}$. Denotamos por S^* o conjunto de disseminadores no final de uma propagação iniciada com S, isto é, $S^* = \mathrm{Dis}_G[S]$.

Teorema 4.6. *Sejam $S \subseteq V$ e $u \in V$. Então $(S \cup \{u\})^* = (S^* \cup \{u\})^*$.*

Demonstração. Inicialmente, vamos mostrar que $(S \cup \{u\})^* \subseteq (S^* \cup \{u\})^*$. Suponha com o propósito de obter uma contradição que $(S \cup \{u\})^* \setminus (S^* \cup \{u\})^* \neq \emptyset$. Note que $\mathrm{Dis}_G[S \cup \{u\}, 0] = S \cup \{u\} \subseteq S^* \cup \{u\} \subseteq (S^* \cup \{u\})^*$. Logo, existe $\tau \geq 1$ para o qual $\mathrm{Dis}_G[S \cup \{u\}, \tau - 1] \subseteq (S^* \cup \{u\})^*$, mas $\mathrm{Dis}_G[S \cup \{u\}, \tau] \setminus (S^* \cup \{u\})^* \neq \emptyset$. Tome $v \in \mathrm{Dis}_G[S \cup \{u\}, \tau] \setminus (S^* \cup \{u\})^*$. Como $v \in \mathrm{Dis}_G[S \cup \{u\}, \tau]$, mas $v \notin \mathrm{Dis}_G[S \cup \{u\}, \tau - 1]$, temos que $t(v) \leq |N(v) \cap \mathrm{Dis}_G[S \cup \{u\}, \tau - 1]|$. Porém, pela escolha de τ, sabemos que

$|N(v) \cap \text{Dis}_G[S \cup \{u\}, \tau - 1]| \leq |N(v) \cap (S^* \cup \{u\})^*|$. Ou seja, se o conjunto semente for $S^* \cup \{u\}$, então v deve se tornar disseminador, contrariando a escolha de v.

Agora, vamos mostrar que $(S^* \cup \{u\})^* \subseteq (S \cup \{u\})^*$. Suponha com o propósito de obter uma contradição que $(S^* \cup \{u\})^* \setminus (S \cup \{u\})^* \neq \emptyset$. Note que $\text{Dis}_G[S^* \cup \{u\}, 0] = S^* \cup \{u\} \subseteq (S \cup \{u\})^*$. Logo, existe $\tau \geq 1$ para o qual $\text{Dis}_G[S^* \cup \{u\}, \tau - 1] \subseteq (S \cup \{u\})^*$, mas $\text{Dis}_G[S^* \cup \{u\}, \tau] \setminus (S \cup \{u\})^* \neq \emptyset$. Tome $v \in \text{Dis}_G[S^* \cup \{u\}, \tau] \setminus (S \cup \{u\})^*$. Como $v \in \text{Dis}_G[S^* \cup \{u\}, \tau]$, mas $v \notin \text{Dis}_G[S^* \cup \{u\}, \tau - 1]$ temos que $t(v) \leq |N(v) \cap \text{Dis}_G[S^* \cup \{u\}, \tau - 1]|$. Porém, pela escolha de τ, $|N(v) \cap \text{Dis}_G[S^* \cup \{u\}, \tau - 1]| \leq |N(v) \cap (S \cup \{u\})^*|$. Ou seja, se o conjunto semente for $S \cup \{u\}$, então v deve se tornar um disseminador, contrariando a escolha de v. □

A seguir, apresentamos as quatro heurísticas denominadas *Greedy Randomized*, *Weighted Greedy Randomized*, *Random plus Greedy* e *Sampled Greedy*. Primeiramente, descreveremos as estratégias empregadas nas fases de construção de cada heurística.

No início da fase de construção, consideramos que todos os vértices são ignorantes e que $n_d(v) = 0$ para cada $v \in V$. Além disso, denotamos por $d_{\max}$ o maior grau em G. Nas próximas seções, considere que G é conexo, pois de acordo com o que foi exposto na Seção 4.1, se G não é conexo, então podemos obter uma solução viável para a instância original ao resolver o `PAP` para cada componente conexa separadamente.

4.3.1 Greedy Randomized

A estratégia de construção da heurística Greedy Randomized (`GR`) é baseada na estratégia padrão de construção do `GRASP` (referimos o leitor à Seção 2.2). Denote por $b(v)$ o benefício de se inserir um vértice v da CL no conjunto semente em construção e por $n_i(v)$ o número de vizinhos ignorantes de v. Para esta estratégia, simplesmente definimos $b(v) = n_i(v)$, pois $n_i(v)$ representa a quantidade de vértices que se tornarão imediatamente conhecedores pela adição de v.

Seja S um conjunto semente inicialmente vazio. Em cada passo da construção, tomamos $\text{CL} = \{v \in V \mid b(v) \geq 1\}$ e $\text{RCL} = \{v \in \text{CL} \mid b(v) \geq b_{\max} - \lfloor \alpha \, (b_{\max} - b_{\min}) \rfloor\}$, para um certo $\alpha \in [0, 1]$, onde $b_{\min}$ e $b_{\max}$ são os benefícios mínimo e máximo, respectivamente, entre todos os vértices da CL. Nesse caso, α controla a gula da construção, de forma que quanto menor o valor de α, mais gananciosa é a composição da RCL. Em seguida, um elemento aleatório da RCL é escolhido para ser inserido em S. Esse processo é repetido até que S seja viável.

Considere a representação de V como um vetor. Mantemos V ordenado, em relação aos benefícios dos vértices, ao longo da execução, a fim de obter a CL e a RCL diretamente de V. Denotamos por $V[i \ldots j]$ o subvetor de V que começa no índice i e termina no índice j. Ademais, denotamos por $pos(v)$ a posição que o vértice v ocupa em V.

Mantemos um vetor auxiliar *map* que, em conjunto com V, satisfaz as seguintes propriedades durante toda a fase de construção: se $map[i] = j \geq 0$, então $V[0 \ldots j]$ contém todos os vértices de V com benefício maior ou igual a i; se $map[i] = -1$, então não existe vértice em V com benefício maior ou igual a i.

O Algoritmo 4.3 descreve a fase de construção da heurística `GR`. Nas linhas 1 e 2, tomamos uma fila de disseminadores Q e um conjunto semente S representado por um

vetor, ambos vazios. O custo, em termos de tempo de execução, associado a essas linhas é constante. No laço da linha 3, inicializamos o benefício de cada vértice com o valor de seu grau, uma vez que inicialmente todos os vértices são ignorantes. O tempo para execução do laço da linha 3 é $\Theta(|V|)$.

Algoritmo 4.3: Fase de construção da heurística Greedy Randomized.

Entrada: Instância $(G = (V, E), t)$; parâmetro α
Saída : Conjunto semente perfeito S

```
1  Q ← fila vazia
2  S ← vetor vazio
3  para i = 0 até |V| − 1 faça
4      b(V[i]) ← d(V[i])
5  OrdenaNãoCrescentePorBenefício(V)
6  map ← vetor de inteiros com tamanho d_max + 1
7  i ← 0
8  para j = d_max até 0 faça
9      enquanto i ≤ |V| − 1 ∧ b(V[i]) ≥ j faça
10         i ← i + 1
11     map[j] ← i − 1
12 enquanto S não é viável faça
13     b_max ← b(V[0])
14     b_min ← b(V[map[1]])                          ▷ CL = V[0 ... map[1]]
15     k ← b_max − ⌊α (b_max − b_min)⌋
16     v ← SorteiaElementoAleatório(V[0 ... map[k]])  ▷ RCL = V[0 ... map[k]]
17     estado(v) ← disseminador
18     InsereNoFinal(S, v)
19     Enfileira(Q, v)
20     enquanto Q não está vazia faça
21         u ← Desenfileira(Q)
22         para cada v ∈ N(u) faça
23             n_d(v) ← n_d(v) + 1
24             se estado(v) = ignorante então
25                 estado(v) ← conhecedor
26                 para cada w ∈ N(v) faça
27                     Troca(V[pos(w)], V[map[b(w)]])
28                     map[b(w)] ← map[b(w)] − 1
29                     b(w) ← b(w) − 1
30             se estado(v) ≠ disseminador ∧ n_d(v) ≥ t(v) então
31                 estado(v) ← disseminador
32                 Enfileira(Q, v)
33 retorna S
```

Na linha 5, ordenamos V de maneira não crescente em relação aos benefícios dos vértices. Observe que, como $1 \leq b(v) \leq d_{\max}$ para todo v em V e $d_{\max} \in \mathcal{O}(|V|)$, podemos realizar essa ordenação com o algoritmo *Counting Sort* em tempo $\mathcal{O}(|V|)$.

Ademais, nas linhas 6 a 11, tomamos um vetor map e o inicializamos de modo que esse, em conjunto com V, atenda às propriedades descritas anteriormente nesta seção. Esse procedimento custa $\Theta(|V|)$.

A construção de S ocorre no laço da linha 12. Nas linhas 13 a 16, realizamos o procedimento de seleção de uma nova semente. Primeiramente, note que os elementos da CL são os elementos pertencentes ao subvetor $V[0 \dots map[1]]$, uma vez que este subvetor contém todos os vértices de V com benefício maior ou igual a 1. Portanto, não se faz necessária uma estrutura de dados adicional para representar a CL, basta que consideremos $V[0 \dots map[1]]$ como a CL. Nesse caso, $b_{\max}$ é igual ao benefício do vértice mais à esquerda em V e $b_{\min}$ é igual ao benefício do vértice cuja posição em V é $map[1]$.

Em seguida tomamos $k = b_{\max} - \lfloor \alpha \, (b_{\max} - b_{\min}) \rfloor$. Note que os elementos da RCL são os elementos pertencentes ao subvetor $V[0 \dots map[k]]$, uma vez que este subvetor contém todos os vértices de V com benefício maior ou igual a k. Portanto, não se faz necessária uma estrutura de dados adicional para representar a RCL, basta que consideremos $V[0 \dots map[k]]$ como a RCL. Assim, o passo seguinte consiste em escolher aleatoriamente uma nova semente em $V[0 \dots map[k]]$. Todos os passos das linhas 13 a 16 podem ser feitos em tempo constante[10]. Nas linhas 17 a 19, alteramos o estado da nova semente e a inserimos em S e na fila de disseminadores Q. O próximo passo é dar continuidade ao processo de propagação em virtude da nova semente.

No laço da linha 20, simulamos a propagação por meio de uma sequência de passos idêntica ao Algoritmo 4.1, exceto pelo laço da linha 26, o qual explicamos a seguir. Como v se tornou conhecedor por disseminação proveniente de u (linha 25), precisamos decrementar em 1 o benefício de cada $w \in N(v)$. Todavia, a fim de manter a ordenação de V e as propriedades do vetor map em conjunto com V, fazemos o que segue. Para cada $w \in N(v)$, trocamos as posições, em V, dos dois elementos $V[pos(w)]$ (o próprio w) e $V[map[b(w)]]$ (o vértice de V mais à direita que possui benefício maior ou igual a $b(w)$). Em seguida, decrementamos o valor de $map[b(w)]$ e de $b(w)$ em 1, nessa ordem. O passos internos ao laço da linha 26 levam tempo constante.

Devido ao processo de propagação e atualização de benefícios dos vértices, cada aresta do grafo é visitada no mínimo uma e no máximo quatro vezes, ou seja, somente quando um de seus extremos torna-se disseminador ou conhecedor. O custo total associado a essa visitação é $\Theta(|V| + |E|)$. Quando uma aresta é visitada, algum de seus extremos pode ter o seu benefício atualizado (isto é, decrementado em 1). Todavia, essa atualização e os ajustes dos vetores V e map decorrentes dela custam $\mathcal{O}(1)$. Portanto, o Algoritmo 4.3 é executado em tempo $\Theta(|V| + |E|)$.

4.3.2 Weighted Greedy Randomized

A estratégia de construção da heurística Weighted Greedy Randomized (`WGR`) difere da abordagem de construção da heurística `GR` em apenas um aspecto. Ao selecionarmos um vértice da RCL, estabelecemos que a chance de v ser escolhido é proporcional ao valor de $|\{u \in N(v) \,|\, (u \text{ não é semente }) \wedge (n_d(u) = t(u) - 1)\}|$, ou seja, o número de vizinhos

[10]Para que a linha 16 seja executada em tempo constante (amortizado), pode-se utilizar o gerador de números pseudo-aletórios Mersenne Twister [41].

de v que são *quase-disseminadores* e que imediatamente se tornarão disseminadores se adicionarmos v ao conjunto semente. Denotamos esse número por $n_{qd}(v)$, os vizinhos quase-disseminadores de v. O contador n_{qd} pode ser mantido para todos os vértices ao longo da propagação de forma semelhante ao contador n_i, da heurística anterior.

Na fase de construção da `GR`, quando selecionamos uma nova semente, a chance de um vértice ser escolhido é a mesma dos demais vértices e, portanto, o sorteio pode ser feito em tempo constante. Já na fase de construção da `WGR`, calculamos a chance de cada vértice v ser selecionado antes do sorteio e, logo após, selecionamos aleatoriamente uma nova semente da RCL de acordo com as probabilidades calculadas. Nesse caso, o custo de tempo relacionado ao sorteio de uma semente é $\mathcal{O}(|\text{RCL}|)$. Como $|\text{RCL}| \in \mathcal{O}(|V|)$ e não mais que $|V|$ sementes são escolhidas, o custo total associado aos sorteios na fase de construção da `WGR` é $\mathcal{O}(|V|^2)$.

Assim, podemos implementar a fase de construção da `WGR` com complexidade de tempo $\mathcal{O}(|V|^2) \cap \Omega(|V| + |E|)$, de maneira análoga à implementação da fase de construção da `GR`.

4.3.3 Random plus Greedy

No método de construção da heurística Random plus Greedy (`RG`), a RCL coincide com a CL. Por essa razão, nos referiremos somente à RCL ao longo desta seção. Inicialmente, a RCL contém todos os vértices. Para um dado inteiro p, tomamos uma nova semente simplesmente escolhendo um elemento aleatório da RCL nas primeiras p iterações, e escolhendo o elemento da RCL com maior benefício nas iterações restantes.

Neste método, também fazemos $b(v) = n_i(v)$. Porém, se ocorrer empate no valor do benefício, damos uma prioridade mais alta ao vértice v com valor mais alto de $n_{qd}(v)$ ou, se o empate persistir, com valor mais alto de $t(v) - n_d(v)$. Note que $t(v) - n_d(v)$ representa a quantidade de vizinhos disseminadores de v que falta para que v se torne disseminador. Se ainda assim tivermos empate, escolhemos o vértice cujo identificador a ele associado possui o menor valor.

Considere a representação de V como um vetor. O Algoritmo 4.4 mostra o procedimento de construção da `RG`. Na linha 1, tomamos uma fila vazia Q, um conjunto semente vazio S representado por um vetor, e inicializamos o vetor RCL com os mesmos elementos de V. A construção de S ocorre no laço da linha 2. A variável p, dada na entrada, é decrementada em 1 logo antes do final de cada iteração (linha 36). Em cada iteração temos dois casos: $p > 0$ ou $p \leq 0$.

Enquanto $p > 0$ (linha 3), selecionamos aleatoriamente um vértice v da RCL como nova semente (linha 4). Em seguida, alteramos o seu estado e o adicionamos em S (linha 5). Logo após, inserimos v em Q (linha 6) e continuamos a propagação a partir de Q (linha 7) utilizando uma modificação do Algoritmo 4.1 em que sempre que um vértice é desenfileirado de Q, ele também é removido da RCL. Cada remoção é feita em tempo constante, trocando o vértice a ser removido com o último elemento da RCL e removendo a última posição do vetor. Os passos feitos nas linhas 4 a 6 têm custo constante.

Quando $p \leq 0$ (linha 8), a seleção da nova semente é feita de maneira puramente gulosa. Se $p = 0$, inicializamos os benefícios dos vértices e demais contadores (linha 10) e em seguida transformamos o vetor RCL em uma fila de prioridades (linha 11), mais

Algoritmo 4.4: Fase de construção da heurística Random plus Greedy.

Entrada: Instância $(G = (V, E), t)$; parâmetro p
Saída : Conjunto semente perfeito S

```
1  Q ← fila vazia; S ← vetor vazio; RCL ← V
2  enquanto S não é viável faça
3      se p > 0 então
4          v ← SorteiaElementoAleatório(RCL)
5          estado(v) ← disseminador; InsereNoFinal(S, v)
6          Enfileira(Q, v)
7          ProcessoDePropagacaoMod(Q)   ▷ Disseminadores são removidos da RCL
8      senão
9          se p = 0 então
10             InicializaContadores(b, n_d, n_qd)
11             RCL ← ConstróiFilaDePrioridade(RCL)
12         v ← ObtémElementoComMaiorPrioridade(RCL)
13         estado(v) ← disseminador; InsereNoFinal(S, v)
14         se n_d(v) = t(v) − 1 então
15             para cada u ∈ N(v) faça
16                 n_qd(u) ← n_qd(u) − 1; AjustaFilaDePrioridade(RCL, u)
17         Enfileira(Q, v)
18         enquanto Q não está vazia faça
19             u ← Desenfileira(Q)
20             Remove(RCL, u)
21             para cada v ∈ N(u) faça
22                 n_d(v) ← n_d(v) + 1
23                 AjustaFilaDePrioridade(RCL, v)
24                 se n_d(v) = t(v) − 1 então
25                     para cada w ∈ N(v) faça
26                         v_qd(w) ← v_qd(w) + 1; AjustaFilaDePrioridade(RCL, w)
27                 se estado(v) = ignorante então
28                     estado(v) ← conhecedor
29                     para cada w ∈ N(v) faça
30                         b(w) ← b(w) − 1; AjustaFilaDePrioridade(RCL, w)
31                 se estado(v) ≠ disseminador ∧ n_d(v) ≥ t(v) então
32                     estado(v) ← disseminador
33                     para cada w ∈ N(v) faça
34                         n_qd(w) ← n_qd(w) − 1; AjustaFilaDePrioridade(RCL, w)
35                     Enfileira(Q, v)
36     p ← p − 1
37 retorna S
```

especificamente um *heap*. As inicializações dos contadores custam, ao todo, $\Theta(|V| + |E|)$, pois para cada vértice verificam-se os estados de seus vizinhos e atualizam-se os contadores

de acordo com esses estados. A transformação do vetor RCL em um *heap* custa $\mathcal{O}(|V|)$.

No primeiro passo da seleção gulosa, tomamos o próximo elemento v da RCL (linha 12) como uma nova semente, em tempo constante, e, em seguida, alteramos o seu estado e o adicionamos em S (linha 13). Observe que se $n_d(v) = t(v)-1$ (linha 14), então é necessário decrementar o contador n_{qd} em 1 (linha 16) de cada um dos vizinhos de v, visto que v passa a se tornar disseminador e, portanto, deixará de ser um vértice quase-disseminador. Como o contador n_{qd} é critério de desempate na fila de prioridade, após o decremento, também ajustamos a fila (linha 16) em tempo $\mathcal{O}(\log |V|)$ para cada decremento.

O próximo passo consiste em inserir v na fila de propagação Q (linha 17) e dar continuidade ao processo de propagação a partir de Q. Isso é feito no laço da linha 18, cujos passos são idênticos aos do Algoritmo 4.1, exceto pelas linhas 20, 23 e 24 e pelos laços das linhas 29 e 33. Na linha 20, removemos o disseminador u da RCL em tempo $\mathcal{O}(\log(|V|))$. Na linha 23, ajustamos a fila de prioridade, em tempo $\mathcal{O}(\log(|V|))$, em decorrência do incremento no contador $n_d(v)$ feito na linha imediatamente anterior.

Na linha 24, se a condição for verdadeira, então temos o caso em que v se tornou quase-disseminador, isto é, $n_d(v) = t(v) - 1$, e portanto, precisamos incrementar em 1 o contador n_{qd} de todos os vizinhos de v. No laço da linha 29, decrementamos o benefício de cada vizinho de v, em decorrência de v ter se tonado conhecedor. Por fim, no laço da linha 33, decrementamos o contador n_{qd} de cada vizinho de v, em decorrência de v ter deixado de ser quase-disseminador, pois tornou-se, de fato, disseminador. Para cada um desses incrementos e decrementos, ajustamos a fila de prioridade em $\mathcal{O}(\log(|V|))$.

Devido ao processo de propagação realizado ao longo da construção, cada aresta do grafo é visitada no mínimo uma e no máximo oito vezes. Isso ocorre somente quando um de seus extremos torna-se quase-disseminador, disseminador ou conhecedor ou deixa de ser um quase-disseminador. O custo total associado a essa visitação é $\Theta(|V|+|E|)$. Por outro lado, sempre que uma aresta é visitada, um de seus extremos pode ter o valor do benefício ou de algum dos contadores n_d ou n_{qd} alterado. Cada alteração implica em um ajuste na RCL com custo $\mathcal{O}(\log(|V|))$ e, portanto, o custo total desses ajustes é $\mathcal{O}(|E| \log |V|)$. Além disso, no pior dos casos, todos os vértices tornam-se disseminadores e o custo total da remoção desses vértices da RCL é $\mathcal{O}(|V| \log |V|)$. Dessa maneira, chegamos que o custo total do Algoritmo 4.4 é $\mathcal{O}(|E| \log |V|) \cap \Omega(|E|)$.

4.3.4 Sampled Greedy

Na estratégia de construção da heurística Sampled Greedy (`SG`) temos que, em cada iteração, a RCL é um subconjunto aleatório de tamanho ℓ da CL. Se |CL| $< \ell$, então RCL = CL. Em cada passo, simplesmente escolhemos o elemento da RCL com maior benefício $b(v) = n_i(v)$ para ser inserido no conjunto semente, de acordo com as mesmas regras de desempate da fase de construção da heurística `RG`.

Considere a representação de V como um vetor. O Algoritmo 4.5 mostra o procedimento de construção da `SG`. Na linha 1, tomamos uma fila vazia Q, um conjunto semente vazio S representado por um vetor, e inicializamos o vetor CL com os mesmos elementos de V. Logo após, inicializamos os benefícios dos vértices e demais contadores (linha 2). As inicializações dos benefícios e do contador n_d são feitas em $\mathcal{O}(|V|)$, de modo que para

todo v pertencente à CL, $b(v) = d(v)$ e $n_d(v) = 0$. Por outro lado, inicializamos, em $\mathcal{O}(|E|)$, os contadores n_{qd} de modo que $n_{qd}(v) = |\{u \in N(v) \mid t(u) = 1\}|$.

A construção de S ocorre no laço da linha 3. Primeiramente, tomamos a RCL como um subconjunto aleatório da CL de tamanho $\min\{|\text{CL}|, \ell\}$ (linha 4). Em seguida, varremos a RCL e obtemos um vértice v que maximiza o benefício (linha 5), alteramos o seu estado e o adicionamos em S (linhas 6 e 7). Observe que se $n_d(v) = t(v) - 1$ (linha 8), então é necessário decrementar o contador n_{qd} em 1 (linha 10) de cada um dos vizinhos de v, visto que v passa a se tornar disseminador e, portanto, deixará de ser um vértice quase-disseminador.

Algoritmo 4.5: Fase de construção da heurística Sampled Greedy.

Entrada: Instância $(G = (V, E), t)$; parâmetro ℓ
Saída : Conjunto semente perfeito S

```
1  Q ← fila vazia; S ← vetor vazio; CL ← V
2  InicializaContadores(b, n_d, n_qd)
3  enquanto S não é viável faça
4      RCL ← ObtémSubconjuntoAleatório(CL, ℓ)
5      v ← ObtémMelhorElemento(RCL)
6      estado(v) ← disseminador
7      InsereNoFinal(S, v)
8      se n_d(v) = t(v) − 1 então
9          para cada u ∈ N(v) faça
10             n_qd(u) ← n_qd(u) − 1
11     Enfileira(Q, v)
12     enquanto Q não está vazia faça
13         u ← Desenfileira(Q)
14         Remove(CL, u)
15         para cada v ∈ N(u) faça
16             n_d(v) ← n_d(v) + 1
17             se n_d(v) = t(v) − 1 então
18                 para cada w ∈ N(v) faça
19                     v_qd(w) ← v_qd(w) + 1
20             se estado(v) = ignorante então
21                 estado(v) ← conhecedor
22                 para cada w ∈ N(v) faça
23                     b(w) ← b(w) − 1
24             se estado(v) ≠ disseminador ∧ n_d(v) ≥ t(v) então
25                 estado(v) ← disseminador
26                 para cada w ∈ N(v) faça
27                     n_qd(w) ← n_qd(w) − 1
28                 Enfileira(Q, v)
29 retorna S
```

O próximo passo consiste em inserir v na fila de propagação Q (linha 11) e dar conti-

nuidade ao processo de propagação a partir de Q. Isso é feito no laço da linha 12, cujos passos são idênticos aos do Algoritmo 4.1, exceto pelas linhas 14 e 17 e pelos laços das linhas 22 e 26. Na linha 14, removemos o disseminador u da RCL, em tempo constante, trocando-se o vértice a ser removido com o último elemento da RCL e removendo a última posição do vetor.

Na linha 17, temos o caso em que v se tornou quase-disseminador, isto é, $n_d(v) = t(v) - 1$, e, portanto, precisamos incrementar em 1 o contador n_{qd} de todos os vizinhos de v. No laço da linha 22, decrementamos o benefício de cada vizinho de v, em decorrência de v ter se tornado conhecedor. Por fim, no laço da linha 26, decrementamos o contador n_{qd} de cada vizinho de v, em decorrência de v ter deixado de ser quase-disseminador, pois tornou-se, de fato, disseminador.

Devido ao processo de propagação realizado ao longo da construção, cada aresta do grafo é visitada no mínimo uma e no máximo oito vezes. Isso ocorre somente quando um de seus extremos torna-se quase-disseminador, disseminador ou conhecedor ou deixa de ser um quase-disseminador. O custo total associado a essa visitação é $\Theta(|E|)$. Por outro lado, sempre que uma aresta é visitada, um de seus extremos pode ter o valor do benefício ou de algum dos contadores n_d ou n_{qd} alterado. Cada alteração tem um custo constante associado. Além disso, no pior dos casos, todos os vértices tornam-se disseminadores e o custo total da remoção desses vértices da RCL é $\mathcal{O}(|V|)$.

Ademais, temos que em cada iteração o custo de construção da RCL (linha 4) e seleção de uma nova semente proveniente da RCL (linha 5) é $\mathcal{O}(|\text{CL}|)$, mas $|\text{CL}| \in \mathcal{O}(|V|)$. Como não mais que $|V|$ sementes são selecionadas, então o custo total associado à escolha de sementes é $\mathcal{O}(|V|^2)$. Dessa maneira, chegamos que o custo total do Algoritmo 4.5 é $\mathcal{O}(|V|^2) \cap \Omega(|E|)$.

4.3.5 Fase de Busca Local

Nesta seção, descrevemos uma abordagem para a fase de busca local que é comum às nossas quatro heurísticas, a não ser por alguns detalhes que veremos mais à frente. Como mencionamos na Seção 2.2, para projetar uma busca local para um dado problema é necessário definir, para uma solução viável S, o que consideramos como solução vizinha de S. No nosso caso, consideraremos que S' é vizinha de S se $S' \subset S$, $S' \neq \emptyset$ e S' é viável. Dessa forma, ao longo da busca local, avaliamos soluções vizinhas obtidas da remoção de uma ou mais sementes da solução corrente.

Representaremos um conjunto semente S como um vetor, onde $S[i \ldots j]$ denota o subvetor de S que começa no índice i e termina no índice j. Considere também a representação de V como um vetor. Além disso, dados dois vetores A e B, $A +\!\!+ B$ denota a concatenação de A e B.

De modo geral, nosso procedimento de busca local possui três etapas distintas conforme descrevemos a seguir.

Etapa 1 O objetivo da primeira etapa consiste em remover do conjunto semente qualquer vértice que, caso não fosse semente, se tornaria disseminador diretamente da interseção de sua vizinhança com o conjunto semente. Em resumo, visitamos cada semente, a

partir do elemento mais à esquerda de S, e removemos de S todo vértice que tenha um número de sementes vizinhas maior ou igual ao seu valor limiar. Claramente, esses vértices se tornarão disseminadores ao longo da propagação a partir das sementes remanescentes e, portanto, não precisam permanecer como sementes.

Denote por $n_s(v)$ o número de vizinhos do vértice v que são sementes. O primeiro passo da fase de busca local é descrito pelo Algoritmo 4.6. Nas linhas 1 a 5, inicializamos o contador n_s para todos os vértices de S, em tempo $\mathcal{O}(|V|)$. Em seguida, tomamos um conjunto semente S' inicialmente vazio (linha 6).

Algoritmo 4.6: Fase de busca local - Etapa 1.

```
   Entrada: Instância (G = (V, E), t); conjunto semente perfeito S
   Saída  : Conjunto semente perfeito reduzido
 1 para i = 0 até |S| − 1 faça          ▷ Inicializa os contadores n_s com valor 0
 2 |  n_s(S[i]) ← 0
 3 para i = 0 até |S| − 1 faça                    ▷ Inicializa os contadores n_s
 4 |  para cada u ∈ N(S[i]) ∩ S faça
 5 |  |  n_s(u) ← n_s(u) + 1
 6 S' ← vetor vazio
 7 para i = 0 até |S| − 1 faça                    ▷ Copia as sementes necessárias
 8 |  se n_s(S[i]) < t(S[i]) então
 9 |  |  InsereNoFinal(S', S[i])
10 |  senão
11 |  |  para cada u ∈ N(S[i]) faça               ▷ Atualiza os contadores n_s
12 |  |  |  n_s(u) ← n_s(S[i]) − 1
13 retorna S'
```

No laço da linha 7, varremos o vetor S e inserimos no final de S' (linha 9) todo vértice v de S que satisfaz $n_s(v) < t(v)$. Se $n_s(v) \geq t(v)$ (linha 10), então v não precisa permanecer como uma semente e, portanto, não copiamos v para S'. Nesse caso, como v deixa de ser semente, decrementamos $n_s(u)$ para todo $u \in N(v)$ através do laço da linha 11. O tempo de execução total associado ao laço da linha 7 é $\mathcal{O}(|V| + |E|)$. Ao final do algoritmo, retornamos o conjunto semente perfeito reduzido S'. Dessa forma, o Algoritmo 4.6 pode ser executado em tempo $\mathcal{O}(|E|)$.

Nas próximas etapas, a ideia principal consiste em, dado um conjunto semente S, tomar $S' \subset S$ e simular a propagação a partir de $S \setminus S'$. Se $S \setminus S'$ for viável, então todos os vértices de S que pertencem a S' não precisam permanecer como sementes. De forma semelhante, são dispensáveis quaisquer vértices de S que pertencem a S' e que se tornem disseminadores ao longo da propagação a partir de $S \setminus S'$.

Etapa 2 O Algoritmo 4.7 apresenta o procedimento da segunda etapa da busca local. Na linha 1, ordenamos o vetor S obtido da primeira etapa para que as sementes consideradas mais propícias para remoção sejam aquelas mais à esquerda em S. O objetivo desta etapa é reduzir o conjunto semente, removendo vértices da primeira metade do vetor S.

Algoritmo 4.7: Fase de busca local - Etapa 2.

Entrada: Instância $(G = (V, E), t)$; conjunto semente perfeito S
Saída : Conjunto semente perfeito reduzido

```
1  Ordena(S)
2  m ← |S|/2; j ← 0; PropagaçãoCompleta(G, S[m ... |S| − 1])
3  se S[m ... |S| − 1] é viável então
4  |  S ← S[m ... |S| − 1]; retorna S
5  senão
6  |  S' ← vetor vazio
7  |  para i = 0 até m − 1 faça                    ▷ Copia vértices não disseminadores
8  |  |  se estado(S[i]) ≠ disseminador então
9  |  |  |  InsereNoFinal(S', S[i])
10 |  |  senão
11 |  |  |  j ← j + 1
12 |  S' ← S' + S[m ... |S| − 1]
13 |  S ← S'
14 m ← (m − j)/2; j ← 0
15 enquanto m > 0 faça                  ▷ Remove sementes da primeira metade de S
16 |  PropagaçãoCompleta(G, S[m ... |S| − 1])
17 |  se S[m ... |S| − 1] é viável então
18 |  |  S ← S[m ... |S| − 1]; retorna S
19 |  senão
20 |  |  S' ← vetor vazio
21 |  |  para i = 0 até m − 1 faça                 ▷ Copia vértices não disseminadores
22 |  |  |  se estado(S[i]) ≠ disseminador então
23 |  |  |  |  InsereNoFinal(S', S[i])
24 |  |  |  senão
25 |  |  |  |  j ← j + 1
26 |  |  S' ← S' + S[m ... |S| − 1]
27 |  |  S ← S'
28 |  PropagaçãoCompleta(G, S[0 ... m − 1 − j] + S[2m − j ... |S| − 1])
29 |  se S[0 ... m − 1 − j] + S[2m − j ... |S| − 1] é viável então
30 |  |  S ← S[0 ... m − 1 − j] + S[2m − j ... |S| − 1]
31 |  senão
32 |  |  S' ← S[0 ... m − 1 − j]
33 |  |  para i = m − j até 2m − 1 − j faça        ▷ Copia vértices não disseminadores
34 |  |  |  se estado(S[i]) ≠ disseminador então
35 |  |  |  |  InsereNoFinal(S', S[i])
36 |  |  S' ← S' + S[2m − j ... |S| − 1]
37 |  |  S ← S'
38 |  m ← (m − j)/2
39 retorna S
```

Os critérios de ordenação variam de acordo com cada heurística. Para as heurísticas `GR` e `WGR`, ordenamos S de forma não decrescente considerando, para cada v, o número de vizinhos conhecedores de v que não são disseminadores e que possuem v como o único vizinho disseminador. A justificativa é que quanto maior esse número, mais vizinhos de v dependem de v para tornarem-se conhecedores e, portanto, mais "essencial" v é. O cálculo desse número para todos os vértices pode ser feito em tempo $\mathcal{O}(|V|+|E|)$. Já a ordenação é feita em tempo $\mathcal{O}(|V|)$ por meio do algoritmo *Counting Sort*.

Para a heurística `RG`, não aplicamos qualquer permutação em S, uma vez que as sementes mais à esquerda são aquelas que foram inseridas de modo puramente aleatório. Por fim, para a heurística `SG`, S é ordenado de forma não decrescente considerando o valor do benefício $b(v)$ que cada vértice v possuía no momento em que v foi escolhido para ser adicionado ao conjunto semente. Esse valor é armazenado durante a fase de construção e claramente não altera a complexidade dessa fase da heurística `SG`, analisada na Seção 4.3.4. Nesse caso, a ordenação é feita em tempo $\mathcal{O}(|V|)$ pelo algoritmo *Counting Sort*.

Para o próximo passo, tomamos $m = |S|/2$ e simulamos o processo de propagação a partir de $S[m \dots |S|-1]$ (linha 2). Se $S[m \dots |S|-1]$ é viável (linha 3), então removemos de S todos os vértices pertencentes a $S[0 \dots m-1]$ e a segunda etapa é encerrada. Caso contrário, removemos de S os j vértices pertencentes a $S[0 \dots m-1]$ que tornaram-se disseminadores ao longo da propagação. O maior custo entre os passos das linhas 2 a 13 está associado à propagação realizada na linha 2 em tempo $\mathcal{O}(|V| + |E|)$. Em seguida, fazemos $m \leftarrow (m-j)/2$ e continuamos como descrito a seguir.

Simulamos o processo de propagação a partir de $S[m \dots |S| - 1]$ (linha 16). Se $S[m \dots |S| - 1]$ é viável (linha 17), então removemos de S os vértices pertencentes a $S[0 \dots m-1]$ e a segunda etapa é encerrada. Caso contrário, removemos de S todos os j vértices em $S[0 \dots, m-1]$ que tornaram-se disseminadores ao longo da propagação. Depois, na linha 28, simulamos o processo de propagação a partir de $S[0 \dots m-1-j] + S[2m-j \dots |S|-1]$. Se $S[0 \dots m-1-j] + S[2m-j \dots |S|-1]$ é viável (linha 29), então removemos de S todos os vértices pertencentes a $S[m-j \dots 2m-1-j]$. Caso contrário, removemos de S todos os vértices em $S[m-j \dots 2m-1-j]$ que tornaram-se disseminadores ao longo da propagação. Logo após, fazemos $m \leftarrow (m-j)/2$ e repetimos o procedimento descrito neste parágrafo até que $m = 0$.

O maior custo entre os passos do laço da linha 15 está associado às propagações realizadas nas linhas 16 e 28, cada uma delas em tempo $\mathcal{O}(|V|+|E|)$. Observe que temos $\mathcal{O}(\log|V|)$ iterações no laço da linha 15. Portanto, o Algoritmo 4.7 pode ser executado em tempo $\mathcal{O}(|E|\log|V|)$.

Etapa 3 Na terceira e última etapa da busca local, objetivamos particionar o conjunto semente e, sequencialmente, remover vértices de cada uma dessas partições.

Esse procedimento é descrito pelo Algoritmo 4.8 e ocorre da seguinte maneira. Sejam S o conjunto semente resultante da segunda etapa e $n = |S|$. Desejamos particionar o conjunto semente S em subconjuntos de tamanho $q \leq n$. Dessa forma, tomamos $\sigma = \lceil |S|/q \rceil$ (linha 1) e, no laço linha 2, particionamos S em $\sigma \geq 2$ conjuntos $S_1, S_2, \dots, S_\sigma$, de modo que S_1 contém os elementos de $S[0 \dots q-1]$, S_2 contém os elementos de $S[q \dots 2q-1]$

e assim por diante, até S_σ, que contém os elementos de $S[(\sigma-1)q \ldots |S|-1]$. Note que S_σ pode conter menos que q elementos. Esse particionamento pode ser feito em tempo $\mathcal{O}(|V|)$.

Algoritmo 4.8: Fase de busca local - Etapa 3.

Entrada: Instância $(G=(V,E),t)$; conjunto semente perfeito S; parâmetro q
Saída : Conjunto semente perfeito reduzido

```
1  σ ← ⌈|S|/q⌉
2  para i = 1 até σ faça                          ▷ Particiona S em σ subconjuntos
3      se i < σ então
4          S_i ← S[(σ−1)q ... σq − 1]
5      senão
6          S_i ← S[(σ−1)q ... |S| − 1]
7  para i = 1 até σ faça                          ▷ Remove sementes de cada subconjunto de S
8      S̄_i ← vetor vazio
9      para j = 1 até σ faça                      ▷ Toma S̄_i = S_1 + ... + S_{i−1} + S_{i+1} + ... + S_σ
10         se j ≠ i então
11             S̄_i ← S̄_i + S_j
12     PropagaçãoCompleta(G, S̄_i)
13     se S̄_i é viável então
14         S_i ← vetor vazio
15     senão
16         S' ← vetor vazio
17         para j = 0 até |S_i| − 1 faça          ▷ Copia vértices não disseminadores
18             se estado(S_i[j]) ≠ disseminador então
19                 InsereNoFinal(S', S_i[j])
20         S_i ← S'
21 S ← vetor vazio
22 para i = 0 até σ faça                          ▷ Toma S = S_1 + S_2 + ... + S_σ
23     S ← S + S_i
24 retorna S
```

Depois, no laço da linha 7, fazemos o que segue. Para cada $i \in \{1, 2, \ldots, \sigma\}$, obtemos o conjunto semente $\overline{S_i} = S_1 + S_2 + \ldots + S_{i-1} + S_{i+1} + \ldots + S_{\sigma-1} + S_\sigma$ e simulamos a propagação a partir de $\overline{S_i}$ (linha 12). Se $\overline{S_i}$ for viável, então removemos todos os elementos de S_i. Caso contrário, todos os vértices em S_i que tornaram-se disseminadores ao longo da propagação são removidos desse vetor. Observe que o passo mais custoso dentro do laço da linha 7 consiste no processo de propagação realizado na linha 12 que toma tempo $\mathcal{O}(|V|+|E|)$. Portanto, o laço pode ser completado em tempo $\mathcal{O}(\sigma(|V|+|E|))$.

Por fim, formamos o conjunto semente perfeito composto por $S_1 + S_2 + \ldots + S_\sigma$ em tempo $\mathcal{O}(|V|)$ e o retornamos. A complexidade total do Algoritmo 4.8 é $\mathcal{O}(\sigma(|V|+|E|))$.

No próximo capítulo, apresentamos um conjunto de experimentos que contemplam o uso das técnicas e algoritmos propostos nesta seção.

Capítulo 5

Experimentos Computacionais

Neste capítulo, descrevemos os procedimentos experimentais realizados nesse trabalho com o principal objetivo de determinar a qualidade das heurísticas desenvolvidas para o PAP. Na Seção 5.1 descrevemos a geração de um conjunto de 840 instâncias do PAP utilizadas nos experimentos. Mais à frente, na Seção 5.2, introduzimos os testes estatísticos que utilizamos para comparar a qualidade das soluções obtidas pelas heurísticas. Na Seção 5.3, discorremos sobre o projeto dos experimentos e, finalmente, apresentamos e discutimos os resultados obtidos na Seção 5.4.

5.1 Geração de Instâncias

Nesta seção, descrevemos o conjunto de instâncias do *benchmark* que geramos para testar nossas heurísticas. A literatura sobre propagação de informações em redes sociais mostra que redes sociais reais tendem a exibir duas características principais [6]. A primeira é o crescimento do número de membros da rede. A segunda é que quando um novo indivíduo surge na rede, este tende a formar conexões com os indivíduos já existentes que possuem um número elevado de conexões.

De acordo com [4], um dos principais algoritmos utilizados para gerar grafos que representam redes sociais com esses atributos é o método Barábasi-Albert (BA) [6]. O BA funciona da seguinte forma.

Suponha que desejamos produzir um grafo conexo $G = (V, E)$ com n vértices. Para algum k, $1 \leq k \leq n-1$, inicializamos G com $|V| = k$ e $E = \emptyset$. Em seguida, um novo vértice é adicionado ao grafo e conectado a cada um dos primeiros k vértices. Iterativamente, para um total de $n-k-1$ vezes, um novo vértice, digamos v, é adicionado e k vértices inseridos anteriormente são escolhidos para serem conectados a v. A probabilidade de se selecionar um vértice $u \neq v$ para ser um dos vizinhos de v é dada por $P(u) = d(u)/\sum_{w \in V-\{v\}} d(w)$. No final, G terá $|V| = n$ vértices e $|E| = (n-k)k = nk - k^2$ arestas, considerando o valor de k com o qual o processo foi iniciado.

Observe que, uma vez que n é estabelecido, o número final de arestas depende unicamente da escolha de k. Denotamos por $|E|_{\min}$ e $|E|_{\max}$ o número mínimo e máximo de arestas, respectivamente, que podem ser geradas por intermédio do método BA. Como $1 \leq k \leq n-1$, o valor da função $f(k) = nk - k^2$ é mínimo para $k = n-1$ ou $k = 1$ e,

portanto, $|E|_{\min} = n-1$, e é máximo quando $k = n/2$ e, neste caso, $|E|_{\max} = \lfloor n^2/4 \rfloor$. Descrevemos agora como uma modificação simples permite que `BA` gere grafos de n vértices com exatamente m arestas, para qualquer m inteiro no intervalo $[|E|_{\min}, |E|_{\max}]$.

Para este fim, como queremos que o valor de $|E|$ seja igual a m, escolhemos $k = \lfloor x \rfloor$, onde x é a menor das raízes de $x^2 - nx + m = 0$. Em seguida, procedemos com o método `BA` padrão descrito anteriormente, obtendo um grafo com $|V| = n$ vértices e $|E| = nk - k^2$ arestas. Se $|E| = m$, conforme desejado, terminamos[11]. No entanto, se $|E| < m$, adicionamos $m - |E|$ novas arestas como segue.

Primeiramente, escolhemos um vértice v já existente no grafo de maneira aleatória e com probabilidade uniforme para ser um dos extremos de uma nova aresta. A seguir, escolhemos o outro extremo $u \neq v$, com $u \notin N(v)$, aleatoriamente de acordo com a probabilidade $P(u)$, e adicionamos a aresta $\{u, v\}$. O procedimento descrito neste parágrafo é repetido até que se obtenha $|E| = m$.

Ressaltamos que essa modificação do `BA` permite controle total sobre o número final de arestas, preservando a propriedade de ligação preferencial de vértices do grafo. Além disso, note que, no segundo caso, o número arestas extras adicionadas é menor ou igual ao valor da expressão $x^2 - nx - (\lfloor x \rfloor^2 - n\lfloor x \rfloor) = (x + \lfloor x \rfloor)(x - \lfloor x \rfloor) + n(\lfloor x \rfloor - x)$. Se fizermos $\alpha = x - \lfloor x \rfloor$, então a expressão é igual a $(x + \lfloor x \rfloor)\alpha - n\alpha = (x + \lfloor x \rfloor - n)\alpha$. Como $\alpha < 1$, o número de arestas adicionais é menor ou igual a $(x + \lfloor x \rfloor - n) \leq 2k + 1 - n$. Portanto, a modificação causa pouco impacto na quantidade final de arestas.

Neste trabalho, utilizamos o `BA` modificado para produzir novas instâncias do `PAP`. Dessa forma, para cada $n \in \{10, 15, 20, \ldots, 95\} \cup \{100, 200, \ldots, 1000\}$ geramos 30 grafos com n vértices e com número de arestas variando de $|E|_{\min}$ a $|E|_{\max}$. Para cada n fixo, definimos $r = (|E|_{\max} - |E|_{\min})/29$ e para cada $m \in \{|E|_{\min} + \lceil r\sigma \rceil \mid \sigma \in \{0, 1, 2, 3 \ldots, 29\}\}$ criamos um grafo com n vértices e m arestas. Para acomodar o caso $n = 10$, definimos $r = (|E|_{\max} - |E|_{\min})/9$ e para cada $m \in \{|E|_{\min} + \lceil r\sigma \rceil \mid \sigma \in \{0, 1, 2, 3, \ldots, 9\}\}$ criamos 3 grafos distintos com 10 vértices e m arestas. Ao final, obtém-se 840 grafos com características de redes sociais de acordo com os critérios definidos em [6].

Conforme vimos na Seção 3.1, além de um grafo, uma função limiar precisa ser estabelecida para compor uma instância do `PAP`. Para cada grafo G gerado, escolhemos construir uma instância (G, t) tomando $t(v) = \lceil 0.5 \cdot d(v) \rceil$, onde t é chamada *função limiar de maioria* (ou *majority threshold function*) [21]. Denotamos por $\mathcal{I}$ o conjunto que contém as 840 instâncias produzidas e por $\mathcal{I}_n \subseteq \mathcal{I}$ o subconjunto das instâncias compostas por grafos com n vértices.

Em [47], disponibilizamos a nossa implementação do `BA` modificado na linguagem de programação `Python`, versão 3. O nosso código utiliza como subrotina uma implementação do `BA` original proveniente da biblioteca `NetworkX` [43], também na linguagem `Python`. Além disso, em [47] podem ser encontradas as 840 instâncias do `PAP` produzidas, assim como as melhores soluções primais conhecidas para essas.

A Tabela 5.1 apresenta informações acerca dos grafos das instâncias de cada conjunto $\mathcal{I}_n$. Para cada valor de n, há uma linha da tabela contendo o valor da cardinalidade do conjunto de vértices, assim como os valores de $|E|_{\min}$, $|E|_{\max}$, $D_{\min}$ e $D_{\max}$, onde estes dois últimos representam, respectivamente, a menor e a maior densidade dentre os

[11] Este caso ocorre se x é inteiro.

grafos das instâncias de $\mathcal{I}_n$. A densidade de um grafo não direcionado é calculada por $2|E|/(|V|^2 - |V|)$.

Tabela 5.1: Descrição dos subconjuntos do *benchmark* $\mathcal{I}$.

	$\|V\|$	$\|E\|_{\min}$	$\|E\|_{\max}$	$D_{\min}$	$D_{\max}$
$\mathcal{I}_{10}$	10	9	25	0.20000	0.55556
$\mathcal{I}_{15}$	15	14	56	0.13333	0.53333
$\mathcal{I}_{20}$	20	19	100	0.10000	0.52632
$\mathcal{I}_{25}$	25	24	156	0.08000	0.52000
$\mathcal{I}_{30}$	30	29	225	0.06667	0.51724
$\mathcal{I}_{35}$	35	34	306	0.05714	0.51429
$\mathcal{I}_{40}$	40	39	400	0.05000	0.51282
$\mathcal{I}_{45}$	45	44	506	0.04444	0.51111
$\mathcal{I}_{50}$	50	49	625	0.04000	0.51020
$\mathcal{I}_{55}$	55	54	756	0.03636	0.50909
$\mathcal{I}_{60}$	60	59	900	0.03333	0.50847
$\mathcal{I}_{65}$	65	64	1056	0.03077	0.50769
$\mathcal{I}_{70}$	70	69	1225	0.02857	0.50725
$\mathcal{I}_{75}$	75	74	1406	0.02667	0.50667
$\mathcal{I}_{80}$	80	79	1600	0.02500	0.50633
$\mathcal{I}_{85}$	85	84	1806	0.02353	0.50588
$\mathcal{I}_{90}$	90	89	2025	0.02222	0.50562
$\mathcal{I}_{95}$	95	94	2256	0.02105	0.50526
$\mathcal{I}_{100}$	100	99	2500	0.02000	0.50505
$\mathcal{I}_{200}$	200	199	10000	0.01000	0.50251
$\mathcal{I}_{300}$	300	299	22500	0.00667	0.50167
$\mathcal{I}_{400}$	400	399	40000	0.00500	0.50125
$\mathcal{I}_{500}$	500	499	62500	0.00400	0.50100
$\mathcal{I}_{600}$	600	599	90000	0.00333	0.50083
$\mathcal{I}_{700}$	700	699	122500	0.00286	0.50072
$\mathcal{I}_{800}$	800	799	160000	0.00250	0.50063
$\mathcal{I}_{900}$	900	899	202500	0.00222	0.50056
$\mathcal{I}_{1000}$	1000	999	250000	0.00200	0.50050

5.2 Testes Estatísticos

Nesta seção, descrevemos três testes estatísticos que utilizamos para comparar a qualidade das soluções produzidas pelas nossas heurísticas para instâncias do `PAP`. Trata-se de testes não paramétricos, os quais são aplicados quando a distribuição dos dados a serem analisados é desconhecida. A seguir, apresentamos resumidamente os testes de acordo com a descrição feita em [25].

Primeiramente, suponha que tenhamos K algoritmos que resolvem um determinado problema de otimização e N instâncias desse problema. Para os três testes, temos como hipótese nula a proposição de que não existe *diferença estatisticamente significativa* (DES) entre os valores das soluções obtidas pelos K algoritmos para as N instâncias. Cada um

dos testes é baseado no conceito de *rank*, cuja definição é exibida a seguir.

Suponha que ordenamos uma lista dos K algoritmos de acordo com os valores das soluções obtidas para a i-ésima instância, de maneira que dada a posição pos(j) do j-ésimo algoritmo após a ordenação, com $1 \leq \text{pos}(j) \leq K$, um algoritmo está em uma posição menor que pos(j) se, e somente se, obteve uma solução com valor melhor ou igual ao valor obtido pelo j-ésimo algoritmo. Seja Γ_j o conjunto dos algoritmos que produziram soluções com o mesmo valor que a solução produzida pelo j-ésimo algoritmo.

O cálculo do *rank* do j-ésimo algoritmo para a i-ésima instância, denotado por r_i^j, é dado pela expressão (5.1):

$$r_i^j = \frac{1}{|\Gamma_j|} \sum_{\ell \in \Gamma_j} \text{pos}(\ell) \quad \forall (i,j) \in \{1, \ldots, N\} \times \{1, \ldots K\} \tag{5.1}$$

Para exemplificar, sejam A, B, C e D quatro algoritmos para um problema de minimização. Suponha que $10, 20, 20$ e 30 foram os valores das soluções produzidas por esses algoritmos, respectivamente, para uma instância do problema. Então, os *ranks* dos algoritmos para essa instância são $1, 2.5, 2.5$ e 4, respectivamente.

A partir do cômputo dos *ranks* obtidos por um mesmo algoritmo para todas as N instâncias, define-se o *rank* médio do j-ésimo algoritmo, denotado por R_j, por meio da expressão (5.2):

$$R_j = \frac{1}{N} \sum_{i=1}^{N} r_i^j \quad \forall j \in \{1, \ldots, K\} \tag{5.2}$$

Teste de Iman-Davenport Nesse ponto, explicitamos o primeiro teste estatístico, chamado de Teste de Iman-Davenport [33], cujo resultado é calculado usando-se as expressões (5.3) e (5.4):

$$\chi_F^2 = \frac{12N}{K(K+1)} \left[\sum_{j=1}^{K} R_j^2 - \frac{K(K+1)^2}{4} \right] \tag{5.3}$$

$$\text{F}_F = \frac{(N-1)\chi_F^2}{N(K-1) - \chi_F^2} \tag{5.4}$$

A estatística F_F segue a distribuição F (de Fisher-Snedecor) com graus de liberdade iguais a $K-1$ e $(K-1)(N-1)$. Assim, usando os graus de liberdade mencionados e o nível de confiança desejado, obtém-se o *valor crítico* que segue a distribuição F. Segundo [25], a tabela de valores críticos para a distribuição F pode ser encontrada na maioria dos livros de estatística. Se F_F for maior que o valor crítico, então rejeita-se a hipótese nula.

Teste de Nemenyi O segundo teste estatístico é o Teste de Nemenyi [42]. Nesse teste, dizemos que os valores das soluções obtidas por dois algoritmos apresentam DES, se os seus respectivos *ranks* médios diferem por pelo menos o valor da *diferença crítica*, denotada por DC e calculada pela expressão (5.5). O valor de q_α está em função do nível de confiança α e pode ser encontrado na Tabela 5(a) do artigo [25].

$$\mathrm{DC} = q_\alpha \sqrt{\frac{K(K+1)}{6N}} \tag{5.5}$$

O resultado do teste de Nemenyi é representado graficamente, como podemos ver no exemplo ilustrado pela Figura 5.1. Os valores de *rank* médio estão representados no eixo horizontal, em ordem crescente da esquerda para a direita. Cada algoritmo é representado por uma linha vertical com seu respectivo nome e valor de *rank* médio logo abaixo. Uma linha horizontal de comprimento igual ao valor da DC é posicionada acima do eixo horizontal principal. Os algoritmos cujos *ranks* médio diferem menos do que a DC são conectados por segmento horizontal em negrito, de comprimento inferior ao valor da DC.

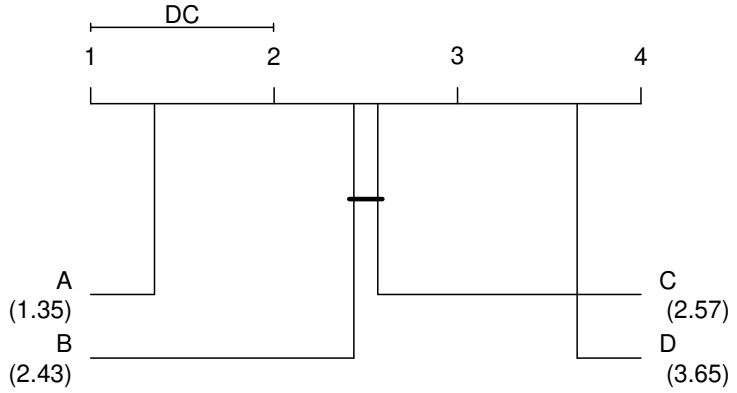


Figura 5.1: Exemplo de resultado do Teste de Nemenyi.

No nosso exemplo, o teste determinou que os valores das soluções produzidas pelos algoritmos B e C não apresentam DES. Para todos os outros pares de algoritmos as soluções apresentam DES.

Teste de Wilcoxon Por fim, descrevemos o terceiro teste, denominado Teste de Wilcoxon [55], que é aplicável quando queremos verificar se há DES entre os valores das soluções obtidas por um único par de algoritmos. O Teste de Wilcoxon é um melhor discriminador do que os testes anteriores quando utilizados para $K = 2$.

Suponha, sem perda de generalidade, que temos dois algoritmos para um problema de minimização. Denote por d_i, com $i \in \{1, \ldots, N\}$ a diferença entre os valores das soluções obtidas pelos dois algoritmos para a i-ésima instância. Considere que as diferenças são *rankeadas* de acordo com os seus valores absolutos e denote por r_{d_i} o *rank* da i-ésima diferença.

Dessa maneira, a expressão $\sum_{d_i<0} r_{d_i}$ consiste na soma dos *ranks* das diferenças negativas, ou seja, das diferenças em que o primeiro algoritmo produziu uma solução melhor que o segundo. Analogamente, a expressão $\sum_{d_i>0} r_{d_i}$ consiste na soma dos *ranks* das diferenças positivas, ou seja, das diferenças em que o segundo algoritmo produziu uma solução melhor que o primeiro. Por fim, na expressão $\sum_{d_i=0} r_{d_i}$, temos a soma dos *ranks* das diferenças em que houve empate no valor das soluções produzidas pelos dois algoritmos.

A partir das expressões supracitadas, definem-se os valores de R^- e R^+ segundo as expressões (5.6) e (5.7):

$$R^- = \sum_{d_i<0} r_{d_i} + \frac{1}{2}\sum_{d_i=0} r_{d_i} \tag{5.6}$$

$$R^+ = \sum_{d_i>0} r_{d_i} + \frac{1}{2}\sum_{d_i=0} r_{d_i} \tag{5.7}$$

Seja $T = \min\{R^-, R^+\}$. De acordo com [25], para $N \leq 25$, o valor crítico para o teste de Wilcoxon pode ser consultado na maioria dos livros de estatística. Nesse caso, a hipótese nula é rejeitada se T é menor ou igual ao valor crítico.

Por outro lado, se $N > 25$, calcula-se a estatística z de acordo com a expressão (5.8):

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{5.8}$$

Nesse segundo caso, rejeita-se a hipótese nula, se $z < -1.96$, considerando um nível de confiança de 95%.

5.3 Projeto dos Experimentos

Nesta seção, delineamos os experimentos conduzidos neste trabalho, cujos dois principais objetivos são:

(i) Determinar a melhor heurística dentre `GR`, `WGR`, `RG` e `SG` quanto à qualidade das soluções produzidas e quanto à velocidade e robustez em obter boas soluções.

(ii) Comparar a heurística `CGR` [21] e a melhor das nossas heurísticas quanto à qualidade das soluções produzidas.

Para atingir o objetivo (i), utilizamos somente as instâncias do *benchmark* $\mathcal{I}$. Para o objetivo (ii), utilizamos um conjunto adicional de instâncias que será apresentado mais à frente nesta seção.

O primeiro passo para (i) consiste em calibrar os parâmetros ajustáveis das heurísticas. Para esse fim, utilizamos o pacote `irace` [39], versão 3.4.1, o qual implementa um procedimento de competição iterada, que é uma extensão do *Iterated F-Race* [8], um algoritmo de competição para a tarefa de configuração automática de parâmetros de algoritmos de otimização.

O `irace` recebe como entrada um conjunto de instâncias de treino, um arquivo que descreve a configuração do experimento (tempo máximo de execução, caminho até o arquivo executável do algoritmo, etc.), e um arquivo que descreve quais parâmetros do algoritmo devem ser configurados e seus respectivos domínios. A saída do `irace` consiste em uma lista ordenada (possivelmente com um único item) das melhores configurações encontradas pela ferramenta para os parâmetros desejados.

Por exemplo, suponha que o nosso algoritmo possui dois parâmetros configuráveis α e β, números reais, tais que $\alpha \in [0, 1]$ e $\beta \in [0, 0.5]$. Uma possível saída do `irace` consiste

em uma lista de pares $[(\alpha = 0.3, \beta = 0.1), (\alpha = 0.25, \beta = 0.2)]$ em que a configuração $(\alpha = 0.3, \beta = 0.1)$ foi aquela a partir da qual o algoritmo produziu as melhores soluções e $(\alpha = 0.25, \beta = 0.2)$ foi a segunda melhor configuração nesse quesito.

Uma vez que os parâmetros para as nossas quatro heurísticas tenham sido ajustados, executamos cada heurística uma vez em cada uma das instâncias do *benchmark* $\mathcal{I}$. Em seguida, fazemos uso de um resolvedor comercial de `PLI`, o `CPLEX` [32], versão 12.10, para obter soluções ótimas para instâncias de $\mathcal{I}$, utilizando as formulações `PLI-RODADAS` e `PLI-ARCOS`. Para cada instância I de $\mathcal{I}$, fornecemos ao resolvedor como solução primal inicial a melhor solução obtida para I dentre as soluções produzidas pelas quatro heurísticas para essa instância. Denotamos por $\mathcal{I}_{\text{ótimo}}$ o conjunto das instâncias de $\mathcal{I}$ para as quais o valor ótimo foi obtido pelo `CPLEX`, com pelo menos um dos modelos desenvolvidos.

Na sequência, verificamos para quantas instâncias de $\mathcal{I}_{\text{ótimo}}$ cada uma das heurísticas conseguiu também encontrar soluções ótimas e, nos casos em que não conseguiu, o quão distantes os valores das soluções ficaram dos valores ótimos. Embora seja verdade que, em geral, heurísticas não visem a obtenção de soluções comprovadamente ótimas, esse resultado fornece uma avaliação qualitativa das soluções produzidas pelos algoritmos (em relação ao valor ótimo) que permite uma comparação direta entre as nossas heurísticas.

Depois, seguimos com a aplicação dos testes estatísticos apresentados na Seção 5.2, de acordo com a metodologia discutida em [25], a fim de determinar a heurística que produziu as melhores soluções, considerando todas as instâncias de $\mathcal{I}$. Nessa etapa, utilizamos o pacote `scmamp` [11], que é implementado na linguagem `R` e possui funções que realizam o cômputo dos testes relatados.

Primeiro, aplicamos o Teste de Iman-Davenport para verificar se existe diferença estatisticamente significativa (DES) entre os valores das soluções produzidas pelos quatro algoritmos. Se não existe DES, então, na prática, não existe razão para se escolher uma heurística específica dentre as quatro, quando o critério de escolha for a qualidade das soluções produzidas para as instâncias em $\mathcal{I}$. Por ouro lado, se há DES, então prosseguimos para o Teste de Nimenyi.

Ao aplicar o Teste de Nimenyi, o resultado fornecido nos permite determinar o subconjunto das heurísticas que produziu as melhores soluções. Se esse subconjunto não for unitário, então as heurísticas pertencentes a ele produziram soluções para as quais não há DES entre os valores das soluções, segundo esse teste. Nesse caso, aplicamos o Teste de Wilcoxon para cada par de heurísticas desse subconjunto com o objetivo de corroborar o resultado do Teste de Nimenyi ou detectar que há DES entre algum dos pares.

Após essa sequência de testes estatísticos, determinamos quais das nossas heurísticas produziram as melhores soluções para o conjunto $\mathcal{I}$. Para essas heurísticas, nós prosseguimos com uma análise de velocidade e robustez por meio de uma técnica chamada *multiple time-to-target plot* (`mttt-plot`) [3, 50].

Nessa etapa, o primeiro passo é definir o conjunto de instâncias a ser utilizado. No nosso caso, escolhemos um subconjunto do *benchmark* $\mathcal{I}$. O segundo passo é determinar para cada instância um *valor alvo* que representa um valor de solução desejável de ser obtido pelas heurística. Em geral, o valor alvo é estabelecido como um fator do melhor valor de solução conhecido para a instância. O terceiro passo consiste em executar cada heurística múltiplas vezes com cada uma das instâncias de $\mathcal{I}$. Uma execução é interrompida

somente quando o algoritmo produzir uma solução tão boa quanto o alvo estabelecido ou quando o limite de tempo de execução for atingido.

Ao final, a partir dos tempos de execução registrados, obtemos gráficos que indicam a probabilidade, em relação ao somatório dos tempos de execução, de cada heurística ter encontrado soluções, para todas as instâncias, pelo menos tão boas quanto os alvos estabelecidos. A Figura 5.2 ilustra um exemplo de dois gráficos obtidos via `mttt-plot`. Desses gráficos, podemos ler que a probabilidade do algoritmo `GRASP+PRF` atingir todos os alvos em até 100 segundos de execução (ao todo) é de aproximadamente 17%. Já a probabilidade do algoritmo `GRASP+PRB` atingir todos os alvos em até 100 segundos de execução (ao todo) é de aproximadamente 72%.

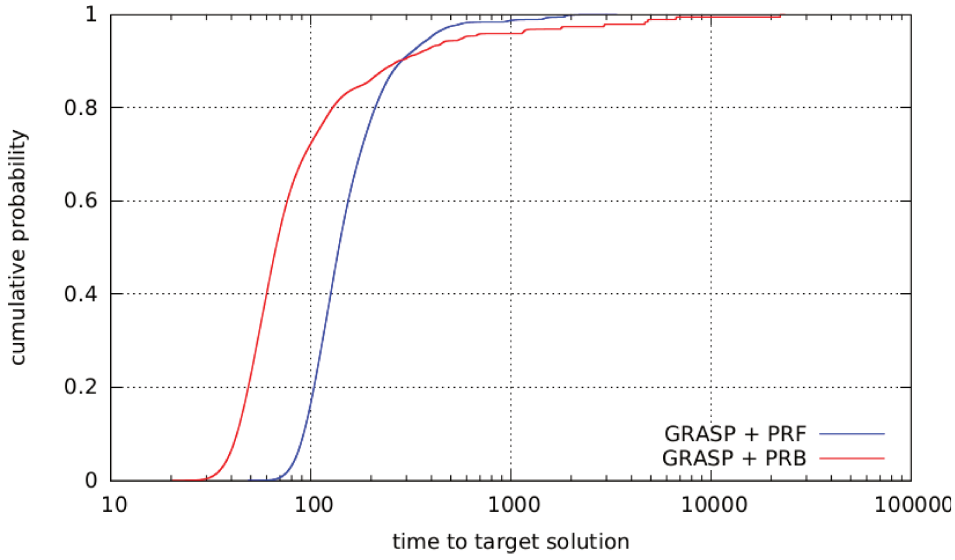


Figura 5.2: Exemplo de gráfico obtido via `mttt-plot` (extraído de [50]).

Dentre as heurísticas para as quais a técnica de `mttt-plot` foi aplicada, escolhemos, como a melhor das heurísticas, aquela que se mostrou mais robusta, de acordo com os resultados do `mttt-plot`.

Finalmente, visando o objetivo (ii) descrito no início desta seção, conduzimos um último experimento a fim de comparar a melhor das nossas heurísticas com a heurística `CGR`. Para essa etapa, executamos a nossa melhor heurística em um conjunto de 17 instâncias que foram utilizadas em [21] para teste da heurística `CGR`. A este conjunto nos referiremos como *benchmark* C. Para cada instância I de C, o grafo contido em I representa uma rede social real e a função limiar empregada é a função limiar de maioria (veja Seção 5.1). A Tabela 5.2 apresenta informações acerca dos grafos das instâncias de C. A qualidade das soluções produzidas por ambas as heurísticas são comparadas diretamente e apresentadas na próxima seção.

As implementações das heurísticas e das formulações de `PLI` para o `CPLEX` foram realizadas na linguagem de programação `C++` e estão disponíveis em [46]. Todas as execuções foram realizadas no mesmo ambiente computacional equipado com um processador Intel® Xeon® E5-2630 v4, 64 GB de memória RAM e sistema operacional Ubuntu 18.04.2 LTS.

Por fim, estabelecemos a sequência de passos que foi realizada para cada execução feita

Tabela 5.2: Descrição do *benchmark C*.

Instância	|V|	|E|
Amazon0302 [38]	262111	899792
BlogCatalog3 [59]	10312	333983
BuzzNet [59]	101163	2763066
CA-AstroPh [38]	18771	198050
CA-CondMat [38]	23133	93439
CA-GrQc [38]	5241	14484
CA-HepPh [38]	12006	118489
CA-HepTh [38]	9875	25973
Cit-HepTh [38]	27769	352285
Delicious [38]	536408	1366136
Douban [59]	154908	327162
Facebook [38]	4039	88234
Jazz [44]	198	2742
Karate [44]	34	78
Last.fm [59]	1191812	4519340
Power Grid [44]	4941	6594
YouTube2 [59]	1138499	2990443

nos experimentos. Por simplificação, vamos considerar que dada uma instância (G, t) do `PAP`, G' é o grafo G com limiares associados aos vértices dados pela função t.

No caso das heurísticas, o primeiro passo consiste em pré-processar G' com as técnicas apresentadas na Seção 4.1. Isto resulta em um grafo G'_p. Adicionalmente, geramos uma função injetora f_v que mapeia os vértices de G'_p nos vértices de G'. Em seguida, aplicamos alguma das heurísticas propostas para obter uma solução $S[G'_p]$ para G'_p. Subsequentemente, usamos a função f_v supracitada para obter uma solução $S[G']$ para G'. Isso encerra o passo a passo da execução das heurísticas.

Já no caso do `CPLEX`, o passo a passo difere do que foi descrito para as heurísticas em apenas um aspecto. Quando usamos o `CPLEX` para solucionar G'_p, também fornecemos ao resolvedor a melhor solução $S[G'_p]$ conhecida. Assim, produzimos uma solução $S'[G'_p]$ pelo `CPLEX`, na qual aplicamos a função de mapeamento de vértices f_v para obtermos uma solução $S'[G']$ para G'. Note que $S'[G']$ é ótima se o `CPLEX` alcançou a otimalidade, caso contrário, temos uma solução (viável) incumbente. A Figura 5.3 sumariza os passos apresentados.

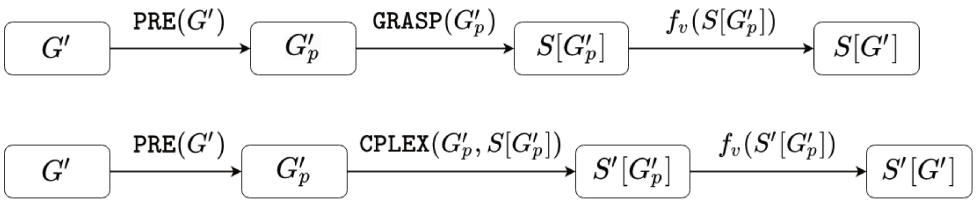


Figura 5.3: Passo a passo de cada execução do `GRASP` e do `CPLEX`.

5.4 Resultados

Nesta seção, apresentamos e discutimos os resultados obtidos a partir dos experimentos realizados de acordo com o projeto descrito na Seção 5.3.

Como mencionado na seção anterior, para cada heurística existem parâmetros de entrada nas fases de construção e de busca local que podem ser calibrados visando um melhor desempenho. Para algumas heurísticas, a calibragem pode ser feita como uma escolha direta dentro de um intervalo de possíveis valores. Para `GR`, por exemplo, simplesmente tomamos $\alpha \in (0, 1)$. Por outro lado, em alguns casos, é mais adequado definir o valor do parâmetro como um fator de algum outro argumento fornecido na entrada. Para `SG`, por exemplo, podemos tomar ℓ como uma fração de $|V|$. No segundo caso, a calibragem é feita no fator, ao qual nos referimos como *parâmetro auxiliar*.

Primeiramente, denote por $pfc \in (0, 1)$ o parâmetro auxiliar da fase de construção. Se V é o conjunto de vértices do grafo dado na entrada dessa fase, então, para `GR` e `WGR`, fazemos $\alpha = pfc$. Já para `RG` e `SG`, tomamos $p = pfc \cdot |V|$ e $\ell = pfc \cdot |V|$, respectivamente. Além disso, denote por $pfbl \in (0, 0.5)$ o parâmetro auxiliar da terceira etapa da busca local de cada uma das heurísticas. Se S é o conjunto semente dado na entrada dessa etapa, fazemos $q = pfbl \cdot |S|$.

Utilizamos o `irace` [39] para calibrar os parâmetros *pfc* e *pfbl*. Para cada heurística, fizemos o que segue. Executamos o `irace` com o conjunto de instâncias de treinamento $\mathcal{I}_{1000}$ (instâncias de $\mathcal{I}$ com 1000 vértices). Observe que, ao longo de sua execução, o `irace` invoca a heurística inúmeras vezes, utilizando o conjunto de treinamento. Portanto, um possível critério de parada para esse programa consiste em limitar a soma dos tempos das execuções da heurística, considerando todas as vezes em que essa foi invocada. Dessa forma, configuramos o `irace` para que sua execução fosse finalizada quando a soma total dos tempos das execuções da heurística ultrapassasse 24 horas. Em cada invocação, fixamos como critério de parada da heurística o total de 1000 iterações[12] de `GRASP`. Além disso, a precisão dos parâmetros foi fixada em duas casas decimais.

A Tabela 5.3 apresenta as melhores configurações de valores para os parâmetros auxiliares de cada heurística, de acordo com o `irace`.

Tabela 5.3: Valores calibrados para os parâmetros auxiliares das heurísticas.

	`GR`	`WGR`	`RG`	`SG`
pfc	0.21	0.35	0.02	0.97
pfbl	0.47	0.36	0.02	0.44

Fazendo uso dos parâmetros calibrados, testamos as quatro heurísticas com todas as instâncias de $\mathcal{I}$. Cada heurística foi executada uma única vez com cada instância. O tempo limite de cada execução foi fixado em 5 minutos, independentemente do tamanho da instância[13]. A seguir, descrevemos as execuções feitas com os modelos de `PLI`.

Denote por `PLI-RODADAS-1` e `PLI-RODADAS-2` os modelos de `PLI` que consistem no modelo `PLI-RODADAS` com a adição das desigualdades (4.7), porém, para `PLI-RODADAS-2`, com

[12] Em experimentos preliminares, constatamos que, em geral, a solução retornada pelos algoritmos foi gerada até a milésima iteração.

[13] Essa configuração mostrou-se suficiente para que cada execução tivesse ao menos 1000 iterações.

adição de (4.7) apenas para τ fixado no valor 1. Inicialmente, verificamos que para algumas instâncias de $\mathcal{I}$, o modelo `PLI-RODADAS-2` fez com que o `CPLEX` alcançasse o valor ótimo mais rapidamente do que com o modelo `PLI-RODADAS-1`. Uma possível explicação para esse fenômeno é que como o modelo `PLI-RODADAS-2` é mais compacto que `PLI-RODADAS-1`, de maneira geral mais passos do `CPLEX` são realizados com `PLI-RODADAS-2` do que com `PLI-RODADAS-1` em um mesmo intervalo de tempo. Além disso, `PLI-RODADAS-2` ainda se beneficia de parte da eficácia das restrições (4.7).

Preliminarmente, averiguamos que para instâncias com mais de 100 vértices geradas de acordo com o procedimento descrito na Seção 5.1, os modelos `PLI-RODADAS-1` e `PLI-RODADAS-2` não foram capazes de produzir soluções ótimas ou, durante a execução do `CPLEX`, ocorria *overflow* de memória devido ao tamanho do modelo carregado. O mesmo caso se aplica ao modelo `PLI-ARCOS` para instâncias com mais de 40 vértices.

Isto posto, para cada um dos modelos `PLI-RODADAS-1`, `PLI-RODADAS-2` e `PLI-ARCOS`, executamos o `CPLEX` uma única vez com cada uma das instâncias do subconjunto de $\mathcal{I}$ composto por instâncias com grafos de até 100 vértices. Para cada execução, estabelecemos o limite de tempo de execução em 1 hora. Ademais, fornecemos ao `CPLEX` como solução primal inicial a melhor solução conhecida para a instância dentre aquelas produzidas pelas quatro heurísticas. Além disso, configuramos o `CPLEX` para que este utilizasse as suas heurísticas nativas e rotinas de obtenção de planos de corte (veja Seção 2.3).

As Figuras 5.4, 5.5 e 5.6 indicam as instâncias para as quais uma solução ótima foi obtida pelos modelos `PLI-RODADAS-1`, `PLI-RODADAS-2` e `PLI-ARCOS`, respectivamente, de acordo com as densidades dos grafos que compõem tais instâncias.

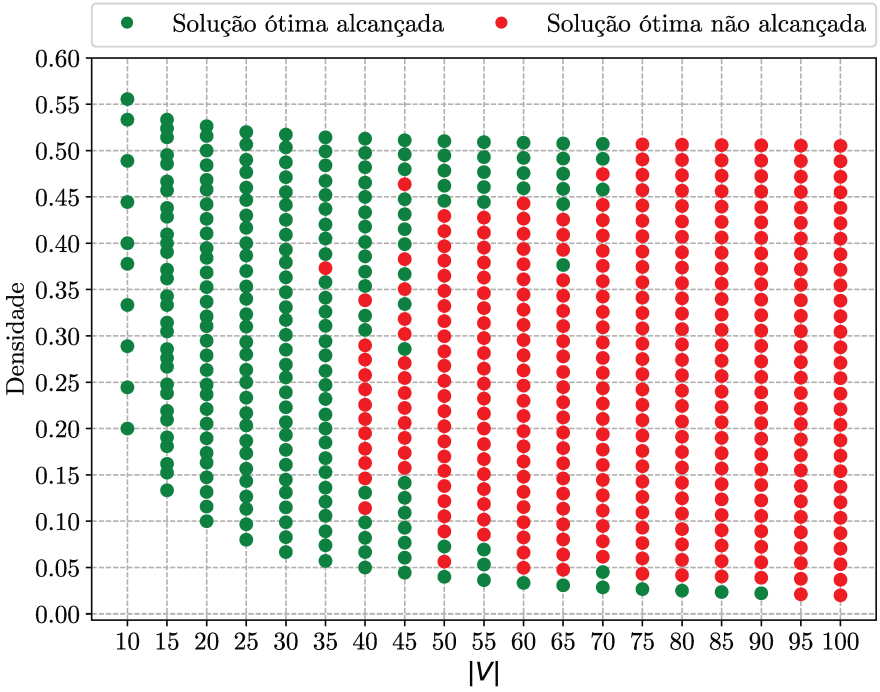


Figura 5.4: Otimalidades atingidas pelo modelo `PLI-RODADAS-1` para as instâncias de $\mathcal{I}$.

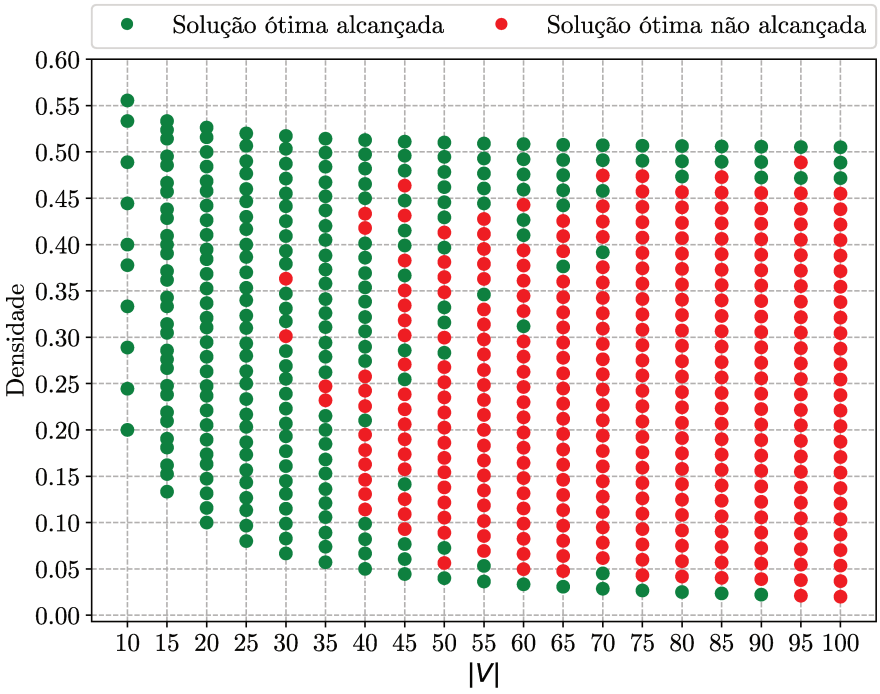


Figura 5.5: Otimalidades atingidas pelo modelo `PLI-RODADAS-2` para as instâncias de $\mathcal{I}$.

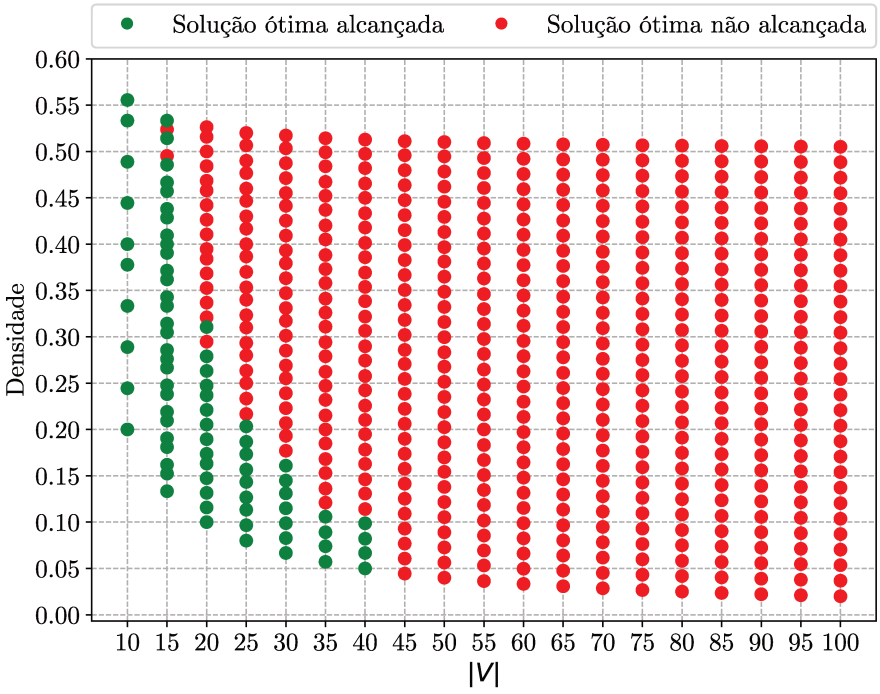


Figura 5.6: Otimalidades atingidas pelo modelo `PLI-ARCOS` para as instâncias de $\mathcal{I}$.

Considerando o nosso *benchmark* $\mathcal{I}$ de 840 instâncias, os modelos `PLI-RODADAS-1` e `PLI-RODADAS-2` produziram soluções ótimas para 250 e 268 instâncias, respectivamente. A união dessas totaliza 281 instâncias. Como todas as instâncias para as quais o valor ótimo foi alcançado pelo modelo `PLI-ARCOS` também tiveram o valor ótimo alcançado pelos

modelos `PLI-RODADAS-1` e `PLI-RODADAS-2`, ao todo, obtivemos os valores ótimos para 281 instâncias. A Figura 5.7 apresenta a união das otimalidades das Figuras 5.4, 5.5, e 5.6.

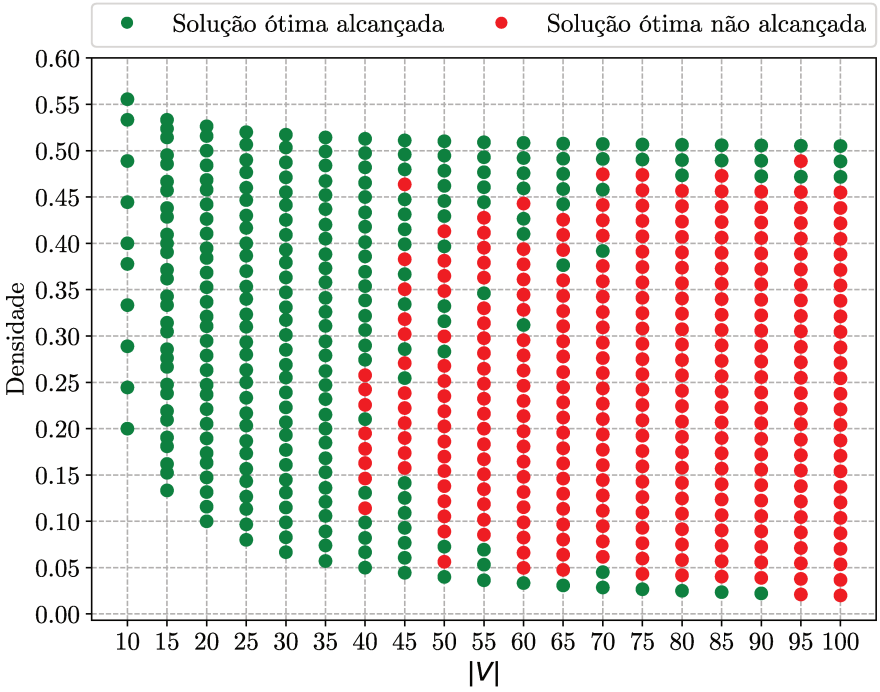


Figura 5.7: Otimalidades atingidas para as instâncias de $\mathcal{I}$.

Pela Figura 5.7, se percebe que os modelos propostos tenderam a apresentar maior dificuldade de obtenção de otimalidade para instâncias que contêm grafos com densidades intermediárias.

As Figuras 5.8 a 5.26 apresentam os valores das soluções obtidas pelas heurísticas para as instâncias de $\mathcal{I}_{10}$ até $\mathcal{I}_{100}$, além dos melhores limitantes primais (MP) e duais (MD) conhecidos para tais instâncias. Note que onde ocorre a sobreposição de pontos que indicam o melhor primal e melhor dual, trata-se do valor ótimo. Além disso, em cada uma das Figuras 5.8, 5.9 e 5.11, quando se vê um único gráfico é porque trata-se da sobreposição de todos os seis gráficos. Nesses casos, todas as instâncias foram solucionadas na otimalidade e todas as heurísticas produziram soluções ótimas para elas.

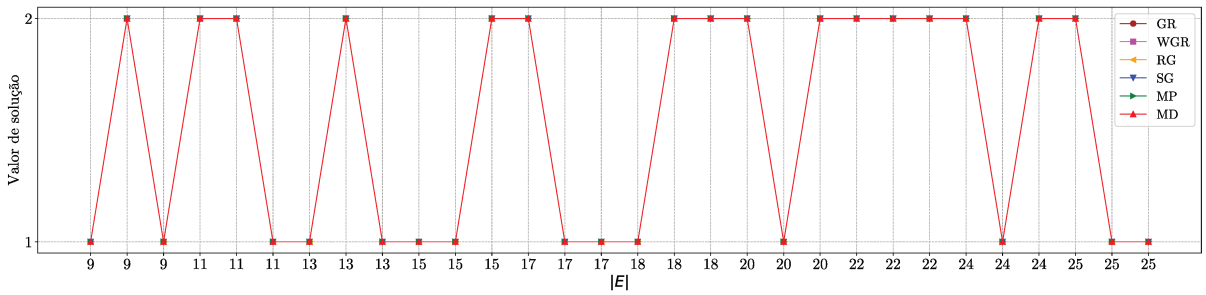


Figura 5.8: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{10}$.

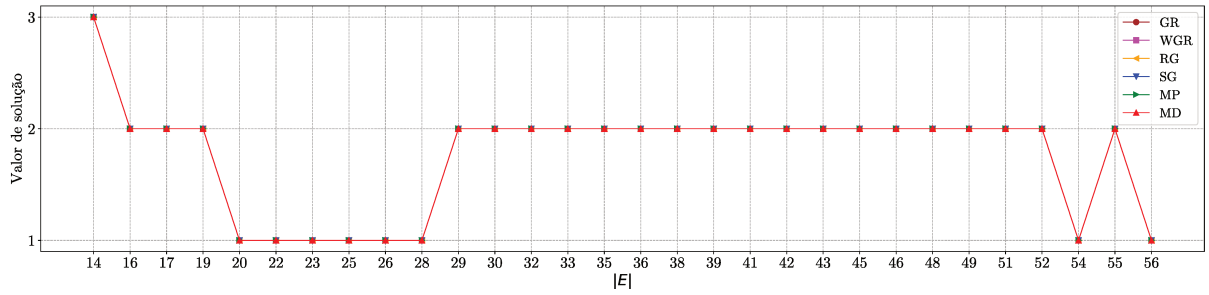


Figura 5.9: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{15}$.

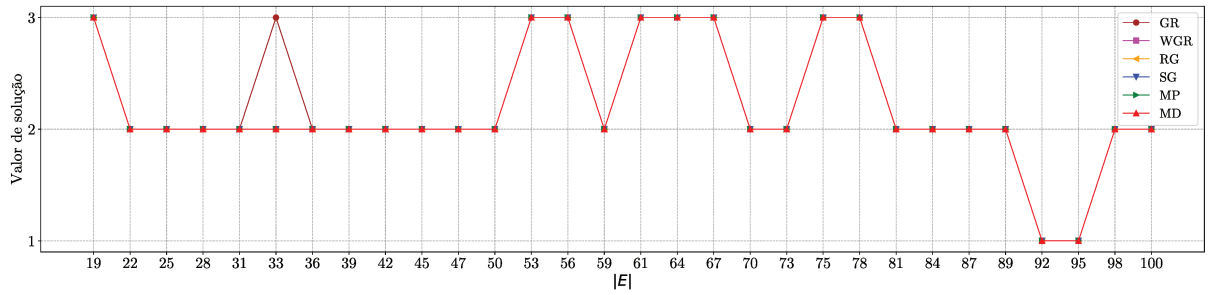


Figura 5.10: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{20}$.

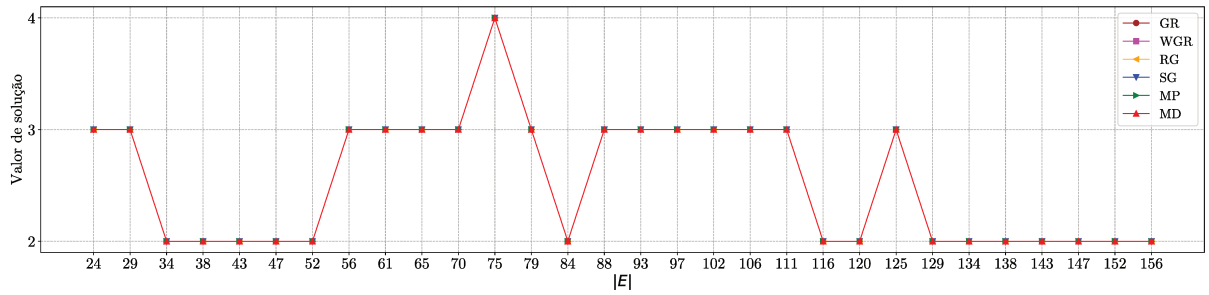


Figura 5.11: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{25}$.

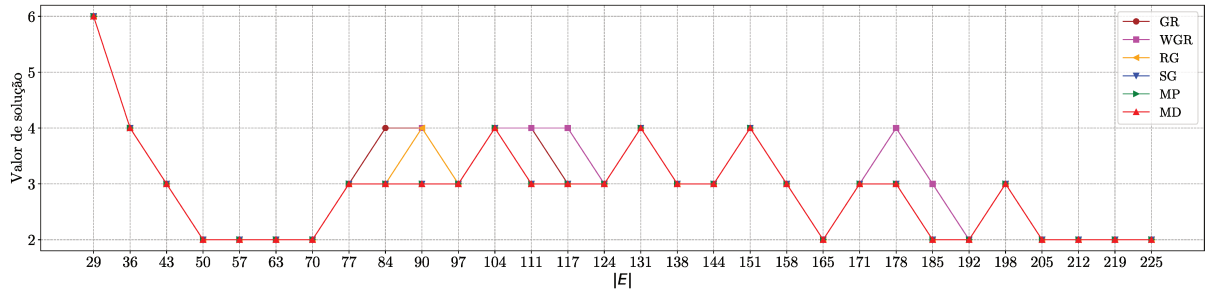


Figura 5.12: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{30}$.

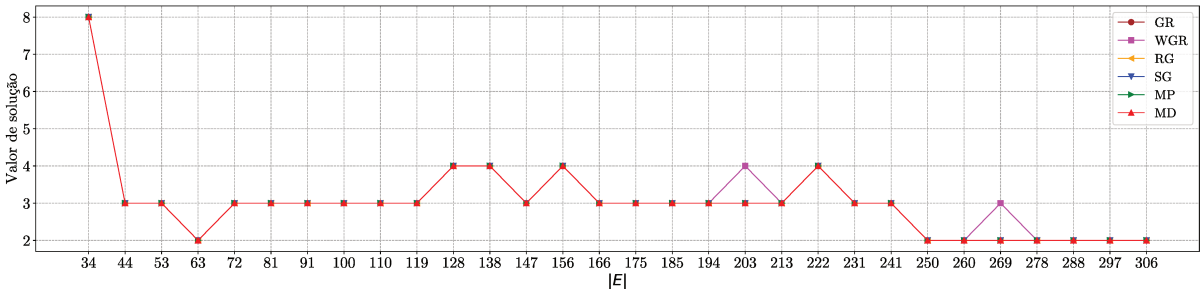


Figura 5.13: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{35}$.

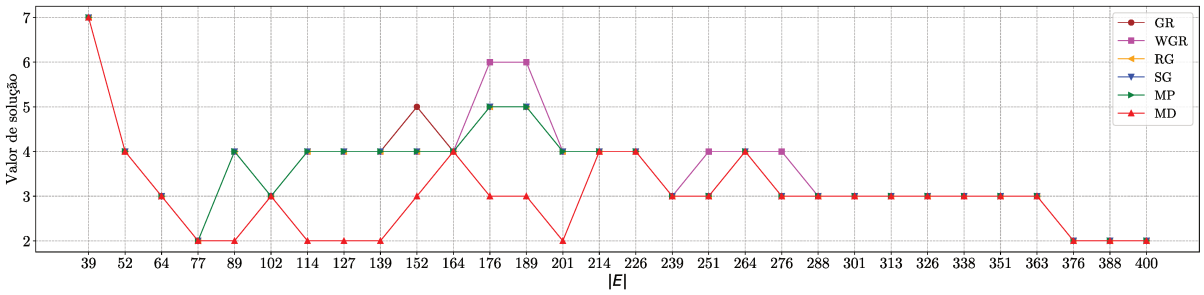


Figura 5.14: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{40}$.

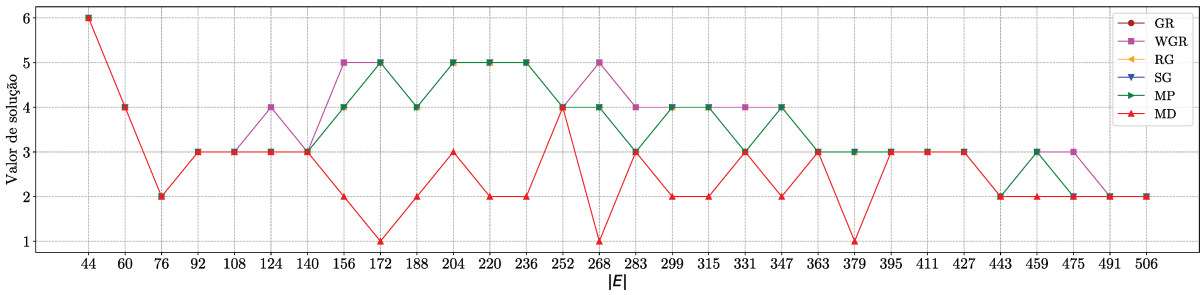


Figura 5.15: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{45}$.

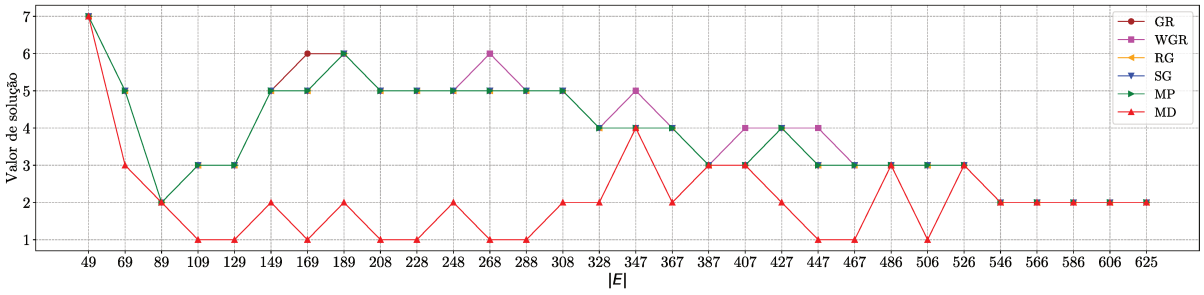


Figura 5.16: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{50}$.

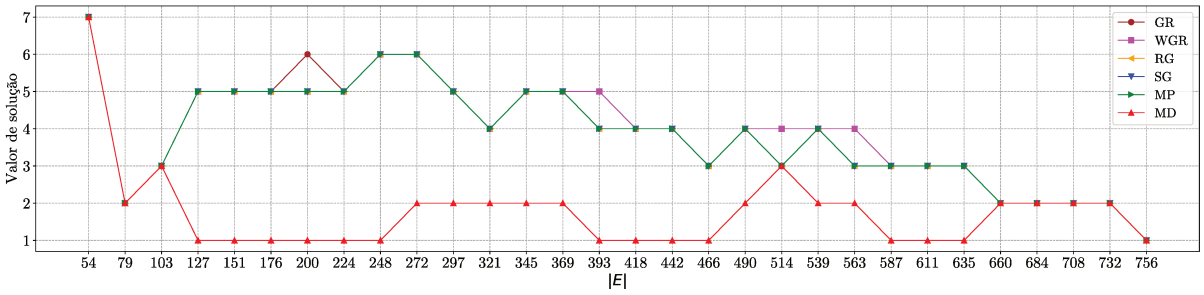


Figura 5.17: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{55}$.

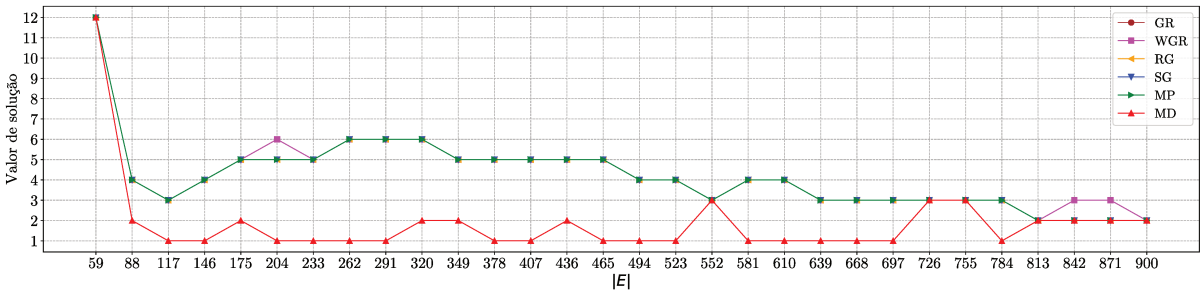


Figura 5.18: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{60}$.

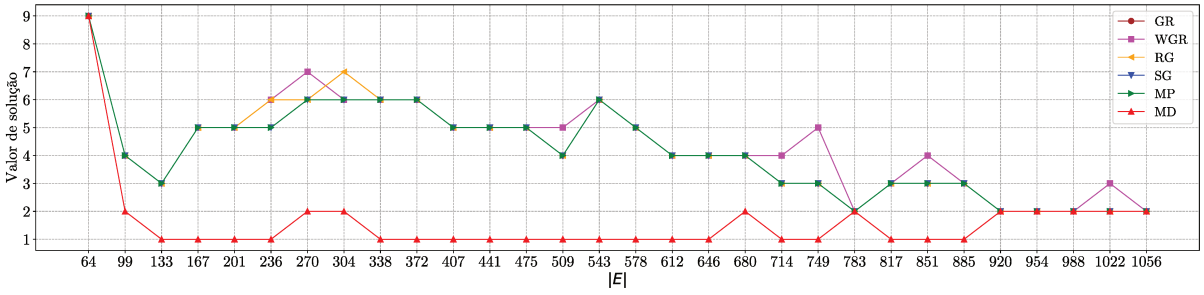


Figura 5.19: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{65}$.

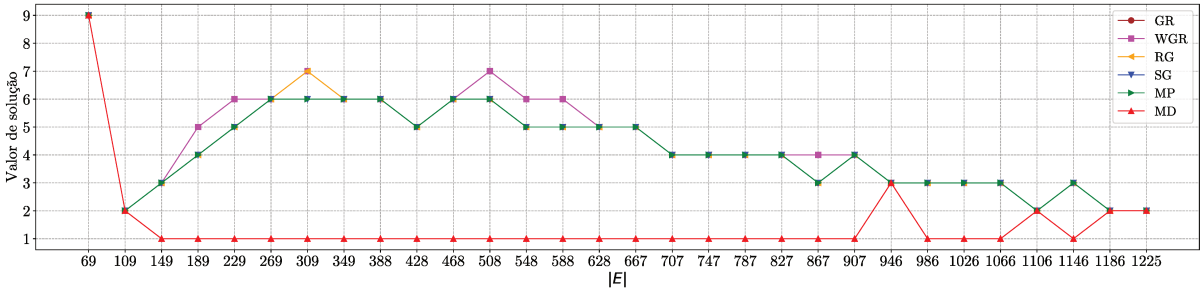


Figura 5.20: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{70}$.

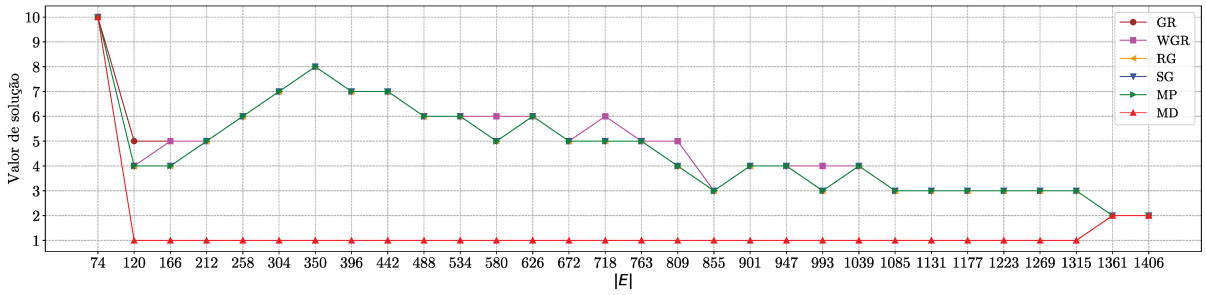


Figura 5.21: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{75}$.

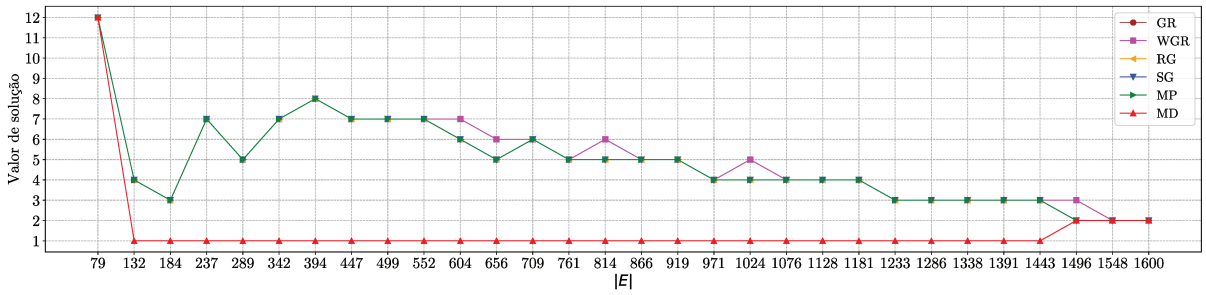


Figura 5.22: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{80}$.

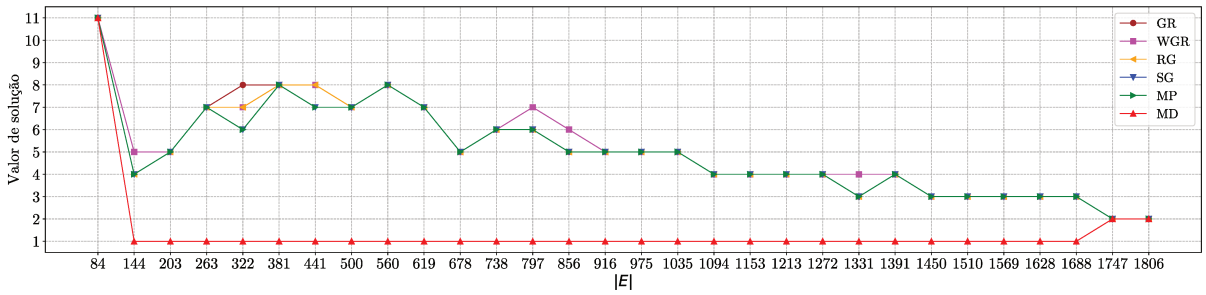


Figura 5.23: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{85}$.

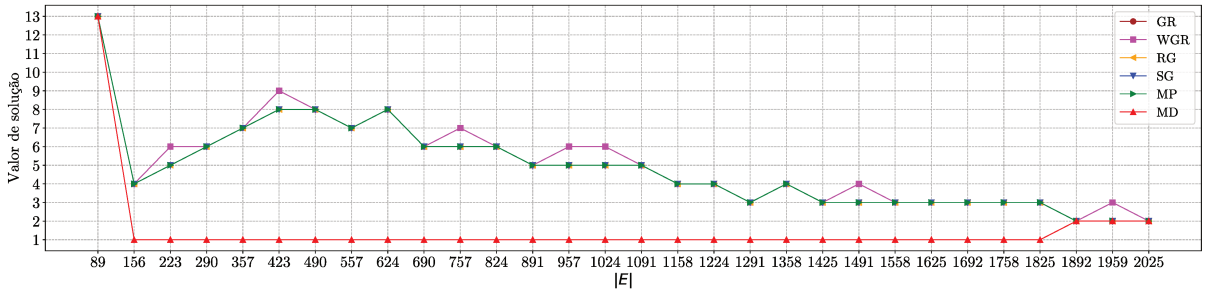


Figura 5.24: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{90}$.

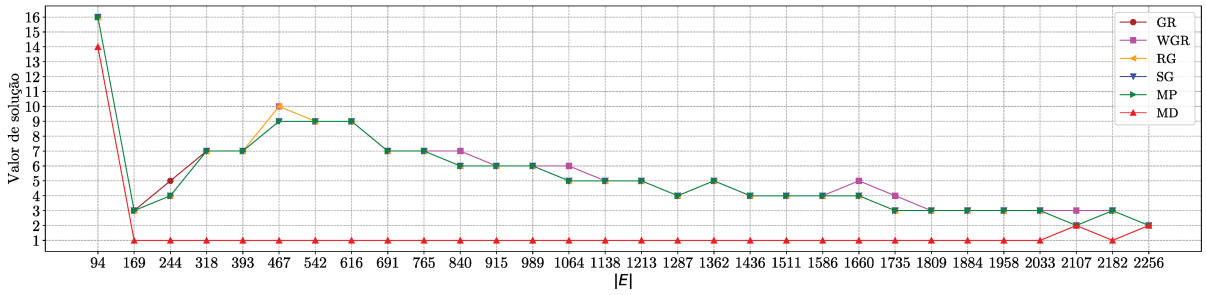


Figura 5.25: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{95}$.

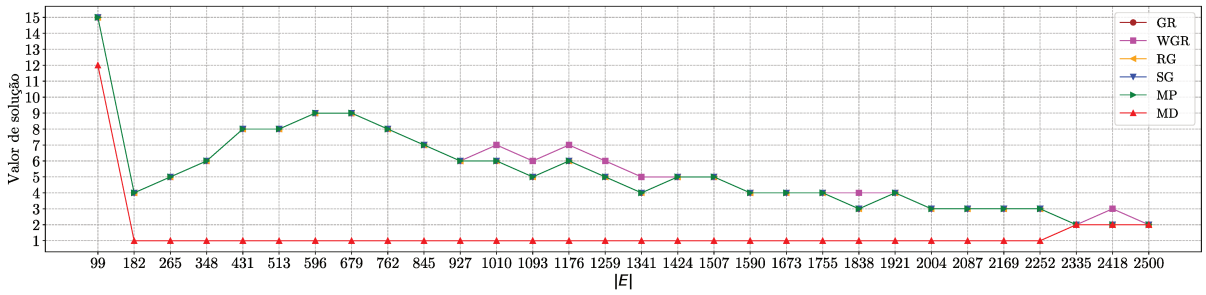


Figura 5.26: Valores das soluções produzidas pelas heurísticas e melhores limitantes conhecidos para as instâncias de $\mathcal{I}_{100}$.

Denote por $\mathcal{I}_{\text{ótimo}}$ o conjunto das 281 instâncias de $\mathcal{I}$ para as quais o valor ótimo nos é conhecido. Por meio das Figuras 5.8 a 5.26, verificamos que `GR`, `WGR`, `RG` e `SG` obtiveram soluções ótimas para 277, 258, 280 e 281 instâncias de $\mathcal{I}_{\text{ótimo}}$, respectivamente. Isso implica que todas as heurísticas obtiveram soluções ótimas para pelo menos 91.81% das instâncias para as quais conhecemos o valor ótimo, o que evidencia uma alta taxa de sucesso das heurísticas nesse quesito.

Considerando os casos em que as heurísticas não produziram soluções ótimas para as instâncias de $\mathcal{I}_{\text{ótimo}}$, a Tabela 5.4 apresenta, para cada heurística[14], a média, mediana e o desvio padrão das razões entre os valores das soluções não ótimas produzidas e os respectivos valores ótimos.

Tabela 5.4: Média, mediana e desvio padrão das razões entre os valores das soluções não ótimas e os respectivos valores ótimos.

	`GR`	`WGR`	`RG`
Média	1.3750	1.4022	1.3333
Mediana	1.3333	1.3333	1.3333
Desvio Padrão	0.0833	0.0894	0

Embora tenhamos, de acordo com Tabela 5.4, que as heurísticas produziram, em geral, soluções viáveis entre 33% e 41% acima do valor ótimo, a partir das Figuras 5.8 a 5.26, constatamos que todas as soluções não ótimas diferem por exatamente uma unidade do

[14]Com exceção da heurística `SG`, pois essa atingiu o valor ótimo para todas as instâncias de $\mathcal{I}_{\text{ótimo}}$.

valor ótimo. Portanto, mesmo quando as heurísticas não produziram soluções ótimas, elas atingiram valores muito próximos ao valor ótimo, nos casos em que este é conhecido.

Outro resultado relevante que destacamos é que para todas as instâncias de $\mathcal{I}$ testadas com os modelos de PLI, ao menos uma heurística produziu alguma solução com valor igual ao melhor limitante primal (veja Figuras 5.8 a 5.26). Na ampla maioria dos casos, ao menos duas heurísticas alcançaram tal feito. Isso implica que os modelos de PLI foram incapazes de melhorar os limitantes primais fornecidos pelas heurísticas, apesar de as heurísticas nativas do CPLEX terem sido ativadas em todas as execuções. Isso nos dá ainda outra evidência da alta qualidade das soluções primais obtidas pelas nossas heurísticas.

A fim de confirmar que as soluções viáveis iniciais que fornecemos ao CPLEX não enviesaram negativamente a busca por limitantes primais, conduzimos execuções adicionais do CPLEX sem o fornecimento dessas soluções. Os limitantes primais obtidos nesse caso nunca foram melhores do que aqueles apresentados nesta seção. Além disso, na maioria dos casos, o CPLEX só alcançou um limitante primal tão bom quanto aqueles produzidos pelas heurísticas após um intervalo de tempo de execução considerável. Portanto, sem o fornecimento das soluções viáveis iniciais, o CPLEX apresentou alguma dificuldade para encontrar bons limitantes primais.

Adiante, nota-se pelas Figuras 5.8 a 5.26 que para instâncias de $\mathcal{I}$ compostas por grafos com mesma quantidade de vértices, aquelas em que o valor ótimo não foi obtido têm limitantes primais, em geral, maiores do que os das soluções cujo valor ótimo foi alcançado. Ademais, à medida que $|V|$ cresce, crescem também os valores dos limitantes primais, principalmente nos casos em que a densidade é baixa ou intermediária, ao contrário do que acontece com os duais. Esses resultados sugerem que a maior dificuldade de obtenção do valor ótimo reside na busca por melhores limitantes duais, ao invés de limitantes primais. Isto indica que, não obstante termos maiores *gaps* de otimalidade para maiores valores de $|V|$, os limitantes primais produzidos pelas heurísticas tendem a ser bons mesmo nesses casos.

A seguir, analisamos o desempenho das fases das heurísticas para as instâncias de $\mathcal{I}_{\text{ótimo}}$. Para cada heurística, a Tabela 5.5 ilustra a quantidade de instâncias para as quais: alguma solução ótima foi gerada pela fase de construção em ao menos uma iteração; nenhuma solução ótima foi produzida na fase de construção em todas as iterações; nenhuma solução ótima foi formada na fase de construção, mas em ao menos uma iteração a busca local encontrou solução ótima.

Tabela 5.5: Desempenho das fases das heurísticas para as instâncias de $\mathcal{I}_{\text{ótimo}}$.

	GR	WGR	RG	SG
# Soluções ótimas obtidas na fase de construção	264	233	273	281
# Soluções ótimas não obtidas na fase de construção	17	48	8	0
# Soluções ótimas obtidas exclusivamente por busca local	13	25	7	0

Pela Tabela 5.5, se percebe que as fases de construção das heurísticas mostraram-se eficazes em obter soluções que são ótimas em pelo menos 83% das instâncias de $\mathcal{I}_{\text{ótimo}}$. Nesse quesito, a heurística SG foi aquela que obteve melhor desempenho, com 100% de êxito. Nos casos em que as fases de construção foram incapazes de atingir o valor ótimo,

as buscas locais apresentaram boa eficácia em aprimorar as soluções construídas e atingir o valor ótimo.

Agora, analisaremos os resultados obtidos para o conjunto $\mathcal{I}_{\text{viável}} = \mathcal{I} \setminus \mathcal{I}_{\text{ótimo}}$. Para cada uma das instâncias desse conjunto, o valor ótimo não é conhecido, mas existe ao menos uma solução viável produzida por alguma das heurísticas. Note que $|\mathcal{I}_{\text{viável}}| = 559$.

Verificamos que `GR`, `WGR`, `RG` e `SG` foram capazes de atingir os melhores valores de solução conhecidos para 531, 376, 412 e 545 instâncias de $\mathcal{I}_{\text{viável}}$, respectivamente. Nesse quesito, as heurísticas que `GR` e `SG` foram aquelas que apresentaram melhor desempenho, obtendo o melhor valor de solução conhecido para 95.35% e 97.49% das instâncias de $\mathcal{I}_{\text{viável}}$, respectivamente. `RG` e `WGR` ficaram atrás com taxa de sucesso de 73.70% e 67.62%, respectivamente.

Para os casos em que as heurísticas não produziram soluções com o melhor valor conhecido para as instâncias de $\mathcal{I}_{\text{viável}}$, a Tabela 5.6 apresenta, para cada heurística, a média, mediana e o desvio padrão das razões entres os valores das soluções produzidas e os respectivos melhores valores conhecidos.

Tabela 5.6: Média, mediana e desvio padrão das razões entre os valores das soluções que ficam aquém das melhores soluções conhecidas e os respectivos melhores valores conhecidos (para instâncias de $\mathcal{I}_{\text{viável}}$).

	GR	WGR	RG	SG
Média	1.1693	1.1655	1.2515	1.0554
Mediana	1.2000	1.1667	1.2500	1.0427
Desvio Padrão	0.0812	0.0909	0.1413	0.0416

De acordo com a Tabela 5.6, entre as quatro heurísticas, aquela que obteve as soluções mais próximas do melhor primal, quando não produziu soluções com o melhor valor conhecido, foi a `SG`.

A seguir, analisamos o desempenho das fases das heurísticas para as instâncias de $\mathcal{I}_{\text{viável}}$. Para cada heurística, a Tabela 5.7 ilustra a quantidade de instâncias para as quais: alguma solução com o melhor valor primal foi gerada pela fase de construção em ao menos uma iteração; nenhuma solução com o melhor valor primal foi produzida na fase de construção em todas as iterações; nenhuma solução com o melhor valor primal foi formada na fase de construção, mas em ao menos uma iteração a busca local encontrou solução com o melhor valor conhecido.

Tabela 5.7: Desempenho das fases das heurísticas para as instâncias de $\mathcal{I}_{\text{viável}}$.

	GR	WGR	RG	SG
# Melhor primal obtido na fase de construção	504	341	299	539
# Melhor primal não obtido na fase de construção	55	218	260	20
# Melhor primal obtido exclusivamente por busca local	27	35	113	6

Pela Tabela 5.7, se percebe que as fases de construção das heurísticas `GR` e `SG` foram aquelas que mostraram-se mais eficazes em produzir soluções com o melhor valor conhecido. Por outro lado, a pior das heurísticas nesse quesito, `RG`, alcançou 53% de efetividade e, portanto, de modo geral, todas as heurísticas obtiveram bom desempenho. Nos casos

em que as fases de construção foram incapazes de atingir o melhor primal, as buscas locais apresentaram eficácia em aprimorar as soluções construídas e alcançar soluções com o melhor valor conhecido.

Em resumo, todas heurísticas mostraram um desempenho muito bom no quesito de atingir soluções ótimas, ou próximas do valor ótimo, para as instâncias de $\mathcal{I}_{\text{ótimo}}$, mas apenas as heurísticas `GR` e `SG` obtiveram alta taxa de sucesso na obtenção de soluções com melhor valor conhecido para as instâncias de $\mathcal{I}_{\text{viável}}$. Com base nesses experimentos, concluímos que essas são as duas melhores heurísticas entre as quatro nos quesitos analisados.

Dessa maneira, prosseguimos com os testes estatísticos para determinar se há DES entre os valores das soluções produzidas pelas heurísticas para as instâncias de $\mathcal{I}$ e quais heurísticas produzem soluções significativamente melhores que as demais. Em todos os testes, utilizamos nível de confiança de 95%.

Seguindo os passos previstos na Seção 5.3, conduzimos o Teste de Iman-Davenport obtendo $\mathrm{F}_F = 24.6975$ e valor crítico igual a 2.6084. Como F_F é maior que o valor crítico, rejeitamos a hipótese nula e chegamos que existe DES entre os valores das soluções obtidas pelas heurísticas para as instâncias de todo o *benchmark* $\mathcal{I}$. Adiante, aplicamos o Teste de Nemenyi, cujo resultado é reportado pela Figura 5.27.

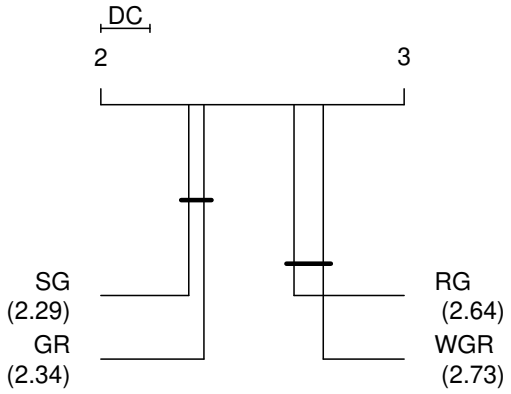


Figura 5.27: Resultado do Teste de Nemenyi.

Conforme ilustrado na Figura 5.27, a heurística `SG` foi aquela que obteve o melhor *rank médio* (2.29), seguida das heurísticas `GR`, `RG` e `WGR`. Além disso, não foi detectada DES entre os valores das soluções produzidas pelas heurísticas do par (`SG`, `GR`). O mesmo se aplica ao par (`RG`, `WGR`). Para todos os outros pares, o teste revelou que existe DES. Portanto, as heurísticas `SG` e `GR` foram aquelas que produziram soluções significativamente melhores para as instâncias de $\mathcal{I}$.

A fim de corroborar o resultado do Teste de Nemenyi para `SG` e `GR`, aplicamos o Teste de Wilcoxon para estas heurísticas, obtendo $z = -1.1132$. Como $z > -1.96$, não é possível rejeitar a hipótese nula e, portanto, chegamos que não foi detectada DES entre `SG` e `GR`.

Neste ponto, concluímos que embora todas as heurísticas tenham apresentado bom desempenho quanto à obtenção de boas soluções, os teste estatísticos demonstraram que a qualidade das soluções produzidas `SG` e `GR` são significativamente superiores às demais, mas estatisticamente indistinguíveis entre si com base nos seus *ranks* sobre $\mathcal{I}$. Portanto,

avançamos para a próxima etapa do experimento, a fim de verificar o desempenho dessas heurísticas nos quesitos de velocidade e robustez.

Nós implementamos o gerador de gráficos `mttt-plot` proposto em [50] e, de maneira análoga ao experimento relatado em [50], executamos as heurísticas `GR` e `SG` por 200 vezes com cada instância de $\mathcal{I}_{1000}$. Para cada instância, o alvo foi configurado como o piso de 1.1 vezes o valor da melhor solução conhecida. Cada execução foi finalizada assim que o alvo foi atingido.

A partir dos registros dos tempos das execuções da heurística `GR`, produzimos o gráfico `mttt-plot` apresentado na Figura 5.28. Cada ponto (x, y) do gráfico representa a probabilidade acumulada y da heurística `GR` ter atingido os respectivos alvos para todas as instâncias de $\mathcal{I}_{1000}$ em até x segundos. Para a produção da Figura 5.28, 10^4 pontos foram gerados.

Desse gráfico, lemos que se conduzirmos execuções sequenciais e únicas de `GR` com todas as instâncias em $\mathcal{I}_{1000}$, durante cerca de 1000 segundos ao todo, a probabilidade de `GR` atingir todos os alvos definidos é muito próxima de 1, o que nos dá uma média de apenas 33.3 segundos por instância.

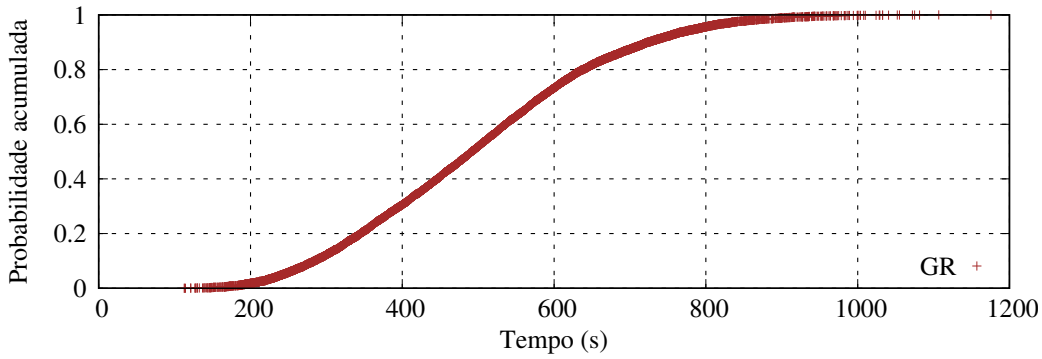


Figura 5.28: Gráfico `mttt-plot` para a heurística `GR` com as instâncias de $\mathcal{I}_{1000}$.

Similarmente, a Figura 5.29 mostra o `mttt-plot` para as execuções da heurística `SG` com 10^4 pontos. Desse gráfico, inferimos que a probabilidade de `SG` atingir todos os alvos definidos após cerca de 45 segundos é muito próxima de 1, de onde se deduz uma média de apenas 1.6 segundos por instância.

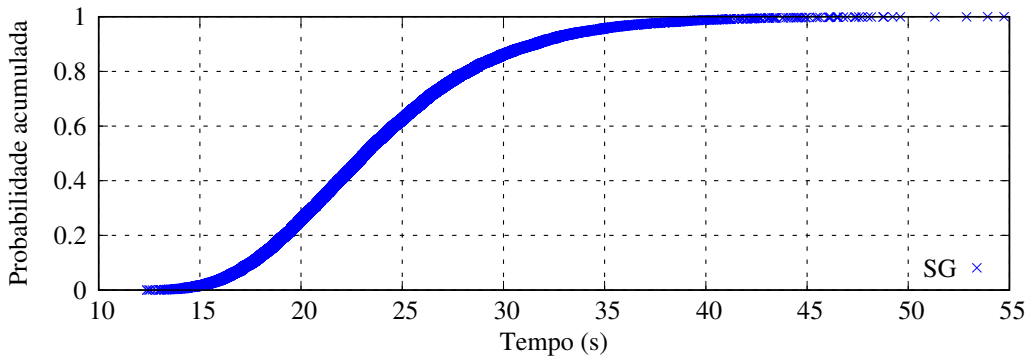


Figura 5.29: Gráfico `mttt-plot` para a heurística `SG` com as instâncias de $\mathcal{I}_{1000}$.

Considerando que o conjunto $\mathcal{I}_{1000}$ contém as maiores instâncias de $\mathcal{I}$, concluímos que ambas as heurísticas provaram-se ser rápidas e robustas para as 840 instâncias do nosso *benchmark*. Ademais, de acordo com os resultados obtidos, a heurística `SG` superou `GR` neste quesito e, portanto, a elegemos como a melhor heurística entre as quatro propostas.

A última etapa do experimento consistiu em executar `SG` com o conjunto de instâncias C (apresentado na Seção 5.3). Em [21], os autores utilizaram esse *benchmark* para testar a heurística `CGR` ali proposta. No nosso experimento, removemos dos grafos originais todos os vértices com grau igual a zero, assim como todas as arestas múltiplas. A heurística `SG` foi executada uma única vez com cada uma das 17 instâncias, utilizando os parâmetros calibrados da Tabela 5.3. Nós estabelecemos um tempo limite de execução de 1 hora por execução[15].

Os valores das soluções produzidas por `SG` são mostrados na Tabela 5.8, onde uma comparação com os resultados produzidos pelo algoritmo `CGR` de [21] pode ser vista. Verificamos que para as instâncias Facebook e Karate as soluções geradas pelo `CGR` têm valores iguais às das produzidas pela heurística `SG`. Para as demais 15 instâncias, `SG` produziu soluções melhores que o algoritmo `CGR`. Adicionalmente, empregamos os nossos modelos de `PLI` para as duas menores instâncias do *benchmark* C, Karate e Jazz, e verificamos que o valor ótimo de Karate é 3. Contudo, não obtivemos o valor ótimo da instância Jazz e, nesse caso, o melhor primal conhecido é 13.

Tabela 5.8: Comparativo das melhores soluções obtidas pelo `CGR` e `SG`.

Instância	|V|	|E|	`CGR`	`SG`
Amazon0302 [38]	262111	899792	36753	**28879**
BlogCatalog3 [59]	10312	333983	221*	**205**
BuzzNet [59]	101163	2763066	141	**126**
CA-AstroPh [38]	18771	198050	2174	**1913**
CA-CondMat [38]	23133	93439	2901	**2271**
CA-GrQc [38]	5241	14484	897	**781**
CA-HepPh [38]	12006	118489	1610	**1294**
CA-HepTh [38]	9875	25973	1531	**1179**
Cit-HepTh [38]	27769	352285	2996	**2548**
Delicious [38]	536408	1366136	19568	**13080**
Douban [59]	154908	327162	4832	**4079**
Facebook [38]	4039	88234	**10**	**10**
Jazz [44]	198	2742	15	**13**
Karate [44]	34	78	**3**	**3**
Last.fm [59]	1191812	4519340	27403	**5450**
Power Grid [44]	4941	6594	1367	**607**
YouTube2 [59]	1138499	2990443	83469*	**38468**

Para comparar o desempenho de nossas heurísticas com os resultados reais do algoritmo `CGR`, tivemos que implementar o `CGR`, já que o código-fonte original não foi disponibilizado. Apesar de a maioria dos resultados relatados em [21] terem sido confirmados,

[15]Nos experimentos conduzidos em [21], os tempos das execuções da heurística `CGR` para as instâncias de C não foram reportados.

obtivemos o valor de solução 221 para BlogCatalog3 com função limiar $t(v) = \lceil 0.5 \cdot d(v) \rceil$, enquanto que o valor da solução de 99 (relatado naquele artigo) é, na realidade, atingível pelo `CGR`, mas com o emprego da função limiar $t(v) = \lceil 0.4 \cdot d(v) \rceil$, conforme nossa comunicação com os autores. Da mesma forma, o valor de solução 83469 para YouTube2 contrasta com o valor 33046 relatado em [21] e, infelizmente, nenhum esclarecimento pôde ser deduzido a partir dos dados contidos no artigo.

Portanto, podemos concluir que `SG` superou a melhor heurística existente até este momento para o `PAP`, com o conjunto C de instâncias compostas por redes sociais reais.

Capítulo 6

Considerações Finais

Neste trabalho, objetivamos um estudo computacional do PAP com o intuito de produzir novas técnicas eficazes para resolver o problema. Desenvolvemos três técnicas de pré-processamento de instâncias, as duas primeiras formulações de programação linear inteira e quatro novas heurísticas para o PAP. A corretude dos modelos de PLI foi devidamente demonstrada, assim como provas de corretude das técnicas de pré-processamento e heurísticas, acompanhadas de suas respectivas análises de complexidade. Com o intuito de conduzir experimentos computacionais, produzimos também um novo *benchmark* de 840 instâncias do PAP que simulam redes sociais reais.

Mostramos que, utilizando os modelos exatos, é possível obter soluções ótimas para instâncias do PAP com grafos de até 100 vértices. Além disso, 281 instâncias foram solucionadas de maneira exata. Para essas instâncias, constatamos que as implementações das heurísticas propostas mostraram-se eficazes em obter soluções ótimas ou com valor próximo do valor ótimo. Para as demais instâncias, verificamos que as heurísticas GR e SG foram aquelas que geraram mais soluções com valor igual ao melhor valor conhecido.

Apresentamos ainda evidências de que mesmo nos casos em que o valor ótimo não é conhecido, os valores das soluções produzidas pelas heurísticas consistem em bons limitantes primais. Averiguamos também que, de modo geral, as fases de construção das heurísticas produziram boas soluções. Para os casos em que isso não ocorreu, comprovamos que as respectivas buscas locais foram eficazes em melhorar as soluções construídas.

Por meio de testes estatísticos consagrados na literatura, verificamos que os valores das soluções produzidas pelas heurísticas GR e SG, para as instâncias do nosso *benchmark*, apresentaram diferença estatisticamente significativa em relação aos valores das soluções obtidas pelas demais heurísticas. Como tais heurísticas são aquelas que produziram as melhores soluções, elegemos SG como nossa melhor heurística devido ao seu desempenho bastante superior ao da GR no teste de eficiência e robustez. Por fim, mostramos que SG supera o algoritmo CGR (atual estado da arte) para um *benchmark* de instâncias de redes sociais reais da literatura [21].

Diante do que foi exposto, concluímos que os objetivos da pesquisa relatada nesta Dissertação foram alcançados.

Salientamos que a relevância do conhecimento produzido neste trabalho não está restrita ao aspecto teórico do problema. De acordo com [21], o mecanismo de propagação de informações adotado para o PAP é uma hipótese razoável de como as informações se

espalham, tendo em vista observações experimentais em redes sociais *online* descritas na literatura. Além disso, com o crescimento significativo dessas redes (em escala e volume), os temas relacionados à disseminação de informações têm recebido cada vez mais atenção da academia e da indústria [45]. Dessa maneira, concluímos que o PAP está diretamente atrelado aos interesses e problemas do mundo real, assim como as abordagens que desenvolvemos para resolvê-lo. Portanto, tais abordagens possuem potencial de aplicabilidade em cenários reais compatíveis com o mecanismo de propagação empregado no PAP.

Note que é possível estabelecer variantes do PAP para diferentes contextos. Suponha, por exemplo, que a influência que um indivíduo u tem sobre outro indivíduo v pode ser distinta daquela que v tem sobre u. Nesse caso, podemos definir uma versão do PAP em que uma rede social é representada por um grafo direcionado e ponderado, de modo que o peso do arco (u, v) quantifica a influência de u sobre v.

Em trabalhos futuros, pretendemos investigar variações do PAP tais como o exemplo supracitado além de problemas relacionados como o PESP e o MESP (veja Seção 3.4). Visamos também estudar uma versão do TSSP, proposta em [28], na qual assume-se que um disseminador propaga a informação somente durante um intervalo restrito de rodadas consecutivas. Nesse caso, a modelagem de como a propagação ocorre possui semelhanças com o processo de contágio epidemiológico de patógenos. Para esses problemas, esperamos que as nossas heurísticas e formulações produzidas para o PAP possam ser adaptadas ou sirvam de ponto de partida para o desenvolvimento de novos métodos de resolução.

Além disso, planejamos aprimorar os nossos algoritmos para o PAP e explorar novas técnicas baseadas em outras meta-heurísticas bem sucedidas, conforme [29]. Adicionalmente, planejamos conduzir uma investigação de técnicas de identificação de *comunidades* em redes sociais [9] com o intuito de criar métodos de decomposição de instâncias do PAP. Finalmente, pretendemos fazer um estudo mais aprofundado dos modelos de PLI propostos, utilizando abordagens de *combinatória poliédrica* [57], a fim de obter novas desigualdades que fortaleçam as formulações para que seja possível obter soluções exatas para instâncias maiores que aquelas aqui reportadas.

Referências Bibliográficas

[1] E. Ackerman, O. Ben-Zwi, and G. Wolfovitz. Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, 411(44-46):4017–4022, Oct. 2010.

[2] L. A. Adamic, T. M. Lento, E. Adar, and P. C. Ng. Information evolution in social networks. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 473–482, 2016.

[3] R. M. Aiex, M. G. Resende, and C. C. Ribeiro. TTTplots: a Perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2007.

[4] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, Jan. 2002.

[5] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic. The role of social networks in information diffusion. In *Proceedings of the 21st International Conference on World Wide Web*, pages 519–528, 2012.

[6] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[7] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[8] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. *F-Race and Iterated F-Race: An Overview*, pages 311–336. Springer, 2010.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[10] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*, volume 290. Macmillan, 1976.

[11] B. Calvo and G. Santafé. SCMAMP: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, 8:248–256, 2016.

[12] A. Campan, T. Truta, and M. Beckerich. Fast dominating set algorithms for social networks. In *26th Modern Artificial Intelligence and Cognitive Science Conference*, volume 1353, pages 55–62. Central Europe Workshop Proceedings, Apr. 2015.

[13] N. Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.

[14] W. Chen, L. Lakshmanan, and C. Castillo. Information and influence propagation in social networks. *Synthesis Lectures on Data Management*, 5:1–177, Oct. 2013.

[15] J. Cheriyan and G. P. Sajeev. Spreadmax: a scalable cascading model for influence maximization in social networks. In *2018 International Conference on Advances in Computing, Communications and Informatics*, pages 1290–1296. Institute of Electrical and Electronic Engineers, Sept. 2018.

[16] F. Cicalese, G. Cordasco, L. Gargano, M. Milanič, J. Peters, and U. Vaccaro. Spread of influence in weighted networks under time and budget constraints. *Theoretical Computer Science*, 586:40–58, June 2015.

[17] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*, volume 271. Springer, 2014.

[18] G. Cordasco, L. Gargano, and A. Rescigno. Threshold-bounded dominating set with incentives. In *International Conference on new Trends in Computing Sciences*, volume 2243, pages 65–76. Central Europe Workshop Proceedings, Sept. 2018.

[19] G. Cordasco, L. Gargano, A. Rescigno, and U. Vaccaro. Evangelism in social networks: Algorithms and complexity. *Networks*, 71(4):346–357, Jul. 2018.

[20] G. Cordasco, L. Gargano, and A. A. Rescigno. Influence propagation over large scale social networks. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1531–1538. Association for Computing Machinery, 2015.

[21] G. Cordasco, L. Gargano, and A. A. Rescigno. Active influence spreading in social networks. *Theoretical Computer Science*, 764:15–29, 2019.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2009.

[23] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 13:339–347, 1951.

[24] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 1989.

[25] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, Dec. 2006.

[26] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[27] P. Festa and M. G. Resende. GRASP: An annotated bibliography. In *Essays and Surveys in Metaheuristics*, pages 325–367. Springer, 2002.

[28] L. Gargano, P. Hell, J. Peters, and U. Vaccaro. Influence diffusion in social networks under time window constraints. In *Structural Information and Communication Complexity*, pages 141–152, Cham, 2013. Springer.

[29] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[30] F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.

[31] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., USA, 1997.

[32] IBM. IBM ILOG CPLEX Optimization Studio v12.10.0 documentation. Available on `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.10.0/COS_KC_home.html`. Accessed December 14, 2020.

[33] R. L. Iman and J. M. Davenport. Approximations of the critical region of the Fbietkan statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595, 1980.

[34] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311, 1984.

[35] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146. Association for Computing Machinery, 2003.

[36] L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

[37] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, USA, 2005.

[38] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. Available on `http://snap.stanford.edu/data`.

[39] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[40] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, USA, 1989.

[41] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, Jan. 1998.

[42] P. Nemenyi. *Distribution-free Multiple Comparisons*. Princeton University, 1963.

[43] NetworkX. Barabási–albert graph generator. Available on `www.networkx.org/documentation/latest/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html`. Accessed December 14, 2020.

[44] M. Newman. Network data, 2013. Available on `http://www-personal.umich.edu/~mejn/netdata/`.

[45] S. Peng, Y. Zhou, L. Cao, S. Yu, J. Niu, and W. Jia. Influence analysis in social networks: A survey. *Journal of Network and Computer Applications*, 106:17–32, 2018.

[46] F. C. Pereira. Heuristics and Integer Programs for the Perfect Awareness Problem, 2021. Available on `https://github.com/fcpereira97/heuristics_and_integer_programs_for_the_perfect_awareness_problem`.

[47] F. C. Pereira, C. C. de Souza, and P. J. de Rezende. The Perfect Awareness Problem - Benchmark Instances, 2020. Available on `www.ic.unicamp.br/~cid/Problem-instances/Perfect-Awareness-Problem/`.

[48] L. Qiu, W. Jia, J. Yu, X. Fan, and W. Gao. Phg: A three-phase algorithm for influence maximization based on community structure. *IEEE Access*, 7:62511–62522, 2019.

[49] M. G. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of Metaheuristics*, pages 283–319. Springer, USA, 2010.

[50] A. Reyes and C. C. Ribeiro. Extending time-to-target plots to multiple instances. *International Transactions in Operational Research*, 25(5):1515–1536, 2018.

[51] K. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 2016.

[52] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[53] K. Tanınmış, N. Aras, and I. Altınel. Influence maximization with deactivation in social networks. *European Journal of Operational Research*, 278(1):105–119, 2019.

[54] E. Tardos and J. Kleinberg. *Algorithm Design*. Pearson, 2005.

[55] F. Wilcoxon. *Individual Comparisons by Ranking Methods*, pages 196–202. Springer, 1992.

[56] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[57] L. A. Wolsey. *Integer programming*. Wiley-Interscience, USA, 1998.

[58] L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[59] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009. Available on `http://socialcomputing.asu.edu`.

Apêndice A

Prova Alternativa de NP-dificuldade do PAP

A prova de NP-dificuldade do PAP apresentada neste apêndice é baseada em uma redução a partir do MDSP, exibida na demonstração do Teorema A.1 abaixo. Dado um grafo não direcionado $G = (V, E)$, dizemos que $S \subseteq V$ é um *conjunto dominante*, se para todo $v \in V$ vale que $v \in S$ ou $\exists\, u \in S$ tal que $\{v, u\} \in E$.

Problema 6 (MDSP)**.** *Dada uma instância composta por um grafo não direcionado $G = (V, E)$, o objetivo do* MDSP *é encontrar um conjunto dominante de tamanho mínimo.*

Vê-se que qualquer conjunto dominante é uma solução viável para o MDSP, mas somente um conjunto dominante de tamanho mínimo é uma solução ótima.

Sejam $I = (G = (V, E), t)$ uma instância do PAP tal que $\forall v \in V\, t(v) = d(v)$ e S uma solução viável de I. O Lema A.1 nos diz que a propagação do PAP a partir de S é encerrada até a rodada número 2. O Lema A.2 afirma que ao final da propagação iniciada por S, todo vértice conhecedor que não pertence a S possui algum vizinho em S. Por fim, o Teorema A.1 estabelece que o PAP é um problema NP-difícil.

Lema A.1. *Se $I = (G = (V, E), t)$ é uma instância do PAP tal que $\forall v \in V\, t(v) = d(v)$ e S é uma solução viável de I, então $Dis_G[S] = Dis_G[S, \rho]$ com $\rho \leq 2$.*

Demonstração. Sejam $I = (G = (V, E), t)$ uma instância do PAP tal que $\forall v \in V\, t(v) = d(v)$ e S uma solução viável de I. Pela definição do problema, temos que a propagação do PAP a partir de S é finalizada em alguma rodada $\rho \geq 1$, quando $\mathrm{Dis}_G[S, \rho] = \mathrm{Dis}_G[S, \rho - 1]$.

Agora suponha por absurdo que $\rho \geq 3$. Logo, $\mathrm{Dis}_G[S, \rho - 1] \neq \mathrm{Dis}_G[S, \rho - 2]$ e, portanto, existe $v \in V$ tal que $v \in \mathrm{Dis}_G[S, \rho - 1]$ e $v \notin \mathrm{Dis}_G[S, \rho - 2]$. Como v se tornou disseminador na rodada $\rho - 1$ e $t(v) = d(v)$, então $N_G(v) \subseteq \mathrm{Dis}_G[S, \rho - 2]$.

Além disso, como v se tornou disseminador na rodada $\rho - 1$, existe $u \in N_G(v)$ tal que $u \in \mathrm{Dis}_G[S, \rho - 2]$ e $u \notin \mathrm{Dis}_G[S, \rho - 3]$. Como u se tornou disseminador na rodada $\rho - 2$ e $t(u) = d(u)$, $N_G(u) \subseteq \mathrm{Dis}_G[S, \rho - 3]$.

Observe que $v \in N_G(u)$, logo $v \in \mathrm{Dis}_G[S, \rho - 3]$. Mas $v \notin \mathrm{Dis}_G[S, \rho - 2]$ e consequentemente $v \notin \mathrm{Dis}_G[S, \rho - 3]$, contradição. Assim, chegamos que $\rho \leq 2$. □

Lema A.2. *Se $I = (G = (V, E), t)$ é uma instância do PAP tal que $\forall v \in V\, t(v) = d(v)$ e S é uma solução viável de I, então $\forall\, v \in Con_G[S] \setminus S$ temos que $N_G(v) \cap S \neq \emptyset$.*

Demonstração. Sejam $I = (G = (V, E), t)$ uma instância do PAP tal que $\forall v \in V\, t(v) = d(v)$ e S uma solução viável de I. Como $\forall v \in V\, t(v) = d(v)$, então, segue-se do Lema A.1 que a propagação do PAP a partir de S termina em uma rodada ρ com $1 \leq \rho \leq 2$. Seja $v \in V$, tal que $v \in Con_G[S] \setminus S$.

Se $\rho = 1$, então $\text{Dis}_G[S] = \text{Dis}_G[S, 1] = \text{Dis}_G[S, 0] = S$. Todavia, $\text{Con}_G[S] = \text{Dis}_G[S] \cup \{u : N_G(u) \cap \text{Dis}_G[S] \neq \emptyset\}$ e, portanto, $\text{Con}_G[S] = S \cup \{u : N_G(u) \cap S \neq \emptyset\}$. Como $v \notin S$, $v \in \{u : N_G(u) \cap S \neq \emptyset\}$. Logo, $N_G(v) \cap S \neq \emptyset$.

Se $\rho = 2$, então $\text{Dis}_G[S] = \text{Dis}_G[S, 2] = \text{Dis}_G[S, 1] = S \cup X$, onde $X = \{u : |N_G(u) \cap S| \geq t(u)\}$. Porém, $\text{Con}_G[S] = \text{Dis}_G[S] \cup \{u : N_G(u) \cap \text{Dis}_G[S] \neq \emptyset\}$ e, portanto, $\text{Con}_G[S] = S \cup X \cup \{u : N_G(u) \cap (S \cup X) \neq \emptyset\}$. Como $v \notin S$, $v \in X$ ou $v \in \{u : N_G(u) \cap (S \cup X) \neq \emptyset\}$. Se $v \in X$, então $N_G(v) \in S$. Se $v \in \{u : N_G(u) \cap (S \cup X) \neq \emptyset\}$, então existe $w \in N_G(v)$ tal que $w \in X$ ou $w \in S$. Note que, como $v \notin S$ e $\forall\, u \in N_G(v)\, t(u) = d(u)$, temos $N_G(v) \cap X = \emptyset$ e, portanto, $w \in S$.

Em ambos os casos, chegamos que $N_G(v) \cap S \neq \emptyset$. □

Teorema A.1. *O PAP é* NP*-difícil.*

Demonstração. Seja I uma instância do MDSP composta por um grafo não direcionado $G = (V, E)$. A partir de I, construa uma instância $I' = (G, t)$ do PAP, tomando $t(v) = d(v)$ para todo $v \in V$.

Primeiramente, vamos mostrar que uma solução S é viável para I se, e somente se, é viável para I'. Tome uma solução $S \subseteq V$ que é viável para I, isto é, $\forall\, v \in V$ vale que $v \in S$ ou $\exists\, u \in S$, com $\{v, u\} \in E$. Assim, $V = S \cup \{u : N_G(u) \cap S \neq \emptyset\}$ e, portanto, $\text{Con}_G[S] = V$, pois no PAP todo disseminador permanece nesse estado até o fim da propagação. Dessa maneira, S é viável para I'. Agora, tome uma solução $S' \subseteq V$ que é viável para I', ou seja, $\text{Con}_G[S'] = V$. Como $\forall\, v \in V\, t(v) = d(v)$, pelo Lema A.2, $\forall\, v \in V \setminus S'$ temos que $N_G(v) \cap S' \neq \emptyset$. Logo, $\forall\, v \in V\ v \in S'$ ou $\exists\, u \in S'$, com $\{v, u\} \in E$. Portanto, S' é viável para I.

Mostraremos a seguir que uma solução viável S é ótima para I se, e somente se, é ótima para I'. Tome uma solução $S \subseteq V$ que é ótima para I. Suponha por absurdo que S não é ótima para I'. Então, existe uma solução ótima $S' \subseteq V$ para I' tal que $|S'| < |S|$. Mas S' é viável para I e além disso $|S'| < |S|$. Por conseguinte, S não é ótima para I, contradição. Agora, tome uma solução $S' \subseteq V$ que é ótima para I'. Suponha por absurdo que S' não é ótima para I. Então, existe uma solução ótima $S'' \subseteq V$ para I tal que $|S''| < |S'|$. Mas S'' é viável para I' e além disso $|S''| < |S'|$, então S' não é ótima para I', contradição.

Assim, apresentamos uma redução do MDSP para o PAP. Como o tamanho de I' é polinomial no tamanho de I, a construção de I' pode ser feita em tempo polinomial no tamanho de G. Portanto, temos uma redução de tempo polinomial do MDSP para o PAP. Como o MDSP é um problema NP-difícil, o PAP também é NP-difícil. □