



Universidade Estadual de Campinas
Instituto de Computação



Natanael Ramos

Um Estudo Computacional do Problema do Brigadista em Grafos

CAMPINAS
2018

Natanael Ramos

Um Estudo Computacional do Problema do Brigadista em Grafos

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

Este exemplar corresponde à versão final da Dissertação defendida por Natanael Ramos e orientada pelo Prof. Dr. Cid Carvalho de Souza.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): CNPq, 133728/2016-1

ORCID: <https://orcid.org/0000-0002-3546-9787>

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

R147e Ramos, Natanael, 1994-
Um estudo computacional do problema do brigadista em grafos / Natanael Ramos. – Campinas, SP : [s.n.], 2018.

Orientador: Cid Carvalho de Souza.

Coorientador: Pedro Jussieu de Rezende.

Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação inteira. 3. Matheurística. 4. Heurística computacional. I. Souza, Cid Carvalho de, 1963-. II. Rezende, Pedro Jussieu de, 1955-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: A computational study of the Firefighter Problem on graphs

Palavras-chave em inglês:

Combinatorial optimization

Integer programming

Matheuristic

Computer heuristics

Área de concentração: Ciência da Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora:

Cid Carvalho de Souza [Orientador]

Luciana Salete Buriol

Fábio Luiz Usberti

Data de defesa: 13-04-2018

Programa de Pós-Graduação: Ciência da Computação



Universidade Estadual de Campinas
Instituto de Computação



Natanael Ramos

Um Estudo Computacional do Problema do Brigadista em Grafos

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza
Instituto de Computação - UNICAMP (orientador)
- Profa. Dra. Luciana Salete Buriol
Instituto de Informática - UFRGS
- Prof. Dr. Fábio Luiz Usberti
Instituto de Computação - UNICAMP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 13 de abril de 2018

Dedicatória

À minha família e em memória do meu grande amigo Mauro.

*“If you must work, work to leave some part
of you on this earth.”*
Keaton Henson

Agradecimentos

Primeiramente, gostaria de expressar minha gratidão aos meus orientadores, Prof. Cid e Prof. Pedro, pela paciência, confiança, por todo o conhecimento passado e pelo compartilhamento de experiências. Durante esses anos, cresceu a minha admiração por vocês e sem sombra de dúvidas, seu aconselhamento foi fundamental para meu crescimento profissional e pessoal.

Agradeço à minha família pelo apoio, motivação e por terem me dado a oportunidade de continuar meus estudos, apesar de todas as dificuldades. Sem vocês, eu não teria chegado até aqui.

Sou grato a todos os meus amigos e colegas, os quais apoiaram direta ou indiretamente meus estudos. Em especial às incríveis amizades que pude fazer no LOCo, as quais me ajudaram no meu crescimento profissional e pessoal, além de proporcionar inúmeros momentos de alegria e lazer, que tornaram essa experiência acadêmica menos intimidadora e mais divertida.

Agradeço aos servidores e professores do Instituto de Computação da Unicamp, que tornam possível dia a dia um ambiente acadêmico de excelência o qual pude usufruir durante o mestrado.

Finalmente, agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (Cnpq) pelo apoio financeiro que tornou possível a realização dessa pesquisa (bolsa 133728/2016-1).

Resumo

O Problema do Brigadista em Grafos (FFP do inglês *The Firefighter Problem* é um modelo determinístico e em tempo discreto para simular a propagação e contenção de incêndios em grafos. Ele pode ser descrito da seguinte forma. Na entrada, é dado um inteiro D representando a quantidade de brigadistas disponíveis, um grafo não direcionado e não ponderado $G = (V, E)$ e um subconjunto de vértices $B \subseteq V$, os focos de incêndio. Então, inicia-se nos elementos de B um processo iterativo de propagação e contenção de fogo através dos vértices de G , em rodadas discretas, o qual termina quando não existem mais vértices que possam ser queimados, ou seja, quando o fogo está contido. O objetivo ao resolver o FFP é maximizar o número de vértices não queimados quando o fogo é contido, com a restrição de que no máximo D vértices podem ser protegidos contra o fogo por rodada. Aplicações práticas do FFP, além da obtenção de estratégias para minimização de danos causados por incêndios, podem ser encontradas em áreas como controle de doenças e segurança em redes.

O FFP é NP-difícil e métodos heurísticos para lidar com o problema foram relatados previamente na literatura. Nesta dissertação, primeiramente, apresentamos melhorias feitas na primeira formulação PLI proposta para o FFP através de técnicas de pre-processamento e agregação de restrições. Em seguida, descrevemos novas heurísticas gulosas e introduzimos uma nova matheurística para o FFP, uma abordagem que se baseia na interoperação entre meta-heurísticas e programação matemática. Experimentos foram conduzidos em um *benchmark* público tanto para configuração de parâmetros quanto para análise de desempenho, através de comparação dos resultados obtidos com aqueles publicados anteriormente. Com respeito às modificações no modelo PLI, um *speedup* de aproximadamente 2 em média foi alcançado. Observamos que as modificações feitas podem levar à geração de soluções infactíveis, mas conseguimos demonstrar que é possível tornar tais soluções factíveis em tempo polinomial. Em referência às heurísticas, essas foram executadas seguindo uma metodologia para construir uma solução na qual escolhas gulosas aleatorizadas são realizadas para selecionar quais vértices serão defendidos, de acordo com conceitos introduzidos na meta-heurística GRASP. Comparando essas heurísticas com as que foram propostas por trabalhos anteriores, observamos que duas das nossas estão entre as cinco melhores na maioria dos casos. Com relação à matheurística, através de uma análise estatística rigorosa, verificamos que existe diferença estatisticamente significativa entre nossa estratégia e as demais, ao mesmo tempo que nossa matheurística conseguiu resultados melhores na maioria das instâncias.

Abstract

The firefighter problem (FFP) is a deterministic discrete-time model for the spread and containment of fire on a graph. Such problem is described as follows. As its inputs, there is an integer D representing the number of available firefighters, an undirected and unweighted graph $G = (V, E)$ and a subset of vertices $B \subseteq V$, the fire outbreaks. Then, an iterative process of fire propagation and containment through the vertices of G is started at the ones from B . This process ends when there are no more vertices to be burnt, that is, the fire is contained. The goal when solving the FFP is to maximize the number of vertices that are not burned when the fire is contained, with the constraint that at most D vertices can be protected against the fire per iteration. Practical applications of the FFP, besides obtaining strategies to minimize the damage caused by fire, can be found in areas such as disease control and network security.

The FFP is NP-hard and heuristic methods to tackle the problem were proposed earlier in the literature. In this dissertation, firstly we present modifications made on the first ILP model proposed to the FFP through techniques of preprocessing and constraint aggregation. Moreover, we describe new greedy heuristics and also we introduce a novel matheuristic to the FFP, an approach based on the interoperation between metaheuristics and mathematical programming. A series of computational experiments were conducted on a public benchmark both for parameter tuning and to compare our results with those obtained previously. In respect to the modifications on the ILP model, a speedup of 2 in average was obtained. While constraint aggregation can lead to infeasible solutions, we prove that the latter can be converted to feasible ones in linear time. Regarding the heuristics, they were executed following a methodology to construct a solution in which greedy randomized choices are made to select which vertices should be defended, according to concepts introduced by the GRASP metaheuristic. Comparing these heuristics with the ones proposed by previous works, we observe that two of ours are between the five best ones in general. In relation to the matheuristic, through rigorous statistical analysis, we were able to verify that there is a statistically significant difference between our strategy and the remaining ones, while our matheuristic had better results on the majority of the instances.

Lista de Figuras

1.1	Exemplo de resolução de uma instância do FFP.	14
4.1	Exemplo de solução infactível gerada pelo M-FFMO.	32
4.2	Vizinhança de uma solução s^0	39
4.3	Exemplo de representação gráfica do teste de Nemenyi.	50
5.1	<i>Ranks</i> cumulativos das heurísticas, considerando todos os valores de α . . .	59
5.2	Testes de Iman-Davenport e Nemenyi entre as heurísticas para cada valor de α	60
5.3	<i>Ranks</i> cumulativos das heurísticas, para cada valor de α	61
5.4	Distância entre soluções.	64
5.5	Número de instâncias nas quais o respectivo índice do <i>pool</i> levou à melhor solução final.	65
5.6	Resultados do teste de Nemenyi entre as funções utilizadas para construção da vizinhança.	66
5.7	Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de Blum et al. [7] e Hu, Windbichler e Raidl [39] usando o conjunto BBGRL de instâncias.	67
5.8	Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o conjunto GBRL de instâncias.	68
5.9	Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o subconjunto A-GBRL de instâncias.	69
5.10	Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o subconjunto G-GBRL de instâncias.	70
5.11	Resultados do teste de robustez.	73

Lista de Tabelas

3.1	Heurísticas de García-Martínez et al. [27]	27
3.2	Parâmetros usados na geração das instâncias de García-Martínez et al. [27].	27
4.1	<i>Ranks</i> médios das heurísticas em 1200 instâncias.	48
4.2	Contagem de vitórias entre as três heurísticas de menor <i>rank</i> médio.	50
5.1	Estatísticas das 1306 instâncias com <i>speedup</i> de pelo menos 1.1.	56
5.2	Estatísticas de <i>speedup</i> das instâncias nas quais o FFMO consumiu pelo menos 5 minutos de computação.	56
5.3	Resultados dos testes estatísticos por valor de α .	58
5.4	Resultados dos testes estatísticos das heurísticas para diferentes valores de α .	58
5.5	<i>Gap</i> (%) entre a solução encontrada pela heurística Th com $\alpha = 0.3$ e o valor ótimo, para as instâncias com $n \in \{50, 100\}$.	61
5.6	Parâmetros da matheurística.	62
5.7	Estatísticas de <i>speedup</i> do T1.5-M-FFMO em relação ao OPT-M-FFMO.	63
5.8	<i>Gap</i> (%) entre a solução obtida pela busca local e a solução inicial da heurística Th para as instâncias do conjunto BBRL.	67
5.9	Estatísticas das diferenças entre os resultados de nossa matheurística e ILP(L,T,+1), quando os da primeira foram piores.	69
5.10	Estatísticas das diferenças entre os resultados de nossa matheurística e ILP(L,T,+1), quando os da primeira foram melhores.	69
5.11	<i>Gap</i> (%) entre a solução obtida pela busca local e a solução inicial da heurística Th para as instâncias do conjunto GBRL.	70
5.12	Instâncias selecionadas para o teste de robustez.	72
5.13	Tamanhos e parâmetros das instâncias do conjunto GER-ER.	74
5.14	Tamanhos e parâmetros das instâncias do conjunto GER-BA.	74
5.15	Comparação dos resultados das estratégias MATHEU e MATHEU-DI.	75

Sumário

1	Introdução	13
2	Preliminares	17
2.1	Notações e Definições	17
2.2	Modelo PLI para o FFP	18
2.3	Matheurísticas	19
2.4	GRASP	20
3	Revisão Bibliográfica	23
4	Metodologia	29
4.1	Melhorias no Modelo FFMO	29
4.2	Matheurística	32
4.2.1	Parâmetro T	33
4.2.2	Heurísticas Gulosas	34
4.2.3	Busca Local para uma Solução	37
4.2.4	<i>Pool</i> de soluções	41
4.2.5	Busca Local Adaptativa no <i>Pool</i> de Soluções	42
4.2.6	Busca Local com Diversificação e Intensificação	45
4.3	Análise Estatística de Resultados de Múltiplas Estratégias em Múltiplas Instâncias	46
4.4	Geração de Instâncias	52
5	Resultados Computacionais	54
5.1	Ambiente Computacional e Instâncias	54
5.2	Melhorias no Modelo FFMO	55
5.3	Heurísticas Gulosas	57
5.4	Matheurística	62
5.5	Teste de Robustez	71
5.6	Busca Local com Diversificação e Intensificação	72
6	Considerações Finais	76

Capítulo 1

Introdução

Incêndios são caracterizados por grandes ocorrências de fogo não controladas, os quais causam danos aos locais por onde são propagados. Os mesmos podem ser provocados acidentalmente, naturalmente (incêndio florestal) ou de forma intencional. Mais especificamente, causas de incêndios incluem mau funcionamento elétrico, práticas agrícolas, fogueiras e também fatores naturais, como certos tipos de vegetações que favorecem a combustão em regiões áridas [1, 6]. A propagação de um incêndio é geralmente facilitada por produtos inflamáveis, por vegetação seca e pelo clima.

Dentre os principais danos causados por incêndios está a degradação da fauna e flora. No Brasil, por exemplo, somente em 2017 foram identificados 275 120 focos de incêndio pelo Instituto Nacional de Pesquisas Espaciais (INPE), sendo que 132 296 desses (aproximadamente 48%) ocorreram no bioma da Amazônia¹, formado por um conjunto de ecossistemas interligados pela Floresta Amazônica e pela Bacia Hidrográfica do Rio Amazonas, o qual ocupa quase metade do território brasileiro.

Vistos tais danos que podem ser originados por incêndios, existem profissionais voluntários ou membros do corpo de bombeiros dedicados à sua prevenção e contenção, os brigadistas. Normalmente, tais profissionais combatem incêndios utilizando métodos diretos através do uso de água, terra ou abafadores quando é possível se aproximar do fogo; métodos intermediários através da limpeza de faixas próximas ao fogo com o objetivo de facilitar a aproximação ao mesmo; ou métodos indiretos, com a construção de aceiros, quando não é possível se posicionar em curta distância do fogo [41].

É interessante para os brigadistas utilizar uma estratégia de contenção de incêndios na qual o dano causado seja minimizado, especialmente dado que o número de brigadistas disponíveis é limitado. Uma maneira de construir tal estratégia é através da obtenção de uma formulação matemática do problema a qual, mediante o desenvolvimento de algoritmos, pode ser resolvida computacionalmente.

Uma das formulações matemáticas do problema de se obter uma estratégia para contenção de incêndios que minimize o dano causado pelo mesmo, considerando uma quantidade limitada de brigadistas, constitui o Problema do Brigadista em Grafos. O problema, aqui denotado por FFP, do inglês *The Firefighter Problem*, foi proposto em 1995 por Hartnell [36] como um modelo determinístico e em tempo discreto para simular a propagação

¹<http://www.inpe.br/queimadas>

e contenção de incêndios em grafos. Ele pode ser descrito da seguinte maneira. Na entrada, é dado um inteiro D representando a quantidade de brigadistas disponíveis, um grafo não direcionado e não ponderado $G = (V, E)$ e um subconjunto de vértices $B \subseteq V$. Então, inicia-se nos elementos de B um processo iterativo de propagação e contenção de fogo através dos vértices de G e termina quando não existem mais vértices que possam ser queimados, ou seja, quando o fogo está contido. O objetivo ao resolver o FFP é maximizar o número de vértices não queimados quando o fogo é contido, com a restrição de que no máximo D vértices podem ser protegidos contra o fogo por rodada.

Durante tal processo, cada um dos vértices de G pode estar em um de três estados: *intocado*, *queimado* ou *defendido*. Inicialmente, todos estão intocados. Então, na primeira rodada, os vértices de B , denominados focos de incêndio, passam para o estado de queimados. Em cada uma das rodadas subsequentes, cada um dos D brigadistas disponíveis pode escolher um vértice intocado para defender e, em seguida, o fogo se propaga de vértices queimados para seus vizinhos intocados.

Essa modelagem pode ser usada no contexto de espalhamento de vírus em computador, de informação em uma rede social e também de disseminação de doenças entre populações [2, 29, 56].

A Figura 1.1 apresenta um exemplo do processo iterativo que constitui a resolução de uma instância do FFP, considerando apenas um brigadista disponível. Vértices rotulados por **Q** ou **F**, no caso de focos de incêndio, estão queimados. Vértices rotulados com **D** estão defendidos e aqueles sem rótulos estão intocados. O número próximo de cada vértice é o seu identificador único. Na Figura 1.1(a) somente o vértice 5 está queimado, pois é um foco de incêndio. Na próxima rodada (Figura 1.1(b)), o vértice 6 é defendido pelo único brigadista disponível e os demais vértices rotulados por **Q** são queimados, uma vez que não foram defendidos e são vizinhos do vértice 5, queimado na rodada anterior. O processo termina na Figura 1.1(c) visto que o fogo foi contido, nesse caso, por uma estratégia em que foram defendidos o vértice 6 na rodada 1 e o 7 na rodada 2. Nesse exemplo, a solução ótima foi obtida, com quatro vértices salvos (i.e. não queimados): 6, 7, 8 e 9.

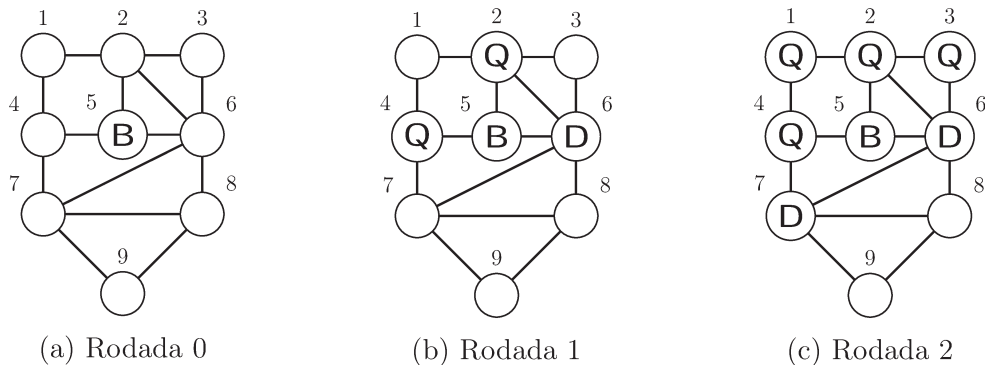


Figura 1.1: Exemplo de resolução de uma instância do FFP.

O FFP foi mostrado pertencer à classe de problemas NP-completo [26, 43, 46]. Portanto, é impossível que haja algoritmos determinísticos polinomiais para resolver tal problema de maneira exata, exceto se $P = NP$ [28]. Visto isso, alguns autores propuseram

algoritmos de aproximação [2, 9, 37, 42] para o FFP, enquanto outros realizaram análise de complexidade parametrizada [5, 11, 15]. Mais recentemente, alguns trabalhos trataram o FFP através de métodos heurísticos [7, 27, 39].

Neste trabalho, também investigamos o FFP do ponto de vista algorítmico, buscando encontrar formas mais eficientes e eficazes de resolvê-lo. De modo a alcançar tal objetivo, primeiramente resolvemos o FFP de maneira exata através do uso de Programação Linear Inteira (PLI) [52]. Também desenvolvemos métodos heurísticos visando obter soluções sem garantia de otimalidade; mas, com um baixo gasto de recursos computacionais. Nosso principal método consiste em uma *matheurística*, ou seja, uma abordagem que se baseia na interoperação de meta-heurísticas e técnicas de programação matemática [8]. Aplicamos conceitos da meta-heurística GRASP (do inglês *Greedy Randomized Search Procedures*) [23, 54] em conjunto com a resolução de modelos de Programação Linear Inteira (PLI) [52]. Utilizamos uma formulação PLI baseada na que foi provida por Develin e Hartke [17], a qual será denotada no restante deste documento por **FFM0**, um mnemônico do inglês *Firefighter Model*.

Comparamos resultados obtidos empiricamente com aqueles reportados em trabalhos previamente publicados. Conduzimos uma série de experimentos considerando diferentes quantidades de brigadistas disponíveis e também instâncias que se constituem de grafos gerados aleatoriamente com diferentes tamanhos (número de vértices e arestas), topologias e densidades.

Nossos principais resultados podem ser sumarizados como segue:

- Nós modificamos a formulação **FFM0**, usando técnicas de fixação de variáveis e agregação de restrições (mais detalhes na Seção 4.1). Considerando instâncias nas quais a solução ótima pode ser encontrada em no máximo uma hora (em específico, grafos com 50 e 100 vértices), obtivemos um *speedup* de aproximadamente 2 em média. Observamos que tais modificações podem levar à geração de soluções infactíveis, mas conseguimos demonstrar que é possível tornar tais soluções factíveis em tempo polinomial.
- Desenvolvemos novas heurísticas gulosas para resolver o FFP. Executamos cada uma das heurísticas seguindo uma metodologia para obter uma solução na qual escolhas gulosas aleatorizadas são realizadas para selecionar quais vértices serão defendidos, seguindo conceitos utilizados na meta-heurística GRASP (detalhes na Seção 4.2.2). De maneira a avaliar seu desempenho computacional, comparamos essas heurísticas com as que são propostas por García-Martínez et al. [27] em um mesmo conjunto de instâncias. Observamos que duas de nossas heurísticas estão entre as cinco melhores na maioria dos casos.
- García-Martínez et al. [27] obteve resultados empíricos que evidenciam o impacto do parâmetro T , um valor estimado para o número de rodadas suficientes para obter uma solução ótima, usado no modelo PLI. Os autores propuseram uma estratégia iterativa para ajustar tal valor. Nós desenvolvemos uma nova metodologia para obter tal estimativa, na qual utilizamos o número de rodadas de uma dada solução factível inicial, utilizada como ponto de partida para o resolvedor PLI (detalhes

na Seção 4.2.1). Considerando as instâncias nas quais o valor ótimo é conhecido, fomos capazes de obter bons resultados, tanto em termos de qualidade de solução (o valor ótimo foi alcançado em 99.95% das instâncias) quanto em tempo de computação (um *speedup* de aproximadamente 31 foi obtido, em média).

- Combinando os resultados mencionados, desenvolvemos uma nova matheurística para o FFP (detalhes na Seção 4.2). Comparamos nossos resultados com aqueles produzidos pelas estratégias descritas nos trabalhos de Blum et al. [7], García-Martínez et al. [27] e Hu, Windbichler e Raidl [39], nos mesmos conjuntos de instâncias. A análise dos resultados foi feita utilizando testes estatísticos não paramétricos de acordo com a metodologia sugerida por Demšar [16]. Mostramos que existe diferença estatisticamente significativa entre nossa estratégia e as demais, ao mesmo tempo que nossa matheurística obteve resultados melhores na maioria das instâncias. Uma variação dessa matheurística também foi proposta, como é discutido na Seção 4.2.6.
- Uma vez que nossa matheurística contém uma fase aleatorizada, conduzimos um teste de robustez de maneira a determinar se os resultados obtidos através de múltiplas repetições possuem pouca variância. Experimentos foram realizados em um conjunto seletivo de instâncias e observamos que nossa estratégia é robusta na maioria dos casos (detalhes na Seção 5.5). De nosso conhecimento, essa é a primeira vez que tal teste é feito dentre os trabalhos de estudo computacional relacionados ao FFP.
- Blum et al. [7] e García-Martínez et al. [27] deixaram em domínio público instâncias do FFP que foram utilizadas em seus respectivos trabalhos, com o intuito de que outros trabalhos que façam algum estudo computacional do FFP utilizem o mesmo *benchmark*. Neste trabalho, também geramos um conjunto de instâncias aleatoriamente utilizando os modelos de Barabási e Albert [4] e Erdős e Rényi [22] (detalhes na Seção 4.4). Conduzimos experimentos nessas instâncias com nossa matheurística (e uma variação proposta), mostrando os resultados separadamente por instâncias geradas por diferentes metodologias. Tais instâncias e os resultados associados foram deixados também em domínio público, no endereço: www.ic.unicamp.br/~cid/Problem-instances/Firefighter-in-Graphs.

Esta dissertação é estruturada da seguinte forma. O Capítulo 2 introduz notações e definições necessárias para entendimento dos demais capítulos, junto à descrição do modelo FFM0, e as principais técnicas que foram utilizadas neste trabalho são discutidas em mais detalhes. Em seguida, no Capítulo 3, as principais publicações presentes na literatura do FFP são apresentadas, com foco em seus resultados mais fundamentais. No Capítulo 4, a metodologia que foi utilizada para desenvolvimento deste trabalho é discutida. A descrição dos experimentos e dos resultados computacionais é feita no Capítulo 5. Por fim, o Capítulo 6 traz as conclusões.

Capítulo 2

Preliminares

Neste capítulo introduzimos notações e conceitos fundamentais para entendimento deste documento. Notações e definições básicas são expostas na Seção 2.1. O FFM0 é descrito na Seção 2.2. Neste trabalho, empregamos conceitos básicos de Otimização Combinatória, os quais assume-se que são conhecidos. Caso o leitor julgue necessário realizar uma revisão sobre os mesmos, recomendamos o livro de Nemhauser e Wolsey [52], o qual cobre tal tema com um enfoque em PLI, uma das principais ferramentas que utilizamos. Visto isso, nas Seções 2.3 e 2.4 discutimos, de forma geral, matheurísticas e a meta-heurística GRASP, técnicas nas quais nossa pesquisa se apoia.

2.1 Notações e Definições

Começamos com notações e definições necessárias ao bom entendimento do texto. Conceitos básicos relacionados à teoria dos grafos são considerados conhecidos e podem ser revistos em algum livro texto sobre o tema, por exemplo, em Diestel [19]. Seja um grafo não ponderado e não direcionado $G = (V, E)$, $V = \{1, \dots, n\}$ seu conjunto de vértices e $E = \{(u, v) : u \text{ e } v \in V, u \neq v\}$ seu conjunto de m arestas. Seja $d_G(u, v)$ o comprimento de um caminho mínimo de um vértice u para um vértice v em G e seja $d_G(v, S)$, $S \subseteq V$ o comprimento do menor entre os caminhos mínimos de v para todos os vértices em S , ou seja, $d_G(v, S) = \min\{d_G(u, v) : \forall u \in S\}$. Definimos como a k -vizinhança de um vértice v em G , denotada por $N_G(v, k)$, como sendo o conjunto $\{u : d_G(u, v) \leq k \text{ e } u \neq v\}$ e a k -vizinhança *estrita*, denotada como $N_G^*(v, k)$, o conjunto $\{u : d_G(u, v) = k \text{ e } u \neq v\}$. Um vértice u é dito um k -vizinho de v se $u \in N_G(v, k)$. Por conveniência, quando $k = 1$ denotamos a vizinhança simplesmente como $N_G(v)$ (similarmente para o caso estrito). A k -vizinhança de um conjunto $S \subseteq V$ em G , denotada por $N_G(S, k)$, é o conjunto $\bigcup_{v \in S} N_G(v, k)$. O conceito de vizinhança estrita é análogo para o conjunto S , incluindo a notação utilizada. Ademais, definimos o grau de um vértice v em G , denotado por $g_G(v)$, como sendo $g_G(v) = |N_G(v)|$. Quando está evidente sobre qual grafo estamos nos referindo, omitimos o subscrito G nas notações.

Nós agora apresentamos uma definição formal do FFP. Sejam parte da entrada do problema um grafo $G = (V, E)$ não ponderado e não direcionado, D um inteiro positivo representando o número de brigadistas disponíveis e $B \subseteq V$ o conjunto não vazio de

vértices que são focos de incêndio. Um processo de contenção e espalhamento de fogo em G se inicia em rodadas discretas, tal que cada um dos vértices de G pode estar em um de três diferentes estados a cada rodada t : *intocado*, *queimado* ou *defendido*. Uma vez que um vértice está queimado ou defendido, ele se mantém nesse estado nas rodadas posteriores. Na primeira rodada $t = 0$, cada vértice em B é queimado e os demais vértices no conjunto $V \setminus B$ permanecem intocados. Em seguida, a cada rodada subsequente $t \in \{1, 2, 3, \dots\}$, no máximo D vértices intocados são selecionados para serem defendidos e o fogo se espalha naqueles que permaneceram intocados e que têm pelo menos um vértice da sua vizinhança queimado. Esse processo termina quando mais nenhum vértice pode ser queimado, i.e., o fogo está contido. Um vértice é dito *salvo* se, ao final do processo, está defendido ou intocado.

Quando o processo termina, uma solução s é obtida, sendo *completa* se o fogo foi contido e *minimal* se todos os vértices defendidos são vizinhos de queimados. Nós representamos tal solução usando o conjunto $\{(v, t) : v \in V \text{ e } t \in \mathbb{Z}\}$, no qual cada par ordenado representa a rodada t na qual o vértice v teve seu estado alterado (se uma alteração ocorreu). O estado de v pode ser inferido a partir de t , tal que:

$$t \begin{cases} = 0, & \text{se } v \in B \text{ e foi queimado,} \\ \geq 1, & \text{se } v \text{ foi defendido na rodada } t, \\ \leq -1, & \text{se } v \text{ foi queimado na rodada } |t|. \end{cases}$$

Denotamos o conjunto de vértices defendidos em s como \mathcal{D}_s , e \mathcal{B}_s o de queimados. O custo de uma solução s é dado por $z(s)$ que denota o número de vértices salvos, i.e., não queimados. O número de rodadas necessárias para conter o fogo em s é denotado por T_s .

Outra importante definição para entendimento do procedimento de busca local que desenvolvemos é a distância entre duas soluções s' e s'' , denotada como $\phi(s', s'')$. Define-se tal distância como sendo o conjunto de comprimentos dos caminhos mínimos de todos os vértices em $\mathcal{D}_{s'}$ até o conjunto $\mathcal{D}_{s''}$ ou, mais formalmente: $\phi(s', s'') = \{d(v, \mathcal{D}_{s''}) : \forall v \in \mathcal{D}_{s'}\}$.

2.2 Modelo PLI para o FFP

O objetivo do FFP é maximizar o número de vértices salvos, o que é equivalente a minimizar a quantidade de vértices queimados no final do processo. Uma maneira de resolvê-lo à otimalidade é aplicar um resolvidor de Programação Linear Inteira (PLI) a partir de uma modelagem matemática PLI do problema. O primeiro modelo PLI para o FFP foi introduzido por Develin e Hartke [17] (como mencionado antes, denotado por **FFMO**) e é descrito nas Equações (2.1) a (2.10). O modelo contém dois conjuntos de variáveis binárias. O primeiro possui uma variável $b_{v,t}, \forall v \in V$ e $0 \leq t \leq T$, que indica se um vértice v está ou não queimado na rodada t e o segundo conjunto possui uma variável $d_{v,t}, \forall v \in V$ e $0 \leq t \leq T$ que indica se um vértice v está ou não defendido na rodada t . O valor inteiro positivo de T é um limitante superior no número de rodadas suficiente para obter uma solução ótima. Os valores de D, T e os vértices no conjunto B são parâmetros de entrada do modelo.

$$\max \quad |V| - \sum_{v \in V} b_{v,T} \quad (2.1)$$

s.a.

$$b_{v,t} + d_{v,t} - b_{v',t-1} \geq 0 \quad \forall v \in V, v' \in N(v) \text{ e } 1 \leq t \leq T \quad (2.2)$$

$$b_{v,t} + d_{v,t} \leq 1 \quad \forall v \in V \text{ e } 1 \leq t \leq T \quad (2.3)$$

$$b_{v,t} - b_{v,t-1} \geq 0 \quad \forall v \in V \text{ e } 1 \leq t \leq T \quad (2.4)$$

$$d_{v,t} - d_{v,t-1} \geq 0 \quad \forall v \in V \text{ e } 1 \leq t \leq T \quad (2.5)$$

$$\sum_{v \in V} (d_{v,t} - d_{v,t-1}) \leq D \quad \text{para } 1 \leq t \leq T \quad (2.6)$$

$$b_{v,0} = 1 \quad \forall v \in B \quad (2.7)$$

$$b_{v,0} = 0 \quad \forall v \in V \setminus B \quad (2.8)$$

$$d_{v,0} = 0 \quad \forall v \in V \quad (2.9)$$

$$b_{v,t}, d_{v,t} \in \{0, 1\} \quad \forall v \in V \text{ e } 1 \leq t \leq T \quad (2.10)$$

A função objetivo (2.1) maximiza o número de vértices salvos. A restrição (2.2) impõe que um vértice v tem que estar queimado ou defendido na rodada t se um de seus vértices vizinhos foi queimado na rodada anterior $t - 1$. A restrição (2.3) garante que um vértice não está queimado e defendido na mesma rodada. Para assegurar que um vértice uma vez queimado se mantenha assim nas rodadas posteriores, têm-se a restrição (2.4). A restrição (2.5) faz o mesmo para um vértice defendido. A restrição (2.6) limita o número de vértices defendidos a cada rodada para ser no máximo D . As restrições (2.7)-(2.9) forçam as condições de inicialização, tal que em $t = 0$ todos os vértices pertencentes ao conjunto B estão queimados, enquanto todos os demais se mantêm intocados. Esse modelo tem uma quantidade de variáveis na ordem de $O(nT)$ e restrições na ordem de $O(mT)$.

2.3 Matheurísticas

Nossa principal estratégia desenvolvida para resolver o FFP consiste em uma *matheurística*, um algoritmo heurístico que se baseia na interoperação de meta-heurísticas e programação matemática. Uma das características fundamentais de tal técnica é a exploração de aspectos derivados do modelo matemático do respectivo problema em alguma parte do algoritmo [8]. Matheurísticas geralmente operam em um modo “mestre-escravo”, o qual segue duas perspectivas diferentes: (i) a meta-heurística atua em um nível mais alto e controla as chamadas para as abordagens exatas; (ii) um algoritmo exato atua como mestre e invoca e controla o uso da meta-heurística.

A abordagem deste trabalho segue a perspectiva (i) na qual, geralmente, a meta-heurística define o espaço de soluções a ser considerado, enquanto o algoritmo exato é invocado para explorar tal espaço [10]. Esse tópico tem atraído considerável atenção da comunidade de pesquisa, visto que já existe inclusive uma conferência voltada exclusi-

vamente ao mesmo [48], além de volumes e edições dedicados ao assunto em revistas científicas [35, 47].

Neste trabalho, usamos conceitos da meta-heurística GRASP (acrônimo do inglês *Greedy Randomized Search Procedures*) [23], a qual será exposta em mais detalhes na Seção 2.4. A parte de nossa estratégia que envolve programação matemática consiste na resolução de uma versão mais restrita do modelo PLI discutido na Seção 2.2.

2.4 GRASP

Nessa seção, os principais componentes da meta-heurística GRASP são detalhados segundo o que é mostrado no artigo seminal [23]. De maneira geral, o GRASP consiste em um processo iterativo, no qual cada iteração possui duas fases: uma fase construtiva gulosa, aleatorizada e adaptativa, seguida de uma fase de busca local. No final do processo, a melhor solução obtida é retornada. O Algoritmo 1 provê uma estrutura genérica do GRASP. A fase construtiva (Linha 3), busca local (Linha 4) e atualização da solução incumbente (Linha 5) são executadas a cada iteração do laço das Linhas 2 a 5, até que algum critério de parada seja satisfeito, como um número máximo de iterações, limitante na qualidade da solução encontrada ou tempo máximo de computação.

Algoritmo 1: Estrutura genérica do GRASP

Entrada: N

Saída: Solução s^*

```

1  $s^* \leftarrow \emptyset$ 
2 enquanto Critério de parada não satisfeito faça
3    $s \leftarrow \text{Construção-Gulosa-Aleatorizada}()$ 
4    $s' \leftarrow \text{Busca-Local}(N, s)$ 
5    $\text{Atualiza-Solução}(s', s^*)$ 
6 retorna  $s^*$ 
```

Na fase construtiva, uma solução factível é construída iterativamente, de forma que um elemento é incluído por vez. A escolha do elemento a ser adicionado à solução é feita baseando-se em uma função gulosa [13] que é utilizada para ordenar uma lista de possíveis candidatos. Considerando um problema de minimização, tal função mensura qual o custo de inserir o respectivo elemento na solução parcialmente construída atual, sem considerar explicitamente qual o custo da solução final, ou seja, faz uma escolha que é localmente ótima, o que provê a característica gulosa do GRASP. O componente probabilístico é caracterizado pela escolha aleatória de um dentre os melhores candidatos armazenados em uma *Lista Restrita de Candidatos* (LRC). Tal lista é construída selecionando uma quantidade restrita de elementos dentre aqueles que podem ser adicionados à solução. O tamanho da LRC é determinado por um parâmetro $\alpha \in [0, 1]$, que indica quão gulosa deve ser a escolha do elemento a entrar na LRC. Geralmente, os elementos da LRC são aqueles que têm custo menor ou igual a $c_{\min} + \alpha \cdot (c_{\max} - c_{\min})$, onde c_{\min} é o menor e c_{\max} o maior custo entre todos os elementos considerados. Quando $\alpha = 0$ a construção é totalmente gulosa (determinística) e quando $\alpha = 1.0$ a escolha é completamente aleatória.

Perceba que a introdução de aleatoriedade na escolha permite que diferentes soluções sejam obtidas na fase construtiva de cada uma das iterações do GRASP.

Uma vez que um elemento é selecionado para ser incluído na solução, os custos dos elementos restantes, segundo a função gulosa, podem sofrer alterações, devendo, portanto, ser atualizados. Isso fornece a característica adaptativa do GRASP. O Algoritmo 2 corresponde a um esquema geral da fase construtiva.

Algoritmo 2: Fase de construção do GRASP

Saída: Solução s

```

1  $s \leftarrow \emptyset$ 
2 enquanto Solução não construída faça
3    $lrc \leftarrow \text{Constrói-LRC}()$ 
4    $e \leftarrow \text{Seleciona-Elemento-Aleatório}(lrc)$ 
5    $s \leftarrow s \cup \{e\}$  ▷ Adiciona  $e$  à solução  $s$ .
6    $\text{Adapta-Função-Gulosa}(e)$  ▷ Atualiza os custos segundo a função gulosa.
7 retorna  $s$ 
```

A solução obtida na fase construtiva pode não ser ótima localmente. Em vista disso, um procedimento de busca local pode melhorar a solução construída. Assim, dada uma solução s e uma estrutura de vizinhança $\text{Viz}(s)$, um algoritmo de busca local funciona de maneira iterativa, sucessivamente substituindo a solução atual por uma solução melhor na sua vizinhança. A vizinhança de uma solução s é formada por soluções que mantêm alguma relação com s considerando o problema. Geralmente, tal procedimento termina quando não existe uma solução melhor na vizinhança e quando isso acontece, a solução é dita localmente ótima. Um procedimento geral de busca local é descrito no Algoritmo 3.

Algoritmo 3: Fase de busca local do GRASP

Entrada: N, s

Saída: Solução s'

```

1  $s' \leftarrow s$ 
2 enquanto  $s'$  não é localmente ótima faça
3   se Existe solução  $s'' \in \text{Viz}(s')$  melhor do que  $s'$  então
4      $s' \leftarrow s''$ 
5 retorna  $s'$ 
```

Como observado por Feo e Resende [23], o GRASP pode ser visto como uma técnica de amostragem repetitiva seguindo uma distribuição de probabilidade desconhecida, na qual média e variância de tal restrição são funções da natureza restritiva da LRC. O GRASP tem sido usado extensivamente em diversos problemas de otimização. Um compilado de aplicações de tal meta-heurística pode ser visto no trabalho de Festa e Resende [24].

Uma estratégia que já foi utilizada junto do GRASP e com certo sucesso [53], é o *Path Relinking*, proposto originalmente por Glover, Laguna e Marti [31] como uma técnica de intensificação para exploração de trajetórias conectando soluções de boa qualidade (elites) obtidas previamente por outros métodos, como por exemplo a Busca Tabu [33]. Começando por uma ou mais soluções elite, chamadas de *iniciais*, caminhos no espaço de

busca são gerados em direção a outras soluções elite, denominadas *guias*, com o intuito de encontrar soluções melhores. De forma a gerar esses caminhos, *movimentos* são selecionados para introduzir atributos na solução corrente, os quais estão presentes na solução elite guia. Esse mecanismo tem o intuito de incorporar atributos presentes em soluções de boa qualidade, ao favorecer sua inclusão durante a seleção de movimentos.

Como discutido em Resende e Ribeiro [53], o *Path Relinking* pode ser utilizado em conjunto com o GRASP, por exemplo, sendo aplicado entre uma solução localmente ótima obtida em uma iteração e outra escolhida de um dado *pool* de soluções de boa qualidade, obtidas durante as iterações anteriores.

Capítulo 3

Revisão Bibliográfica

Neste capítulo, descrevemos os resultados obtidos por trabalhos presentes na literatura. Nosso foco é naqueles que realizaram estudos computacionais para o FFP e serviram como base para nossas comparações, mas também discutimos brevemente trabalhos de natureza teórica.

O FFP foi proposto inicialmente por Hartnell [36] em 1995. Segundo o *survey* de Finbow e MacGillivray [25], a versão de decisão do FFP foi provada pertencer à classe NP-completo [28] pela primeira vez no trabalho de MacGillivray e Wang [46], onde tal prova se restringiu a grafos bipartidos, porém, não tivemos acesso a tal artigo. Mais tarde, provou-se o mesmo para árvores de grau máximo três [26] e grafos cúbicos [43]. Em ambos casos, a prova de complexidade consiste em uma redução do *not-all-equal 3-SAT* [28] para o FFP. Como tais resultados relativos à complexidade do FFP são válidos para grafos com uma topologia específica, os mesmo podem ser naturalmente estendidos ao caso geral.

Chlebíková e Chopin [11] fazem uma análise sobre quais são as propriedades de um grafo que governam a complexidade do FFP, sendo essas o *pathwidth* [55] e o grau máximo. Os autores mostram que o FFP é NP-completo para árvores com *pathwidth* de valor no máximo 3 e $D \geq 1$ e analisam a complexidade parametrizada do problema usando como parâmetros os valores para as respectivas propriedades. Mais recentemente, os autores estenderam seus resultados com provas de NP-completude para grafos muito densos [12].

Leizhen e Weifan [45] definem o conceito de *taxa de sobrevivência* (do inglês *surviving rate*) de um grafo G , denotada por $\rho(G)$, como sendo a porcentagem média de vértices que podem ser salvos quando o fogo inicia em um vértice aleatório. No mesmo trabalho, os autores encontram limitantes para o valor de $\rho(G)$ para diferentes classes de grafos. Posteriormente, Wang, Finbow e Wang [58] estendem os resultados para outras classes.

Os seguintes trabalhos propuseram algoritmos de aproximação para o FFP. Anshelevich et al. [2] considera em seu trabalho o modelo do FFP para o espalhamento de doenças e contenção através de vacinas, no qual são introduzidas diferentes variações do FFP e diferentes funções objetivo. Os autores obtiveram resultados relacionados a aproximações para tais variações e demonstram através de uma redução de uma delas, denominada MAXSAVE, para o FFP que o mesmo não é aproximável por um fator $n^{1-\epsilon}$, $\epsilon > 0$ a não ser que $P = NP$. Hartnell e Li [37] propõem um algoritmo guloso 1/2-aproximado para o FFP em árvores. Os autores Cai, Verbin e Yang [9] trazem em seu trabalho uma abordagem usando Programação Linear (PL) e arredondamento aleatório também para árvores,

com fator de aproximação de $1 - (1/e)$, mais tarde melhorado por Iwaikawa, Kamiyama e Matsui [42], para um fator de 0.7144 usando técnicas de enumeração explícita e indução regressiva.

O trabalho de Cai, Verbin e Yang [9] também lida com a complexidade parametrizada do FFP. Os autores mostram que o FFP em árvores é *Fixed Parameter Tractable* (FPT) e admite *kernels* de tamanho polinomial [21] para várias escolhas de um parâmetro k , incluindo o número de vértices salvos, número de folhas salvas e o número de vértices defendidos. Também são deixadas várias questões em aberto, mais tarde exploradas nos artigos de Bazgan, Chopin e Fellows [5] e Cygan, Fomin e Leeuwen [15], os quais também estenderam os resultados de Cai, Verbin e Yang [9], considerando outros parâmetros, como o número de cobertura de vértices e o *treewidth* [21].

O artigo de Develin e Hartke [17] expõe uma série de resultados teóricos considerando o FFP especificamente em grafos *grid*. Os autores também propuseram a primeira formulação PLI para o FFP, a qual foi utilizada para provar um dos resultados. Tal prova foi realizada para mostrar que em um grafo *square grid* bidimensional infinito, se o fogo inicia em um único vértice arbitrário e $D = 2$, então pelo menos 18 vértices são queimados em qualquer estratégia e o menor número de rodadas suficientes para conter o fogo é 8.

Blum et al. [7] propõem a primeira adaptação de uma meta-heurística para o FFP, sendo essa a *Ant Colony Optimization* (ACO) [20]. Os dois componentes fundamentais da ACO são a representação de uma solução e modelo de feromônio. Como representação de uma solução uma permutação π dos n vértices é utilizada, onde a ordem em que os vértices aparecem em π (da esquerda para a direita) indica quais vértices devem ser defendidos a cada rodada. Já o modelo de feromônio define um valor $\tau_{v,j}$ para cada vértice $v \in V$ e para cada posição $j, 1 \leq j \leq |N|$ de π , indicando o ganho ao inserir o vértice v na posição j , considerando a qualidade da solução obtida após aplicar tal operação. Ademais, os autores criaram uma versão híbrida da estratégia ACO, denominada HyACO (*Hybrid ACO*). Nessa versão, a melhor solução encontrada pela ACO, em um dado limite de tempo, é passada como entrada para a técnica denominada *solution polishing* do CPLEX, usando o FFM0. Tal técnica consiste em uma busca local, sem garantia de otimalidade, que utiliza *branch and cut* [14]. De maneira a avaliar o desempenho dos métodos propostos, os autores conduziram experimentos com $D \in \{1, 2, \dots, 10\}$ em um conjunto de instâncias formado por grafos gerados de maneira aleatória seguindo o modelo de Gilbert [30] (também conhecido como o modelo $G(n, p)$ de Erdős e Rényi [22]). Nesse modelo, dado um valor n para a quantidade desejada de vértices, adiciona-se ou não uma aresta entre dois vértices u e v com dada probabilidade p , para todos os pares possíveis de vértices distintos. Foram gerados grafos com $n \in \{50, 100, 500, 1000\}$, de forma que, para cada valor de n , as seguintes probabilidades foram consideradas, respectivamente: $\{0.1, 0.15, 0.2\}$, $\{0.05, 0.075, 0.1\}$, $\{0.015, 0.02, 0.025\}$, $\{0.0075, 0.01, 0.0125\}$. Para cada combinação de n e p foram gerados 10 grafos, totalizando 120 instâncias. Em todos os grafos, o vértice de índice 1 foi considerado como foco de incêndio. Durante os experimentos, a execução do CPLEX como resolvido PLI, utilizando o modelo de Develin e Hartke [17], também foi utilizada como base de comparação. Um tempo limite de $n/2$ segundos foi dado a cada estratégia, sendo que na HyACO $n/4$ são dedicados à primeira fase (ACO) e o restante para a segunda fase (*solution polishing*). Os testes mostraram

que, para grafos com $n \in \{50, 100\}$, o CPLEX alcança os melhores resultados, enquanto que para grafos com quantidade maior de vértices a estratégia HyACO supera as demais, na maioria dos casos.

Posteriormente, Hu, Windbichler e Raidl [39] sugeriram uma nova representação de solução para o FFP, de maneira a melhorar a complexidade de espaço sem, contudo, afetar a complexidade do algoritmo de avaliação em relação à representação utilizada por Blum et al. [7]. Segundo os autores, uma solução é representada por um vetor de *bits* $P = \langle p_1, \dots, p_n \rangle$ no qual p_v indica se o vértice v deve ou não ser defendido. A rodada na qual o vértice deve ser defendido é inferida durante o processo de avaliação da solução.

Em conjunto com a contribuição de uma nova representação de solução para o FFP, também é apresentada uma adaptação da meta-heurística *Variable Neighborhood Search* (VNS) para o respectivo problema. Os experimentos com tal técnica são conduzidos nas instâncias de Blum et al. [7], seguindo a mesma metodologia empregada por aqueles autores para execução dos experimentos, com exceção do tempo limite, o qual foi reduzido para $n/4$ devido a diferença no ambiente computacional utilizado. Essa diferença corresponde a um fator de 2, medida utilizando-se o *Standard Performance Evaluation Corporation (SPEC) benchmark*¹.

Os resultados obtidos com a estratégia VNS foram comparados com os de Blum et al. [7]. A análise feita pelos autores mostrou que a VNS não foi capaz de superar a estratégia HyACO de maneira significativa, mas foi pelo menos tão boa quanto ela em termos de qualidade de solução em grande parte das instâncias.

García-Martínez et al. [27] conseguiram resultados para o FFP através de abordagens heurísticas. Primeiramente, os autores mostraram empiricamente o impacto de diferentes valores do parâmetro T no desempenho do modelo FFM0, sendo esses valores entre $\{2, \dots, 50\}$. Um total de 20 instâncias foram utilizadas: grafos com $n = 100$, valor de $D = 1$ e com um limite de tempo de computação de 50 segundos. Como observações de tal experimento, os autores verificaram que valores subestimados de T (próximos de 2) levam à obtenção de soluções incompletas com um tempo de computação pequeno. Já quando T é superestimado (próximo a 50), o número de variáveis e restrições do modelo PLI cresce substancialmente, dificultando uma busca eficiente no espaço de soluções pelo resolvidor dadas as restrições de recursos computacionais. Para aumentar a probabilidade de obter soluções ótimas, ou mesmo completas, valores de T entre 10 e 20 se mostraram eficazes. Vista tal dificuldade em encontrar valores adequados para o parâmetro T , García-Martínez et al. [27] propuseram uma estratégia adaptativa para resolver o modelo PLI, na qual o processo de busca é reiniciado quando um valor de T insuficiente é encontrado, uma solução ótima é obtida ou quando um dado tempo limite é atingido. Tal estratégia é denotada por $ILP(T_0, C_{\text{comp}}, \Delta_T)$, sendo cada parâmetro:

- T_0 : utilizado como valor inicial de T . Três diferentes valores são considerados: (1) $T_0 = 2$. (2) $T_0 = L$, o qual denota o maior dentre os menores caminhos do foco de incêndio até todos os demais vértices. (3) $T_0 = 2L$.
- C_{comp} : restrição de completude, a qual indica se o resolvidor PLI deve ou não retornar soluções completas caso elas existam para o valor corrente de T . (a) Caso

¹<https://www.spec.org/benchmarks.html>

seja requerida, uma restrição é adicionada ao modelo PLI descrito no Capítulo 2:

$$b_{v,T} = b_{v,T-1}, \forall v \in V \quad (3.1)$$

Tal restrição impõe que o conjunto de vértices queimados não seja alterado na última rodada. (b) Caso não, se uma solução incompleta for devolvida pelo resolvidor, um procedimento que defende vértices aleatoriamente até que o fogo seja contido é utilizado.

- Δ_T : quando é realizado um reinício no valor T , são considerados os valores: (a) incremento de uma unidade em T ($\Delta_T = +1$). (b) dobrar o valor de T ($\Delta_T = \times 2$)

Uma estratégia foi obtida para cada combinação dos três parâmetros, com exceção da que tem $T_0 = 2L$, visto que, segundo os autores, não precisa de reinícios, totalizando 9 diferentes estratégias.

Os autores também propuseram heurísticas gulosas para resolver o FFP, cada heurística utiliza um determinado critério para atribuir prioridades aos vértices, de forma que a cada rodada, o vértice de maior prioridade é selecionado para ser defendido. Os critérios utilizados pelas heurísticas são os seguintes, onde os nomes entre parênteses são as notações utilizadas pelos autores:

- **Grau (Degree)**: Vértices de maior grau recebem maior prioridade.
- **Descendentes (Desc)**: Vértice com maior número de descendentes recebem maior prioridade. Um vértice v é considerado descendente de um vértice u se existe um caminho entre os dois e se u está mais próximo de B do que v .
- **Ameaça (Th)**: Vértices que podem ser queimados na rodada subsequente (ameaçados) recebem maior prioridade.

Também foram propostas duas estratégias que determinam *quando* um vértice deve ser defendido, sendo elas:

1. **Agora (Now)**: Defende o vértice de maior prioridade o mais brevemente possível, i.e., na rodada atual enquanto houverem brigadistas disponíveis.
2. **Depois (Later)**: Seja $d(v, B)$ o comprimento do menor caminho do vértice v até os vértices que são focos de incêndio. Essa estratégia escalona a defesa de v para o mais tarde possível em uma rodada $l, 1 \leq l \leq d(v, B)$.

Os autores combinam esses critérios e estratégias de escalonamento, denotando cada combinação por $H(\langle \text{critério} \rangle, \langle \text{quando} \rangle)$, como mostrado na Tabela 3.1. Colunas são os nomes das heurísticas e linhas a denotação das estratégias de escalonamento.

Observa-se que a heurística **Th** é combinada com **Degree** e **Desc**. Sendo estas utilizadas como critério de desempate para decidir, entre os vértices ameaçados, qual deve ser priorizado para ser defendido. Outra observação é que existe um conflito entre a estratégia de escalonamento **Later** e a heurística **Th**, logo essas não foram combinadas.

Tabela 3.1: Heurísticas de García-Martínez et al. [27]

Quando/Critério	Degree	Desc	Desc + Th	Degree + Th
Now	H(Degree, Now)	H(Desc, Now)	H(ThDesc, Now)	H(ThDegree, Now)
Later	H(Degree, Later)	H(Desc, Later)	-	-

Ademais, no mesmo artigo, uma estratégia de *backtracking* foi proposta para o FFP, a qual explora a árvore de possibilidades de diferentes subconjuntos de vértices a serem defendidos.

O trabalho descreve experimentos em um conjunto de 240 instâncias (denotado por GBRL), sendo 120 grafos gerados segundo o modelo de Gilbert [30], nomeados *grafos aleatórios* (denotados por A-GBRL) e 120 segundo um modelo geométrico, chamados *grafos geométricos aleatórios* (denotados por G-GBRL). Nesse modelo geométrico, n vértices são amostrados em um espaço bidimensional $[0, 1] \times [0, 1]$ e vértices que estão a uma distância euclidiana inferior a um valor r são conectados por uma aresta. Grafos foram gerados com n vértices, $n \in \{50, 100, 500, 1000\}$ e os valores de p e r foram escolhidos de maneira a obter grau médio entre $\{7.5, 10, 12.5\}$, sendo tais valores apresentados na Tabela 3.2. Para cada combinação de n com p ou r , denominada de *classe de problema* pelos autores, foram gerados 10 grafos.

Tabela 3.2: Parâmetros usados na geração das instâncias de García-Martínez et al. [27].

Grafos Aleatórios		Grafos geométricos aleatórios	
n	p	n	r
50	$\{0.1500, 0.2000, 0.2500\}$	50	$\{0.2590, 0.2990, 0.3340\}$
100	$\{0.0750, 0.1000, 0.1250\}$	100	$\{0.1690, 0.1950, 0.2190\}$
500	$\{0.0150, 0.0200, 0.0250\}$	500	$\{0.0710, 0.0830, 0.0930\}$
1000	$\{0.0075, 0.0100, 0.0125\}$	1000	$\{0.0500, 0.0580, 0.0650\}$

Assim como no trabalho de Blum et al. [7] um tempo limite de computação de $n/2$ segundos foi atribuído a cada estratégia. A quantidade de brigadistas utilizada foi $D \in \{1, 2, \dots, 10\}$ e o foco de incêndio é sempre o vértice de índice 1. Para a análise de resultados, os autores utilizaram o teste de Iman-Davenport [40] para identificar se existe diferença estatisticamente significativa entre várias estratégias e, caso exista, o teste de Holm [38] é empregado de maneira a verificar se existe diferença entre a estratégia de melhor *rank* (algoritmo de controle) e as demais estratégias.

Sobre os resultados obtidos referentes às estratégias de ajuste de T , os autores verificaram que $\text{ILP}(L, T, +1)$ e $\text{ILP}(L, F, +1)$ estão sempre entre as três melhores, para todos os conjuntos de instâncias, A-GBRL, G-GBRL e GBRL. O teste de Holm não identificou diferença entre as duas abordagens. Para o caso particular do conjunto G-GBRL, os melhores resultados foram conseguidos por $\text{ILP}(2, F, +1)$.

Em seus experimentos, os autores fazem múltiplas execuções de cada heurística da seguinte maneira. Na primeira delas, a heurística roda no modo determinístico, sempre escolhendo os vértices de maior prioridade para serem defendidos. Antes de cada uma das próximas execuções, uma lista restrita de vértices é construída, selecionando vértices

aleatoriamente dentre aqueles que estão intocados, seguindo uma distribuição uniforme. Em seguida, as escolhas gulosas da respectiva heurística são feitas sobre essa lista. As heurísticas $H(\text{ThDesc}, \text{Now})$ e $H(\text{ThDegree}, \text{Now})$ se mantiveram entre as duas melhores para todos os conjuntos de instâncias A-GBRL, G-GBRL e GBRL. O teste de Holm não identificou diferença significativa entre as duas.

Os autores observaram que ao comparar as estratégias $ILP(L, T, +1)$, $ILP(2, F, +1)$, $H(\text{ThDesc}, \text{Now})$ e $H(\text{ThDegree}, \text{Now})$, a $ILP(L, T, +1)$ obteve os melhores resultados no caso geral, com diferença estatisticamente significativa identificada em relação às demais, segundo o teste de Holm.

Os três trabalhos supracitados são os únicos presentes na literatura que exibem estudos computacionais do FFP. Mais detalhes sobre análises teóricas do FFP podem ser encontradas no *survey* de Finbow e MacGillivray [25].

Dentre as variações já propostas do FFP, têm-se os seguintes trabalhos. Michalak [50] propõe um procedimento evolutivo de busca local para a versão multiobjetivo do FFP [49]. Seja π uma permutação dos n vértices, na qual a ordem em que os vértices aparecem implica em qual rodada eles devem ser defendidos durante o processo de avaliação da solução. A busca local baseia-se em movimentos de transposição em π de maneira a obter soluções vizinhas e diferentes variações são propostas para reduzir o tamanho da vizinhança considerada. Resultados obtidos são comparados para tais variações, tendo como instâncias grafos com 50 a 250 vértices gerados utilizando o modelo de Gilbert [30]. Michalak [51] propõem Algoritmos de Estimação de Distribuição (EDA - *Estimation of Distribution Algorithms*) para o FFP considerando uma variação na qual custos são atribuídos aos vértices e o objetivo é maximizar o somatório dos vértices salvos multiplicados por seus respectivos custos. Comparações são realizadas entre os seus resultados com a EDA e as estratégias de Blum et al. [7] e Hu, Windbichler e Raidl [39], de forma que para grafos até 1000 vértices a estratégia de Hu, Windbichler e Raidl [39] levou a melhores resultados, enquanto que para grafos com mais de 1000 vértices a estratégia do autor é a melhor. Uma variação do FFP denominada *Politician's Firefighting* é introduzida por Scott, Stege e Zeh [57], na qual não existe um número fixo de brigadistas disponíveis, mas, para cada vértice queimado, um novo brigadista pode ser utilizado na próxima rodada para defender um de seus vizinhos intocados. Um algoritmo polinomial é proposto para árvores, junto da análise de complexidade do problema, provando que o mesmo em sua versão geral pertence à classe NP-difícil [28]. Por fim, uma análise da complexidade parametrizada de tal variação também é feita.

Capítulo 4

Metodologia

Nesse capítulo, apresentamos metodologias empregadas para resolver o FFP, analisar resultados e gerar instâncias.

4.1 Melhorias no Modelo FFM0

Nós fizemos duas modificações no FFM0. A primeira foi um pré-processamento para fixação de variáveis. Isso foi feito ao notar-se que as variáveis $b_{v,t}$ podem ser fixadas em zero para todo $v \in V \setminus B$ e $1 \leq t < d(v, B)$, uma vez que o fogo não pode alcançar um vértice v em uma rodada $t < d(v, B)$. A segunda alteração consistiu na substituição das restrições (2.6) por suas versões agregadas. Para uma dada rodada t , com $1 \leq t \leq T$, se somarmos as restrições (2.6) para $1 \leq t' \leq t$, após cancelarmos os termos, uma vez que $d_{v,0} = 0, \forall v \in V$, temos:

$$\sum_{v \in V} d_{v,t} \leq t \cdot D \quad \text{para } 1 \leq t \leq T. \quad (2.6')$$

Visto isso, substituímos a restrição (2.6) por a sua versão agregada (2.6'). No restante do documento, denotamos esse modelo com ambas modificações por M-FFM0.

A restrição (2.6') pode levar a geração de soluções infactíveis, uma vez que permite que brigadistas sejam acumulados ao longo das rodadas e consequentemente, mais do que D vértices podem ser defendidos em uma rodada, violando uma das restrições do problema.

Apesar de tal violação, nós demonstramos que é possível transformar uma solução infactível gerada pelo M-FFM0 em uma solução factível, em tempo polinomial. Para provar isso, seja d_t o número de vértices defendidos na rodada t e $l_t = d_t - D$, tal que l_t é considerado um *excesso* se $l_t > 0$ e um *déficit* se $l_t < 0$. Primeiramente, provamos os seguintes lemas que são fundamentais para mostrar que tal transformação é possível.

Lema 1. *Seja s uma solução infactível para o FFM0 gerada pelo M-FFM0. Seja \bar{t} a primeira rodada de s na qual um excesso ocorre. Então, existe um conjunto não vazio de rodadas $L \subseteq \{1, \dots, \bar{t} - 1\}$ tal que $l_{t'} < 0, \forall t' \in L$.*

Demonstração. Suponha por contradição que tal conjunto L não existe. Então, temos $l_{t'} = 0, \forall t' \in \{1, 2, 3, \dots, \bar{t} - 1\}$, levando a $\sum_{t'=1}^{\bar{t}-1} d_{t'} = (\bar{t} - 1) \cdot D$. Na rodada \bar{t} , pela

restrição (2.6'), no máximo $d_{\bar{t}} = \bar{t} \cdot D - (\bar{t} - 1) \cdot D = D$ podem ser defendidos. Então, tem-se $l_{\bar{t}} = d_{\bar{t}} - D \leq 0$, levando à contradição com o fato de que um excesso ocorreu em \bar{t} . \square

Lema 2. *Seja s uma solução infactível para o FFMO produzida pelo M-FFMO. Seja \bar{t} a primeira rodada na qual um excesso ocorre. Então, o excesso em \bar{t} não é maior do que o déficit cumulativo das rodadas anteriores do conjunto L do Lema 1, i.e., $l_{\bar{t}} \leq \left| \sum_{t' \in L} l_{t'} \right|$.*

Demonstração. Pela restrição (2.6') temos $\sum_{t'=1}^{\bar{t}-1} d_{t'} \leq (\bar{t} - 1) \cdot D$, o que implica que o déficit em $\bar{t} - 1$ é $\sum_{t'=1}^{\bar{t}-1} d_{t'} - (\bar{t} - 1) \cdot D \leq 0$. Então, o número máximo de vértices que podem ser defendidos em \bar{t} pode ser computado por:

$$\begin{aligned} d_{\bar{t}} \leq D + \left| \sum_{t'=1}^{\bar{t}-1} d_{t'} - (\bar{t} - 1) \cdot D \right| &\Rightarrow l_{\bar{t}} = d_{\bar{t}} - D \leq \left| \sum_{t'=1}^{\bar{t}-1} d_{t'} - (\bar{t} - 1) \cdot D \right| \\ &= \left| \sum_{t'=1}^{\bar{t}-1} (d_{t'} - D) \right| = \left| \sum_{t'=1}^{\bar{t}-1} l_{t'} \right| \end{aligned} \quad (4.1)$$

Pelo Lema 1, existe um conjunto não vazio $L \subseteq \{1, 2, 3, \dots, \bar{t} - 1\}$ com $l_{t'} < 0$, $\forall t' \in L$. Suponha por contradição que $\sum_{t' \in L} l_{t'} + l_{\bar{t}} > 0$. Temos $\sum_{t' \in L} l_{t'} > -l_{\bar{t}}$ e $l_{t'} < 0, \forall t' \in L$, o que leva a:

$$\left| \sum_{t' \in L} l_{t'} \right| < l_{\bar{t}} \quad (4.2)$$

Existe um conjunto $L' = \{1, 2, 3, \dots, \bar{t} - 1\} \setminus L$, tal que $\forall t'' \in L', l_{t''} = 0$, em outras palavras, $L \cup L'$ contem todas as rodadas de 1 até $\bar{t} - 1$. Então, podemos unir os conjuntos L' e L e a soma dos déficits ainda será a mesma:

$$\left| \sum_{t' \in L \cup L'} l_{t'} \right| = \left| \sum_{t' \in L} l_{t'} \right| \quad (4.3)$$

Pela Equação (4.2) sabemos que o lado direito da Equação (4.3) é limitado por $l_{\bar{t}}$, levando a:

$$\left| \sum_{t' \in L \cup L'} l_{t'} \right| < l_{\bar{t}} \quad (4.4)$$

Uma vez que $L \cup L'$ contem todas as $\bar{t} - 1$ rodadas, podemos reescrever a Equação (4.4) da seguinte forma:

$$\left| \sum_{t'=1}^{\bar{t}-1} d_{t'} - (\bar{t} - 1) \cdot D \right| < d_{\bar{t}} - D \quad (4.5)$$

Segundo a Equação (4.1), a Equação (4.5) viola a restrição (2.6'), contradizendo a suposição de que s é uma solução factível. \square

Teorema 3. *Seja s uma solução gerada pelo M-FFMO. É possível obter uma solução factível s^* a partir de s , removendo o excesso de todas as suas rodadas, de forma que as duas soluções tenham o mesmo custo, i.e., $z(s) = z(s^*)$.*

Demonstração. Seja s uma solução factível para o M-FFMO e infactível para o FFMO. Então, existe pelo menos uma rodada t , $1 \leq t \leq T_s$ tal que $l_t > 0$. Seja $L_e = \{t : 1 \leq t \leq T_s : l_t > 0\}$. Fazemos uma prova por indução no número $|L_e|$ de rodadas nas quais um excesso ocorre. Seja $\bar{t} = \min\{L_e\}$, pelo Lema 2 o excesso em $l_{\bar{t}}$ não é maior que $\left|\sum_{t'=\bar{t}}^{\bar{t}-1} l_{t'}\right|$, então podemos transferir esse excesso para as rodadas anteriores com déficit pertencentes ao conjunto L , obtido pelo Lema 1. Fazemos tal transferência alterando a rodada \bar{t} na qual um vértice foi defendido para alguma rodada t' , $t' \in L$ e $l_{t'} < 0$. As transferências são feitas até que $l_{\bar{t}} = 0$, removendo o excesso da rodada \bar{t} . Temos agora uma rodada com excesso a menos e pela hipótese de indução, podemos remover todo o excesso de $|L_e| - 1$ rodadas, conseguindo uma solução factível s^* . Uma vez que não alteramos o número de vértices queimados ou defendidos, temos que $z(s) = z(s^*)$. \square

Um algoritmo polinomial de complexidade $O(n)$ pode ser derivado do Teorema 3, como descrito no Algoritmo 4. O algoritmo basicamente mantém dois apontadores para as primeiras rodada com excesso e déficit, os quais são movidos adiante à medida que excessos e déficits são removidos.

Algoritmo 4: Algoritmo de correção de uma solução gerada pelo M-FFMO.

Entrada: s

```

1   $\bar{t} \leftarrow 2$                                 ▷ Rodada candidata a ser de excesso.
2   $\underline{t} \leftarrow 1$                             ▷ Rodada candidata a ser de déficit.
3  enquanto  $\bar{t} \leq T_s$  faça
4      se  $l_{\bar{t}} > 0$  e  $l_{\underline{t}} < 0$  então
5          ▷ Transfere excesso para a rodada com déficit.
6          enquanto  $l_{\underline{t}} < 0$  e  $l_{\bar{t}} > 0$  faça
7               $s \leftarrow s \setminus \{v, \bar{t}\}$ 
8               $s \leftarrow s \cup \{v, \underline{t}\}$ 
9               $l_{\underline{t}} \leftarrow l_{\underline{t}} + 1$ 
10              $l_{\bar{t}} \leftarrow l_{\bar{t}} - 1$ 
11         ▷ Se não há um excesso nessa rodada, mova para a próxima.
12     se  $l_{\bar{t}} \leq 0$  então
13          $\bar{t} \leftarrow \bar{t} + 1$ 
14     ▷ Se não há um déficit nessa rodada, mova para a próxima.
15     se  $l_{\underline{t}} \geq 0$  então
16          $\underline{t} \leftarrow \underline{t} + 1$ 

```

A complexidade do algoritmo se segue do fato de que T_s é sempre menor do que n e nas Linhas 6 e 7, o número de vértices que têm suas defesas antecipadas nunca excede n .

Com o intuito de ilustrar o funcionamento do Teorema 3, uma instância com $n = 12$ e $D = 1$ é apresentada na Figura 4.1 (adaptado de um dos exemplos de García-Martínez et al. [27]), na qual o M-FFMO gera a seguinte solução infactível s , representada como foi discutido na Seção 2.1: $s \leftarrow \{(1, 0), (2, 1), (3, -1), (4, -1), (7, -2), (8, -2), (9, 3), (10, 3)\}$.

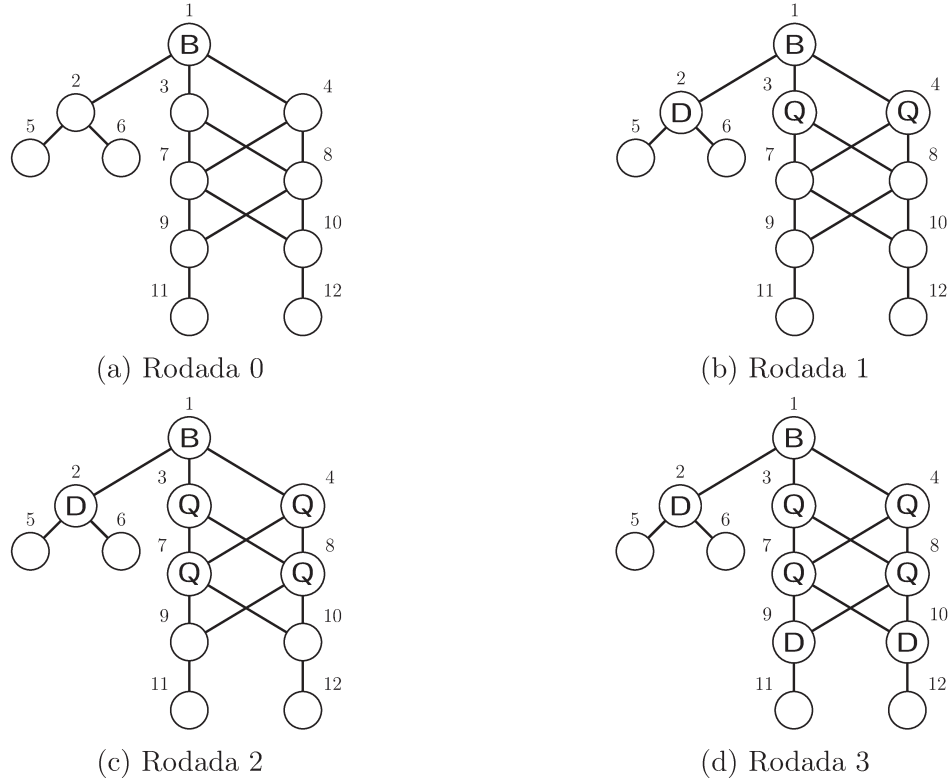


Figura 4.1: Exemplo de solução infactível gerada pelo M-FFMO.

Na rodada 3 (Figura 4.1(d)), os vértices 9 e 10 são defendidos, mas existe apenas um brigadista disponível. Tal fato acontece porque, uma vez que na rodada 2 (Figura 4.1(c)) nenhum vértice foi defendido, então a quantidade de brigadistas disponíveis foi acumulada até a rodada 3, onde todos foram utilizados e a restrição (2.6') ainda foi satisfeita.

Aplicando o Teorema 3, seja o conjunto de rodadas com excesso $L_e \leftarrow \{3\}$ e a primeira rodada com excesso $\bar{t} = 3$. Pelo Lema 1, temos $L \leftarrow \{2\}$, então, podemos mudar a rodada na qual um dos vértices, 9 ou 10, foi defendido para a rodada 2, removendo o excesso da rodada 3. Temos $L_e \leftarrow \emptyset$, como queríamos, uma vez que não existe mais nenhuma rodada com excesso. Assumindo que o vértice 9 foi o escolhido para ser defendido na rodada 2, temos então uma solução factível $s^* \leftarrow \{(1, 0), (2, 1), (3, -1), (4, -1), (7, -2), (8, -2), (9, 2), (10, 3)\}$.

4.2 Matheurística

Nessa seção, descrevemos nossa metodologia para resolver o FFP usando uma matheurística, a qual denotaremos por **MATHEU** no restante do documento. Além da descrição de seus principais componentes, também discutimos uma variação que testamos.

Uma visão geral da nossa estratégia é exibida no Algoritmo 5.

Algoritmo 5: Estrutura geral da MATHEU.

Entrada: h, η, α, ρ

Saída: s^*

- 1 Executa uma heurística h η vezes em modo probabilístico com o valor de α dado, obtendo um conjunto S de soluções distintas.
 - 2 Seleciona ρ soluções de S de maneira a construir um *pool* de soluções P .
 - 3 Executa uma busca local via um modelo PLI em cada solução de P e retorna a melhor solução encontrada s^* .
 - 4 **retorna** s^*
-

Na Linha 1, uma heurística gulosa é executada η vezes. A cada vez, um procedimento construtivo é usado para criar uma solução. Numa iteração desse processo, um elemento para ser adicionado à solução (um vértice para ser defendido) é selecionado utilizando um critério guloso aleatorizado. Após as η execuções, um conjunto S de soluções distintas geradas pela heurística é obtido, conforme é descrito em mais detalhes na Seção 4.2.2.

Na Linha 2, um *pool* de soluções é construído a partir do conjunto S . Nosso objetivo com essa fase é selecionar um número reduzido de soluções diversas e de boa qualidade. Veja a Seção 4.2.4 para uma discussão aprofundada sobre como isso é feito.

Na Linha 3, um procedimento de busca local (descrito na Seção 4.2.3) é aplicado em cada solução de P . A busca local utiliza um resolvidor de PLI para resolver uma versão mais restrita do M-FFMO com algumas modificações, as quais são descritas na Seção 4.1.

4.2.1 Parâmetro T

Como descrito no Capítulo 2, o valor do parâmetro T , usado no modelo FFMO, é um limitante superior para o número de rodadas suficientes para obter uma solução ótima para o FFP. Também observa-se que o número de restrições e variáveis do FFMO cresce proporcionalmente com o valor de T e, conseqüentemente, o tempo de computação. García-Martínez et al. [27] ressalta o impacto do valor do parâmetro na performance do FFMO, também apresentando uma técnica para encontrar um valor apropriado para T usando uma busca incremental, a partir de um valor inicial T^0 .

Na busca local de nossa matheurística (veja a Seção 4.2.3), versões restritas do M-FFMO têm que ser resolvidas, idealmente, muitas vezes. Então, para que tal procedimento seja efetivo, é requerido que T assuma um valor que leve a um bom balanço entre tempo de computação e qualidade de solução.

Nesse trabalho, ajustamos o valor de T segundo dois critérios. Considerando o primeiro, com o objetivo de obter soluções com garantia de otimalidade, definimos um limitante superior $T = \lceil \frac{n}{D} \rceil$. A ideia por trás desse limitante é a observação de que existe uma solução ótima para qualquer instância do FFP na qual, em todas as rodadas, exceto possivelmente a última, exatamente D vértices são defendidos. Isso implica que o fogo deve ser contido em no máximo $\lceil \frac{n}{D} \rceil$ rodadas, visto que isso seria suficiente para defender todos os vértices do grafo. Então, esse é um limitante superior válido para garantir a corretude do FFMO, i.e., se resolvido com esse valor fixo de T , o modelo produz uma solução ótima para o FFP.

Como segundo critério, de modo a gerar soluções sem garantia de otimalidade, mas em um tempo de computação potencialmente pequeno, a seguinte estratégia é adotada. Primeiramente, uma heurística gulosa é executada para construir uma solução inicial s^0 , na qual a contenção do fogo ocorre na T_{s^0} -ésima rodada. Então, definimos $T = \lceil (1+\epsilon) \cdot T_{s^0} \rceil$ para algum valor apropriado de $\epsilon > 0$ (veja a Seção 5.4 para detalhes). Essa estratégia se baseia na observação empírica de que, na maioria dos casos, o número de rodadas para conter o fogo em uma solução ótima é alguma fração das que são requeridas por uma boa solução heurística.

4.2.2 Heurísticas Gulosas

Nós projetamos uma nova estratégia de busca local para resolver o FFP, que funciona do seguinte modo. Dada uma solução inicial factível s^0 , tentamos melhorar sua qualidade, em termos de valor da função objetivo, usando uma versão mais restrita do M-FFMO. Para obter s^0 , executamos uma heurística gulosa várias vezes probabilisticamente. A heurística é escolhida dentre as que foram propostas por García-Martínez et al. [27] e outras que nós projetamos.

Os autores García-Martínez et al. [27] propuseram várias heurísticas gulosas para o FFP. Neste trabalho, desenvolvemos novos critérios gulosos para tentar suprimir algumas desvantagens identificadas nessas heurísticas em testes preliminares. Abaixo, descrevemos nossas heurísticas e em seguida, no Capítulo 5, experimentos computacionais são reportados para avaliar suas eficácias.

Nossas heurísticas também são gulosas no sentido que, a cada rodada, os vértices de maior prioridade são selecionados para serem defendidos. A novidade da nossa proposta são os critérios para atribuição de prioridades aos vértices.

De forma a explicar o funcionamento das heurísticas, seja U_t o conjunto de todos os vértices intocados na rodada t . Os critérios que projetamos podem ser sumarizados como segue:

- **Número de descendentes por uma Busca em Largura (DescBFS):** Seja V' o conjunto de vértices queimados na rodada $t - 1$. Na rodada t , executamos uma Busca em Largura (BFS) [13] no grafo G , iniciando simultaneamente em todos os vértices de V' . Então, obtemos o nível de cada vértice na árvore BFS, o qual é o valor $d(v, V'), \forall v \in U_t$, computado pelo algoritmo da BFS. A seguir, calculamos o número de descendentes $\text{desc}(v), \forall v \in U_t$, segundo a Equação (4.6).

$$\text{desc}(v) = \begin{cases} 0, & \text{se } v \text{ é uma folha} \\ |N'_G(v)| + \left| \bigcup_{v' \in N'_G(v)} \text{desc}(v') \right|, & \text{caso contrário.} \end{cases} \quad (4.6)$$

na qual $N'_G(v)$ é o conjunto $\{v' : v' \in N_G(v) \text{ e } d(v', V') > d(v, V')\}$, i.e., os vértices que são vizinhos de v e estão em um nível inferior ao de v na árvore BFS. Então, atribuímos as prioridades aos vértices em ordem decrescente de acordo com o número de descendentes.

No trabalho de García-Martínez et al. [27], o termo descendente também é utilizado na heurística *Desc*, porém, na definição dos autores, um vértice v é dito ser descendente de u se existe um caminho entre os dois, o que é sempre verdade em um grafo não direcionado conexo e também se $d(u, B) < d(v, B)$, ou seja, v está mais distante do foco de incêndio do que u . Em nossa definição, os descendentes de um vértice v são aqueles que irão queimar caso mais nenhum vértice seja defendido, incluindo v . Essa estratégia é usada para, de alguma forma, mensurar a importância de defender v . Acreditamos que nossa definição de descendentes dada acima é mais apropriada para o FFP, uma vez que a mesma captura mais precisamente a essência do problema.

- **Grau Dinâmico** (*DynDegree*, do inglês *Dynamic Degree*): Em cada rodada t , computamos o grau de cada vértice $v \in U_t$ considerando apenas seus vizinhos que não estão queimados ou defendidos. Então, atribuímos as prioridades aos vértices em ordem decrescente de acordo com o seu respectivo grau, i.e., o vértice com maior grau recebe prioridade mais alta.

Mais uma vez, no trabalho de García-Martínez et al. [27] o grau dos vértices também é considerado em uma tentativa de avaliar o dano que pode resultar ao queimá-los. Porém, os valores dos graus são calculados somente uma vez, antes do início do processo de propagação e contenção do fogo. Como consequência, a natureza dinâmica do processo não é refletida, a qual tende a alterar o potencial ganho em defender um dado vértice.

Seguimos com a análise assintótica de complexidade de tempo das heurísticas *DescBFS* e *DynDegree*, sempre considerando o pior caso. Sejam $n = |V_G|$ e $m = |E_G|$. Na heurística *DescBFS*, o algoritmo da BFS tem complexidade $O(n+m)$ [13]. Visto que cada vértice tem uma vizinhança de tamanho $O(n)$ e cada vizinho tem na ordem de $O(n)$ descendentes, a operação de união entre os conjuntos de descendentes tem complexidade $O(n^2)$ por vértice. Então, o cômputo da Equação (4.6) para todos os vértices tem complexidade $O(n + m + n^3) = O(n^3)$. Considerando a heurística *DynDegree*, a obtenção dos graus dinâmicos dos vértices pela primeira vez tem complexidade $O(n + m)$, dado que a representação de G utilizada foi uma lista de adjacências [13] e que o custo de verificar o estado de um vértice (queimado, defendido ou intocado) é constante, utilizando a representação de solução apresentada na Seção 2.1. Nota-se que nessa heurística, a cada rodada, os graus que precisam ser atualizados são somente os dos vértices nas adjacências daqueles que foram queimados ou defendidos. Então, considerando todas as rodadas, essa heurística tem complexidade $O(m)$ visto que cada aresta é visitada uma única vez.

Como as operações de cômputo dos descendentes e de atualização dos graus dinâmicos são executadas em uma quantidade de rodadas limitada por $\lceil \frac{n}{D} \rceil$ (como discutido na Seção 4.2.1), a heurística *DescBFS* tem complexidade assintótica: $O(\lceil \frac{n}{D} \rceil n^3)$ e a *DynDegree*: $O(n + 2m) = O(n + m)$. O custo de selecionar no máximo D vértices de maior prioridade para serem defendidos foi desconsiderado na análise uma vez que o mesmo não é assintoticamente superior ao custo de cômputo do número de descendentes.

As duas heurísticas mencionadas *DescBFS* e *DynDegree* foram combinadas com a heurística gulosa *Th* descrita por García-Martínez et al. [27], a qual dá maior prioridade

aos vértices que são vizinhos de vértices queimados na rodada anterior (ameaçados). O primeiro critério dá origem à heurística **ThDescBFS**, enquanto o segundo corresponde à **ThDynDegree**. Nessas heurísticas, o número de descendentes (**ThDescBFS**) ou o grau dinâmico (**ThDynDegree**) são utilizados como critérios de desempate dentre todos os vértices ameaçados. A complexidade das heurísticas após tais combinações não é alterada, uma vez que a heurística **Th** apenas restringe quais vértices são candidatos a serem defendidos. Em seu trabalho, os autores García-Martínez et al. [27] apresentaram a definição da heurística **Th**, porém, somente a utilizaram em seus experimentos em conjunto com algum critério de desempate, seja ele o número de descendentes ou o valor do grau (como foi discutido no Capítulo 3). Neste trabalho, consideramos em nossos experimentos também a heurística **Th** sem nenhuma regra preestabelecida para desempates. Dessa forma, a escolha do vértice para ser defendido dentre os que estão ameaçados é arbitrária.

Geralmente, uma heurística gulosa pode ser executada em dois modos distintos: *determinístico* e *probabilístico*. Para o FFP, considerando o modo determinístico, as heurísticas são executadas sempre escolhendo no máximo D vértices intocados de maior prioridade para serem defendidos por rodada, ou seja, é uma escolha puramente gulosa. O modo probabilístico é derivado da fase construtiva da meta-heurística GRASP (como discutido na Seção 2.4) e consiste em repetir por um número fixo de vezes ou por um tempo de computação limitado, uma versão modificada do algoritmo básico no qual a escolha gulosa opera de maneira probabilística. Visando entender tal modo de funcionamento, seja $r(v, h)$ uma função que calcula o *rank* do vértice v de acordo com o critério guloso de alguma heurística h e α um parâmetro real no intervalo $[0, 1]$. Para escolher um vértice a ser defendido, uma LRC é construída a partir do conjunto U_t , seguindo o procedimento descrito na Seção 2.4 e também apresentado no Algoritmo 6.

Algoritmo 6: Construção da LRC para o FFP.

Entrada: U_t, h

Saída: LRC

- | | | |
|---|--|-----------------------|
| 1 | $\underline{r} \leftarrow \min\{r(v, h) : \forall v \in U_t\}$ | ▷ <i>Rank</i> mínimo. |
| 2 | $\bar{r} \leftarrow \max\{r(v, h) : \forall v \in U_t\}$ | ▷ <i>Rank</i> máximo. |
| 3 | $\text{LRC} \leftarrow \{v : r(v, h) \leq \underline{r} + \alpha \cdot (\bar{r} - \underline{r}) \text{ e } v \in U_t\}$ | ▷ Constrói a LRC. |
-

Uma vez que a LRC esteja construída, um vértice é selecionado aleatoriamente (sem reposição) da mesma, seguindo uma distribuição uniforme. Essas escolhas são feitas até que $\min\{|U_t|, D\}$ vértices tenham sido selecionados para serem defendidos. Se não foi contido, o processo de propagação do fogo continua, queimando todos os vértices intocados adjacentes aos queimados. Observa-se que quando $\alpha = 0$, a LRC contém apenas os vértices de menor *rank*, e quando $\alpha = 1$, contém todos os vértices. No primeiro caso, a escolha é completamente gulosa, enquanto que no segundo, é completamente aleatória.

Na **MATHEU**, o modo probabilístico é executado várias vezes, tendo como critério de parada o número de iterações η ou um tempo limite. Na primeira iteração, o modo determinístico é executado e nas iterações restantes, segue-se com o modo probabilístico, com um valor predefinido de α . As soluções distintas concebidas por esse procedimento são mantidas para depois serem utilizadas na construção de um *pool* de soluções P , discutido

em mais detalhes na Seção 4.2.4.

Na seção seguinte, descrevemos nosso procedimento de busca local, o qual utiliza uma das soluções geradas por uma heurística como solução inicial.

4.2.3 Busca Local para uma Solução

Seja s^0 uma solução factível de um *pool* de soluções P , cuja construção será discutida na Seção 4.2.4. Em seguida, fazemos a descrição do nosso procedimento de busca local.

A busca local funciona da seguinte forma. Para algum valor predefinido de k , o conjunto de vértices defendidos em s^0 , denotado por \mathcal{D}_{s^0} é expandido com uma fração dos vértices da sua k -vizinhança, denotada por $N(\mathcal{D}_{s^0}, k)$, selecionados de acordo com alguma regra gulosa. Uma versão restrita da instância original do FFP é considerada, na qual somente os vértices desse conjunto resultante podem ser defendidos. Essa instância modificada é então entregue a um resolvidor PLI. O Algoritmo 7 apresenta uma visão geral desse procedimento de busca local, o qual é discutido em detalhes abaixo.

Algoritmo 7: Busca Local

Entrada: $s^0, k, \sigma, f, \tau, T, G$

Saída: solução s_{bl}

- 1 $\overline{N}_{s^0} \leftarrow \text{ConstroiVizinhanca}(k, \sigma, f, \mathcal{D}_{s^0})$
 - 2 $N \leftarrow \{V_G \setminus \overline{N}_{s^0}\}$ ▷ Vértices que não estão na vizinhança.
 - 3 $C \leftarrow d_{v,t} = 0, \forall v \in N \text{ e } \forall t, 0 \leq t \leq T$ ▷ Conjunto de restrições adicionais.
 - 4 $s_{bl} \leftarrow \text{ResolveModeloPLI}(s^0, C, \tau)$ ▷ Resolve uma versão restrita do M-FFM0.
 - 5 **retorna** s_{bl}
-

O procedimento **ConstroiVizinhanca** na Linha 1 seleciona um subconjunto $\overline{N}_{s^0} \subseteq N(\mathcal{D}_{s^0}, k)$ de vértices, tendo como parâmetros:

- k : Determina o comprimento do caminho para considerar se um vértice v é vizinho dos vértices defendidos em s^0 , i.e., $v \in N(\mathcal{D}_{s^0}, k)$.
- σ : Valor real no intervalo $[0, 1]$ que determina a fração de vértices para serem selecionados do conjunto $N(\mathcal{D}_{s^0}, k)$.
- $f : v \mapsto \mathbb{N}^*$: Função que atribui um valor para um vértice v de acordo com algum critério guloso. Consideramos cinco funções diferentes, com seus respectivos critérios gulosos e complexidade assintótica:
 - f -Deg: Maior grau, desempates são realizados arbitrariamente. Complexidade: $O(n)$.
 - f -NeiDeg: Maior grau e, no caso de empates, considera o maior grau de seus vizinhos. Complexidade: $O(n + m)$.
 - f -DegDesc: Maior grau e, no caso de empates, considera o número de descendentes, computado segundo García-Martínez et al. [27]. Complexidade: $O(n)$.

- *f-Prox*: Menor valor de $d(v, B)$, ou seja, quanto menor a distância para os vértices em B , maior é a prioridade. Complexidade: $O(n|B|)$.
- *f-Desc*: Maior número de descendentes, calculado segundo García-Martínez et al. [27]. Complexidade: $O(n)$.

Adicionamos o prefixo *f*- com o intuito de diferenciar as funções das heurísticas apresentadas na Seção 4.2.2.

O conjunto \overline{N}_{s^0} é construído com os $\sigma \cdot |N(\mathcal{D}_{s^0}, k)|$ vértices de maior prioridade do conjunto $N(\mathcal{D}_{s^0}, k)$. As prioridades são atribuídas segundo a função f . Para considerações futuras, é importante tomar nota de que, dado um valor $\sigma' > \sigma$, a vizinhança com $\sigma' \cdot |N(\mathcal{D}_{s^0}, k)|$ vértices inclui a vizinhança com σ , considerando os mesmos valores dos parâmetros s^0 e k , uma vez que o procedimento de construção da vizinhança é determinístico. Na Linha 2, o conjunto N é formado pelos vértices que não estão em \overline{N}_{s^0} . Em seguida, na Linha 3, um novo conjunto de restrições C é criado, o qual fixa em 0 as variáveis $d_{v,t}$, $\forall v \in N$ e $0 \leq t \leq T$. O procedimento de busca local é genérico suficiente para qualquer valor de T , assim, no Capítulo 5, fazemos uma discussão sobre como ajustamos este parâmetro. Ademais, uma versão mais restrita do M-FFMO (denotada por RM-FFMO) com as restrições adicionais é resolvida na Linha 4, com um tempo limite τ . Ao final, a solução s_{bl} é retornada.

O intuito de usar tal estratégia é o seguinte. Uma vez que restringimos o conjunto de vértices que podem ser defendidos para somente aqueles que estão no conjunto \overline{N}_{s^0} , as variáveis d dos demais vértices são fixadas em zero e conseqüentemente, o espaço de soluções do problema é reduzido drasticamente. Então esperamos que seja muito mais rápido resolver o RM-FFMO do que o M-FFMO, tornando-o uma opção viável para ser utilizado como busca local.

Na prática, sempre passamos a solução s^0 como entrada para o resolvedor PLI, a qual será usada como incumbente. Tal técnica é conhecida como *warm start* e permite que porções do espaço de busca sejam eliminadas, resultando em árvores de enumeração menores [14] e, conseqüentemente, tempos de computação menores.

A Figura 4.2 apresenta um exemplo de vizinhança de uma solução s^0 (Figura 4.2(a)) para uma instância com $n = 7$ e $D = 1$. Considerando $k = 2$, na Figura 4.2(b) tem-se destacada em cinza a 2-vizinhança do conjunto \mathcal{D}_{s^0} , constituída pelos vértices rotulados por **V** e também os defendidos em s^0 . Em seguida, na Figura 4.2(c), $\sigma \cdot |N(\mathcal{D}_{s^0}, 2)| = 0.5 \cdot 4 = 2$ vértices são escolhidos dentre os do conjunto $N(\mathcal{D}_{s^0}, 2)$ segundo algum critério guloso arbitrário. Note que os vértices de \mathcal{D}_{s^0} também estão incluídos na vizinhança final \overline{N}_{s^0} . Resolvendo até a otimalidade o submodelo no qual somente os vértices de \overline{N}_{s^0} podem ser defendidos, obtém-se a solução da Figura 4.2(d). Para esse exemplo em particular, essa também é a solução ótima da instância original.

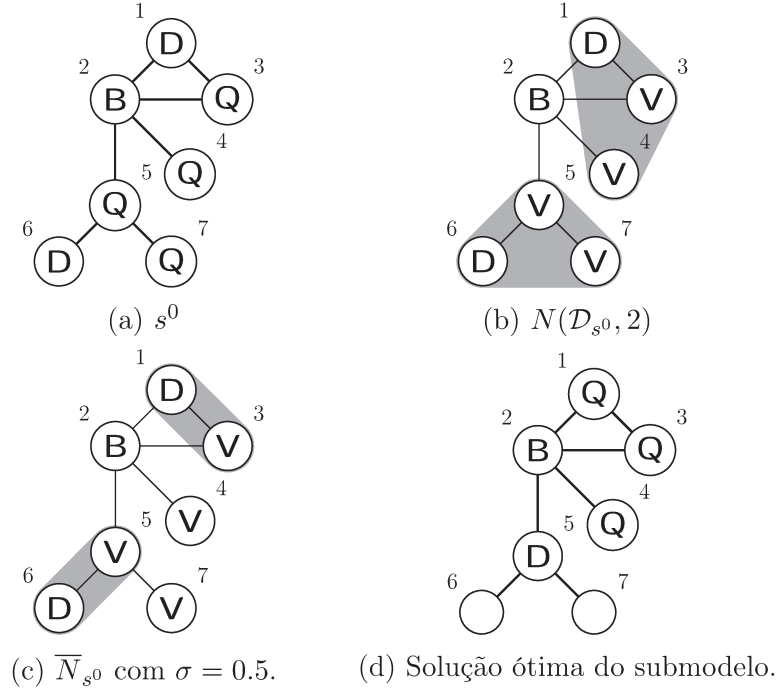


Figura 4.2: Vizinhança de uma solução s^0 .

Em relação à complexidade assintótica das funções utilizadas para atribuição de prioridades, algumas das informações necessárias podem ser calculadas somente uma vez por instância. O grau dos vértices é obtido em $O(n+m)$, utilizando a representação por lista de adjacências. Calcula-se o comprimento de todos os caminhos mínimos através de n buscas em largura, cada uma utilizando um vértice diferente como raiz, então, a complexidade é $O(n(n+m))$. O número de descendentes de cada vértice, segundo a definição de García-Martínez et al. [27], também pode ser computado uma única vez. Como visto no Capítulo 3, um vértice v é considerado descendente de u se $d(v, B) > d(u, B)$, assumindo que o grafo seja conexo. Uma vez que já se tenham os comprimentos de todos os caminhos mínimos, são $O(n|B|)$ operações para computar os valores de $d(v, B)$, $\forall v \in V$. De maneira a obter o número de descendentes é necessário verificar todos os pares u, v de vértices, o que tem complexidade $O(n^2)$. Então, a complexidade final é $O(n|B| + n^2) \in O(n^2)$.

Considerando a complexidade assintótica do procedimento **ConstroiVizinhanca**, o conjunto $N(\mathcal{D}_{s^0}, k)$ é computado em $O(n+m)$, através de uma adaptação do procedimento de BFS, descrita como segue. Iniciamos a busca com todos os vértices do conjunto \mathcal{D}_{s^0} na fila de visitação e então, prosseguimos com a BFS até o nível de profundidade k . Os vértices da árvore resultante, exceto os do conjunto \mathcal{D}_{s^0} , constituem o conjunto $N(\mathcal{D}_{s^0}, k)$. O mesmo procedimento pode ser naturalmente estendido para o caso de k -vizinhança estrita. Seja $l = \lceil \sigma \cdot |N(\mathcal{D}_{s^0}, k)| \rceil$. Uma vez que $|N(\mathcal{D}_{s^0}, k)| \in O(n)$, para selecionar os l vértices de maior prioridade, a l -ésima estatística de ordem é obtida em $O(n)$ e em seguida, é utilizada como pivô no procedimento de partição em $N(\mathcal{D}_{s^0}, k)$, com complexidade $O(n)$ [13]. Como as informações da função f já são calculadas previamente, consultá-las leva tempo constante. A complexidade final do procedimento **ConstroiVizinhanca** é $O((n+m) + n) \in O(n+m)$.

O procedimento **ConstroiVizinhanca** mencionado previamente utiliza a função f

para atribuir prioridades aos vértices, os quais serão selecionados posteriormente em ordem decrescente de acordo com suas prioridades, até que o tamanho desejado da vizinhança seja atingido. A seguir, no Algoritmo 8, uma modificação de tal procedimento é apresentada, no qual vértices próximos ao conjunto \mathcal{D}_{s^0} tem preferência para serem adicionados a \overline{N}_{s^0} , usando f apenas para desempates.

Algoritmo 8: Construção da vizinhança.

Entrada: $k, \sigma, f, \mathcal{D}_{s^0}$
Saída: \overline{N}_{s^0}

```

1  $\overline{N}_{s^0} \leftarrow \mathcal{D}_{s^0}$  ▷ Inclui os vértices defendidos na vizinhança.
2  $\mu \leftarrow \sigma \cdot |N(\mathcal{D}_{s^0}, k)| + |\mathcal{D}_{s^0}|$  ▷ Tamanho desejado da vizinhança.
3  $\ell \leftarrow 1$  ▷ Distância atual para considerar vértices vizinhos.
4  $\mu' \leftarrow |\overline{N}_{s^0}| + |N^*(\mathcal{D}_{s^0}, \ell)|$  ▷ Tamanho da vizinhança atual.
5 enquanto  $\mu' < \mu$  faça
6    $\overline{N}_{s^0} \leftarrow \overline{N}_{s^0} \cup N^*(\mathcal{D}_{s^0}, \ell)$  ▷ Adiciona a  $\ell$ -vizinhança estrita de  $\mathcal{D}_{s^0}$ .
7    $\ell \leftarrow \ell + 1$ 
   ▷ Se todas as vizinhanças já foram exploradas.
8   se  $\ell = k + 1$  então retorna  $\overline{N}_{s^0}$ 
   ▷ Tamanho de  $\overline{N}_{s^0}$  caso a  $\ell$ -vizinhança estrita atual seja adicionada.
9    $\mu' \leftarrow |\overline{N}_{s^0}| + |N^*(\mathcal{D}_{s^0}, \ell)|$ 
   ▷ Ordena as vizinhanças restantes segundo a função  $f$ .
10  $S \leftarrow \text{ordena}(\bigcup_{\ell}^k N(\mathcal{D}_{s^0}, \ell), f)$ 
   ▷ Obtém os  $\min\{|S|, (m - |\overline{N}_{s^0}|)\}$  vértices de maior prioridade.
11  $S' \leftarrow \{s_i \in S : 1 \leq i \leq \min\{|S|, (\mu - |\overline{N}_{s^0}|)\}\}$ 
12  $\overline{N}_{s^0} \leftarrow \overline{N}_{s^0} \cup S'$  ▷ Constrói a vizinhança final.
13 retorna  $\overline{N}_{s^0}$ 

```

Inicializações básicas são realizadas nas Linhas 1 a 4. Nas Linhas 5 a 9, para $\ell \leq k$, unimos a ℓ -vizinhança estrita do conjunto \mathcal{D}_{s^0} a \overline{N}_{s^0} , enquanto a cardinalidade do conjunto resultante de tal operação não exceda μ . Se tal excesso não ocorre para nenhum valor de ℓ , o conjunto \overline{N}_{s^0} é retornado na Linha 8. Caso contrário, na Linha 10, obtemos o conjunto S ordenando decrescentemente as ℓ -vizinhanças restantes que ainda não foram exploradas. Nesse caso, o critério de ordenação segue as prioridades atribuídas segundo a função f . Então, na Linha 11 selecionamos uma quantidade de vértices de S suficiente para alcançar o tamanho μ da vizinhança e, na sequência, tais vértices são adicionados ao conjunto \overline{N}_{s^0} . Ao final, a vizinhança é retornada.

Nota-se que o Algoritmo 8 é uma relaxação do que é utilizado no procedimento `ConstroiVizinhanca`, uma vez que deixamos de ordenar todos os vértices do conjunto $N(\mathcal{D}_{s^0}, k)$. A ordenação só é feita se ainda é possível adicionar alguns dos vértices da vizinhança ainda não explorada. O conjunto de funções f que podem ser usadas nesse algoritmo se restringe àquelas que podem ser computadas antes do processo de obtenção de uma solução, como por exemplo, o grau ou o número de descendentes dos vértices. A complexidade assintótica do pior caso do procedimento original é mantida.

Nosso objetivo com tal algoritmo é manter os vértices mais próximos do conjunto \mathcal{D}_{s^0}

na vizinhança, baseando em uma observação empírica de que os vértices defendidos em uma solução heurística estão relativamente próximos daqueles defendidos em uma solução ótima, como será discutido em mais detalhes no Capítulo 5.

4.2.4 *Pool* de soluções

Nosso procedimento de busca local é custoso em termos de recursos computacionais, visto que consiste na resolução de um modelo PLI. Assim, desejamos executá-lo por um número controlado de vezes. Fazemos isso mantendo um *pool* com um número pequeno de soluções diversas entre si e de boa qualidade. A forma como é feita a construção desse *pool* é descrita a seguir.

Seja s uma solução factível e completa e \overline{N}_s uma vizinhança de s obtida segundo o procedimento da Seção 4.2.3. Primeiro, mantemos em um conjunto S todas as soluções distintas geradas por várias execuções de uma dentre as heurísticas descritas na Seção 4.2.2, em modo probabilístico. Duas soluções s_i e s_j são ditas distintas se: $\oplus(\overline{N}_{s_i}, \overline{N}_{s_j}) > 0$, onde \oplus é uma função que retorna o número de vértices na diferença simétrica dos conjuntos \overline{N}_{s_i} e \overline{N}_{s_j} , ambos construídos utilizando o procedimento **ConstroiVizinhanca** apresentado na Seção 4.2.3.

Seja $q_0 = \min\{z(s) : \forall s \in S\}$ e $q_5 = \max\{z(s) : \forall s \in S\}$ e considere os quartis $q_1 \leq q_2 \leq q_3$ de S relativos aos valores $z(s), \forall s \in S$. Para $i \in \{1, 2, 3, 4\}$, define-se o conjunto $S_i \leftarrow \{s | z(s) \in [q_{5-i}, q_{5-i+1}], \forall s \in S\}$, de forma que S_1 contenha o primeiro quarto de soluções com melhor qualidade, S_2 o segundo quarto e assim sucessivamente.

O Algoritmo 9 descreve o procedimento para construir um *pool* P com no máximo ρ soluções. Seja \bar{s} a solução com melhor função objetivo em S , ou seja, $\bar{s} \leftarrow \operatorname{argmax}\{z(s) : s \in S\}$. Primeiro, inicializamos P como um conjunto unitário tendo \bar{s} como seu único elemento. Então, iteramos nos conjuntos S_i , procurando por soluções para incluir no *pool*. Inicialmente, as primeiras ρ soluções visitadas são adicionadas a P (Linha 5). Se P já está cheio, na Linha 7, a solução s' em P que tem a vizinhança $\overline{N}_{s'}$ de menor diferença simétrica em relação à vizinhança de melhor solução $\overline{N}_{\bar{s}}$ é obtida. Em seguida, nas Linhas 8 e 10 testamos se a solução corrente s_k deve ser adicionada a P . Isso acontece somente se a diferença simétrica entre $\overline{N}_{\bar{s}}$ e \overline{N}_{s_k} é maior do que a de $\overline{N}_{s'}$. Nesse caso, trocamos s' por s_k em P . Uma solução é sempre inserida no final de P , de forma que ao término de sua construção, as soluções estão ordenadas em ordem decrescente de acordo com o valor de função objetivo (da melhor para a pior). Essas decisões favorecem a inclusão de soluções diversas em P , mas não necessariamente com boa qualidade. De forma a atender esse segundo requerimento, na Linha 2 iteramos nos quartis de soluções em ordem (das melhores para piores), de forma que, somente prosseguimos para S_{i+1} se o *pool* não foi preenchido com S_i .

Para fins práticos, as operações de diferença simétrica entre dois conjuntos podem ser feitas em $O(1)$, representando as vizinhanças por um vetor de bits e utilizando a operação *bitwise xor*. Então, a complexidade assintótica do algoritmo de construção do *pool* é $O(|S|\rho(n+m))$, dado que cada solução de S é inspecionada uma vez no pior caso e se o *pool* está cheio, são feitas $\rho(n+m)$ operações para encontrar as vizinhanças e selecionar o elemento de menor diferença simétrica, por solução.

Algoritmo 9: Algoritmo para construção do *pool* de soluções.

Entrada: $\{S_i : i \in \{1, 2, 3, 4\}\}, \bar{s}$
Saída: P

```

1  $P \leftarrow \{\bar{s}\}$  ▷ Inicia o pool com a solução incumbente.
2 para  $i \in \{1 \dots 4\}$  faça
3   para  $s_k \in S_i$  faça
4     ▷ Caso o pool não esteja cheio.
5     se  $|P| < \rho$  então
6        $P \leftarrow P \cup \{s_k\}$ 
7     senão
8       ▷ Solução com vizinhança de menor diferença simétrica em relação
9       a  $\bar{N}_{\bar{s}}$ 
10       $s' \leftarrow \operatorname{argmin}\{\oplus(\bar{N}_s, \bar{N}_{\bar{s}}) : s \in P; s \neq \bar{s}\}$ 
11      se  $\oplus(\bar{N}_{s_k}, \bar{N}_{\bar{s}}) > \oplus(\bar{N}_{s'}, \bar{N}_{\bar{s}})$  então
12         $P \leftarrow P \setminus \{s'\}$ 
13         $P \leftarrow P \cup \{s_k\}$ 
14 se  $|P| = \rho$  então
15   pare
16 retorna  $P$ 

```

4.2.5 Busca Local Adaptativa no *Pool* de Soluções

O parâmetro σ tem um impacto importante no desempenho da busca local do Algoritmo 7, uma vez que determina o tamanho da vizinhança a ser explorada. Um valor pequeno pode restringir significativamente o espaço de busca do RM-FFMO, possivelmente reduzindo as chances de encontrar soluções com boa qualidade. Por outro lado, um valor grande pode fazer com que o tamanho do RM-FFMO (em termos de número de variáveis e restrições) seja quase igual ao do M-FFMO, eventualmente requerendo uma quantidade inviável de recursos computacionais. Para lidar com tal situação, desenvolvemos uma estratégia adaptativa para ajustar o valor de σ , de maneira a criar um equilíbrio entre o tempo de computação e a qualidade das soluções produzidas pela busca local. Essa estratégia é implementada através do Algoritmo 10, o qual é executado logo após a busca local ter retornado uma solução s . A ideia é aumentar o tamanho da vizinhança na próxima chamada da busca local, caso s seja ótima, i.e., quando o RM-FFMO tiver sido resolvido dentro do tempo limite permitido. A expectativa é que expandindo o espaço de busca, também aumentemos as chances de encontrar soluções melhores. Do contrário, caso s não seja ótima, o tamanho da vizinhança é reduzido para aumentar as chances de resolver o RM-FFMO na próxima rodada à otimalidade.

Algoritmo 10: Atualização do valor de σ .

Entrada: σ, s
Saída: Novo valor de σ

```

1 se  $s$  é ótima então retorna  $\min\{1, \sigma + 0.1\}$ 
2 senão retorna  $\max\{0, \sigma - 0.1\}$ 

```

A seguir, no Algoritmo 11, descrevemos nosso procedimento para executar a busca local nas soluções de P , enquanto o tamanho da vizinhança é alterado adaptativamente.

Seja s_P^* a solução em P com o melhor valor de função objetivo e τ' o tempo de computação gasto pela matheurística antes da execução do Algoritmo 11. As inicializações básicas são feitas nas Linhas 2 a 4. Sempre inicializamos σ com o valor de 0.5, i.e., 50% da vizinhança. Na Linha 1, removemos s_P^* de P e invertemos a ordem das soluções restantes, de forma que as soluções fiquem ordenadas crescentemente de acordo com o valor da função objetivo (da pior para a melhor).

Espera-se que a vizinhança de uma solução com boa qualidade contenha soluções similares e, potencialmente, também com boa qualidade. Com o intuito de garantir que a busca local seja executada com a solução s_P^* com um tempo limite grande o suficiente, na Linha 5, reservamos metade do tempo limite para sua execução, denotado por $\tau^{s_P^*}$, o que implica que o valor de τ deve ser decrementado pela mesma quantidade.

Nas Linhas 7 a 14, a busca local é executada em cada solução s de P' . Além de obter diferentes soluções, potencialmente com boa qualidade, também desejamos ajustar o valor de σ nesse laço do algoritmo, antes de usá-lo na busca local com a solução s_P^* . Devido à ordenação na Linha 2, iteramos nas soluções de P' da pior para a melhor, aprimorando o ajuste do valor de σ à medida que movemos em direção às melhores soluções. Na Linha 11 a solução incumbente s^* é atualizada somente se s' , a solução encontrada na vizinhança de s , tiver custo maior e for completa. Nas Linhas 16 e 17, a busca local é executada na melhor solução do *pool*.

O valor de Γ na Linha 19 indica se um reinício (discutido logo abaixo) no valor de σ foi executado ou não. Nas Linhas 22 a 33, se ainda resta algum tempo de computação disponível, a busca local é executada iterativamente, iniciando com a solução incumbente. Nesse procedimento, a última solução factível produzida pela busca local nas iterações anteriores (denotada por s'') é passada como entrada para a busca local na atual. As iterações de tal procedimento terminam quando não existe diferença simétrica entre os conjuntos $\overline{N}_{s'}$ e $\overline{N}_{s''}$ na Linha 27, ou quando o valor de σ não foi atualizado e um reinício já foi executado. Na Linha 30, se as condições necessárias são atendidas, um reinício é executado. Tal operação consiste em alterar σ para seu valor inicial de 0.5, e ocorre quando o valor de σ não foi alterado desde a última iteração após uma atualização feita no Algoritmo 10, ou seja, quando σ chega a seu valor máximo de um. A razão de fazer tal reinício é a seguinte. Nosso procedimento iterativo permite que a busca local seja executada novamente mesmo quando uma solução s' encontrada na vizinhança da solução anterior s'' não traga alguma melhoria, o que a difere da maioria dos procedimentos de busca local determinísticos. A ideia é que mesmo que a solução s' seja de pior qualidade, sua vizinhança $\overline{N}_{s'}$ pode ser diferente da vizinhança $\overline{N}_{s''}$, o que pode fazer com que soluções diferentes e potencialmente, de melhor qualidade, sejam encontradas. Dito isso, quando o valor de σ chega a um e não é mais alterado pelo Algoritmo 10, o mesmo seria utilizado para explorar $\overline{N}_{s'}$. Porém, visto que a vizinhança de s' difere daquela de s'' , tal valor pode impedir que essa vizinhança seja explorada até encontrar sua solução localmente ótima no tempo limite dado, devido ao seu tamanho. Então, atribuir um valor reduzido para σ pode ser benéfico. Por fim, a melhor solução encontrada é retornada na Linha 34.

Algoritmo 11: Busca Local com tamanho adaptativo de vizinhança.

Entrada: $P, k, f, \tau, \tau', s_P^*, \epsilon, G$
Saída: s^*

```

1  $P' \leftarrow \text{Inverte}(P \setminus s_P^*)$  ▷ Modifica o pool, removendo sua melhor solução.
2  $s^* \leftarrow s_P^*$  ▷ Inicializa a solução incumbente.
3  $\sigma \leftarrow 0.5$  ▷ Tamanho inicial da vizinhança.
4  $\rho' \leftarrow |P'|$  ▷ Contador de soluções do pool não visitadas.
5  $\tau^{s_P^*} \leftarrow \tau/2$ 
6  $\tau \leftarrow \tau/2$  ▷ Atualiza o tempo limite atual para as demais soluções.
7 para  $s \in P'$  faça
8    $\tau^s \leftarrow (\tau - \tau' - \tau^{s_P^*})/\rho'$  ▷ Tempo limite para cada solução de  $P'$ .
9    $T \leftarrow \lceil (1 + \epsilon) \cdot T_s \rceil$ 
10   $s' \leftarrow \text{BuscaLocal}(s, k, \sigma, f, \tau^s, T, G)$ 
11   $s^* \leftarrow \text{argmax}\{z(s'), z(s^*)\}$ 
12  Executa Algoritmo 10 com  $s'$  e  $\sigma$ 
13   $\rho' \leftarrow \rho' - 1$ 
14  Atualiza tempo de computação  $\tau'$ .
15  $T \leftarrow \lceil (1 + \epsilon) \cdot T_{s_P^*} \rceil$ 
16  $s' \leftarrow \text{BuscaLocal}(s_P^*, k, \sigma, f, \tau^{s_P^*}, T, G)$  ▷ Executa a busca local com  $s_P^*$  e um
   valor ajustado de  $\sigma$ .
17  $s^* \leftarrow \text{argmax}\{z(s'), z(s^*)\}$ 
18  $s' \leftarrow s^*$ ;  $s'' \leftarrow s'$ 
19  $\Gamma \leftarrow \text{falso}$ 
20  $\sigma' \leftarrow \sigma$  ▷ Mantem uma referência para o valor de  $\sigma$  da última iteração.
21 Atualiza tempo de computação  $\tau'$ .
22 enquanto  $\tau' < \tau$  faça
23    $T \leftarrow \lceil (1 + \epsilon) \cdot T_{s''} \rceil$ 
24    $s' \leftarrow \text{BuscaLocal}(s'', k, \sigma, f, \tau - \tau', T, G)$ 
25    $s^* \leftarrow \text{argmax}\{z(s'), z(s^*)\}$ 
26   Executa Algoritmo 10 com  $s'$  e  $\sigma$ 
27   se  $\oplus(\overline{N}_{s'}, \overline{N}_{s''}) = 0$  ou  $(\sigma = \sigma' \text{ e } \Gamma = \text{verdadeiro})$  então pare
28   se  $\sigma = \sigma'$  e  $\Gamma = \text{falso}$  então
29      $\sigma \leftarrow 0.5$ 
30      $\Gamma \leftarrow \text{verdadeiro}$ 
31    $s'' \leftarrow s'$  ▷ Atualiza solução previamente obtida.
32    $\sigma' \leftarrow \sigma$ 
33   Atualiza tempo de computação  $\tau'$ .
34 retorna  $s^*$ 

```

Nós utilizamos essa estratégia iterativa ao final para intensificar a exploração da solução incumbente, uma vez que acredita-se que soluções com boa qualidade estariam em sua vizinhança.

4.2.6 Busca Local com Diversificação e Intensificação

Nessa seção, introduzimos outra estratégia de busca local para o *pool* de soluções que utiliza o procedimento da Seção 4.2.3, a qual introduz mecanismos para construir vizinhanças mais diversas de soluções e também intensificar a busca com a solução incumbente. Uma visão geral dessa estratégia (denotada por MATHEU-DI) é apresentada no Algoritmo 12.

Algoritmo 12: Busca Local com Intensificação e Diversificação.

Entrada: $P, k, f, \tau, \tau', \epsilon, \mathcal{V}$
Saída: s^*

```

1  $s^* \leftarrow P[1]$  ▷ Inicializa a solução incumbente.
2 para  $s \in P$  faça
3    $p \leftarrow 1$ 
4    $r \leftarrow |\mathcal{V}|$ 
5    $s' \leftarrow s$  ▷ Solução atualmente sendo explorada.
   ▷ Enquanto houver tempo de computação disponível.
6   enquanto  $\tau' \leq \tau$  faça
7      $q \leftarrow \lfloor \frac{(p+r)}{2} \rfloor$  ▷ Meio do conjunto de valores de  $\sigma$ .
8      $\sigma \leftarrow \mathcal{V}[q]$ 
9      $\tau^{s'} \leftarrow \frac{(\tau - \tau')}{\lceil \lg(r-p+1) \rceil}$  ▷ Tempo máximo de computação para essa solução.
10     $T \leftarrow \lceil (1 + \epsilon) \cdot T_{s'} \rceil$  ▷ Cômputo do limitante superior  $T$ .
11     $s'' \leftarrow \text{BuscaLocal}(s', k, \sigma, f, \tau^{s'}, T, G)$  ▷ Execução da busca local.
    ▷ Aumenta a vizinhança.
12    se  $s''$  é ótima então  $p \leftarrow q + 1$ 
    ▷ Reduz a vizinhança.
13    senão  $r \leftarrow q - 1$ 
14    se  $p \geq r$  então
    ▷ Caso as vizinhanças sejam iguais, retorna ao laço da Linha 2
15    se  $\oplus(\overline{N}_{s'}, \overline{N}_{s''}) = 0$  então Vá para a Linha 2.
16     $p \leftarrow 1$ 
17     $r \leftarrow |\mathcal{V}|$ 
    ▷ Caso as soluções sejam diferentes, começa a busca por
    diferentes vizinhanças da solução incumbente (intensificação).
18    se  $s' \neq s^*$  então  $s' \leftarrow s^*$ 
    ▷ Caso as soluções sejam iguais, retorna ao laço da Linha 2.
19    senão Vá para a Linha 2.
20     $s^* \leftarrow \text{argmax}\{z(s''), z(s^*)\}$  ▷ Atualiza solução incumbente.
21    Atualiza o tempo decorrido  $\tau'$ .
22 retorna  $s^*$ 
```

Seja $\mathcal{V} \leftarrow \{\sigma_1, \dots, \sigma_m\}$ um conjunto com m valores predefinidos para o parâmetro σ , ordenados crescentemente. Para cada solução s do *pool*, fazemos uma busca binária [13]

nos valores de \mathcal{V} . Nas Linhas 3 a 4, inicializamos os marcadores de início e fim do conjunto de valores de vizinhanças, p e r , respectivamente. Nas Linhas 6 a 21, a busca local é executada em uma mesma solução s' com diferentes valores de σ . Primeiramente, obtemos o valor σ , correspondente ao valor localizado na posição do meio de \mathcal{V} , nas Linhas 7 a 8. Em seguida, na Linha 9, o tempo limite para realizar a busca na vizinhança da solução s' atual é calculado. No denominador, temos $r - p + 1$, que corresponde ao número de valores de σ ainda não explorados em \mathcal{V} e como estamos usando uma busca binária, apenas $\lceil \lg(r - p + 1) \rceil$ valores serão explorados efetivamente. Logo em seguida, executamos a busca local com valor computado de T , retornando a solução s'' .

Nas Linhas 12 a 13, calculamos qual metade do conjunto \mathcal{V} será explorada na próxima iteração. Se s'' é localmente ótima, significa que, no tempo limite dado, o resolvidor foi capaz de explorar o espaço de busca efetivamente e provar sua otimalidade. Isso é um indicativo de que podemos executar a busca local com vizinhanças maiores (metade direita de \mathcal{V}) e, possivelmente, encontrar soluções com melhor qualidade. Caso contrário, fazemos a busca local em vizinhanças menores (metade esquerda de \mathcal{V}), com o intuito de encontrar a solução localmente ótima do RM-FFMO. A busca binária entre os valores de \mathcal{V} é possível uma vez que estes estão ordenados e também pela propriedade, apresentada na Seção 4.2.3, de que vizinhanças maiores contêm as menores. Assim, se a solução localmente ótima foi encontrada para algum σ_k , não é necessário explorar os valores de σ_i para $i < k$. Esse processo de explorar diferentes vizinhanças de uma mesma solução é o que traz a característica de diversificação do nosso processo, uma vez que, potencialmente, diferentes soluções serão geradas para diferentes valores de σ .

Nas Linhas 14 a 19, se a busca binária chegou ao fim, verificamos se a vizinhança da solução da iteração corrente difere da vizinhança da solução da última iteração. Caso não exista diferença, paramos a exploração em torno da solução s' . Caso contrário, iniciamos um processo de intensificação na solução incumbente, explorando diferentes vizinhanças da mesma com o intuito de encontrar alguma solução vizinha com melhor qualidade.

4.3 Análise Estatística de Resultados de Múltiplas Estratégias em Múltiplas Instâncias

Nesse trabalho, quando fizemos comparações entre diferentes estratégias em múltiplas instâncias, utilizamos testes estatísticos não paramétricos. O intuito de tais testes é verificar se a diferença entre duas ou mais estratégias é estatisticamente significativa ou não. Os resultados desses testes são úteis para fornecer um embasamento científico mais rigoroso às comparações feitas. O trabalho de Demšar [16] sugere uma metodologia para fazer tais comparações no contexto de aprendizado de máquina, onde diversos testes estatísticos são utilizados para identificar se existe ou não diferença estatisticamente significativa (DES) entre vários classificadores em múltiplos *datasets*. Nessa seção, discutimos brevemente tais testes e a forma de interpretação de seus resultados.

A fim de facilitar o entendimento da mecânica dos testes, seguiremos com alguns dados de exemplo ao longo da seção. Tais dados constituem-se de valores de função objetivo de sete heurísticas genéricas executadas em 1200 instâncias do FFP. Também exibiremos

trechos de código na linguagem de programação R¹, com o intuito de expor as ferramentas que a linguagem possui para auxiliar na análise dos dados.

Primeiramente, iniciada uma sessão do R, instalamos e carregamos o pacote `scmamp`², o qual possui a implementação dos testes estatísticos e gráficos apresentados no trabalho de Demšar [16]. Podemos instalá-lo segundo o Código 4.1.

```
> install.packages('scmamp')
> library(scmamp)
```

Código 4.1: Instalação do pacote `scmamp`

Os dados podem estar em um arquivo `csv`, o qual tem seus campos geralmente separados por um caractere de vírgula (,). Para nossa análise, assume-se que a primeira linha do arquivo contenha o cabeçalho com cada coluna indicando o nome das heurísticas. Cada uma das linhas subsequentes possui os resultados (valor de função objetivo) de cada instância, onde cada coluna corresponde ao resultado da respectiva heurística. Assumindo que o arquivo tenha nome `output.csv`, vê-se no Código 4.2 uma das maneiras de fazer sua leitura em R.

```
> df <- read.csv('output.csv', sep=',', header=T)
```

Código 4.2: Leitura do arquivo de resultados.

Nossos dados de resultados das heurísticas estão agora em um *dataframe* de mesma estrutura do arquivo `csv`. Cada coluna corresponde aos resultados de uma heurística na instância da respectiva linha. Por praticidade, chamaremos esse *dataframe* de `df` no restante dessa seção. Pode-se ter uma visão dos primeiros dados de `df` com o comando `head`, como apresentado no Código 4.3.

```
> head(df)
  H1 H2 H3 H4 H5 H6 H7
1 92 10 93 93 93 93 93
2 92 29 97 97 97 97 97
3 92 40 98 98 98 98 98
4 92 50 99 99 99 99 99
5 92 55 99 99 99 99 99
6 44 25 37 31 34 34 31
```

Código 4.3: *Dataframe* com os dados de resultados das heurísticas.

Para todos os testes, consideramos um nível de confiança de 95% com a hipótese nula de que não existe DES entre todas as heurísticas. Todos os testes se baseiam em *ranks*, de forma que a heurística de *rank* k é aquela com o k -ésimo melhor resultado para uma instância.

O primeiro teste a ser utilizado é o de Iman-Davenport para identificar se existe DES entre as heurísticas. A primeira informação que precisamos é o valor do *rank* médio R_j de cada uma das heurísticas, calculado da seguinte forma. Seja r_i^j o *rank* da j -ésima heurística na i -ésima instância (*rank* médio é usado em caso de empates). O valor de R_j para a j -ésima heurística é dado por: $R_j = \frac{1}{N} \cdot \sum_i r_i^j, \forall j \in \{1, \dots, k\}$, onde k é o

¹<https://www.r-project.org/>

²<https://cran.r-project.org/web/packages/scmamp/index.html>

número de heurísticas e N o número de instâncias. Os valores obtidos podem ser vistos na Tabela 4.1.

Tabela 4.1: *Ranks* médios das heurísticas em 1200 instâncias.

H1	H2	H3	H4	H5	H6	H7
4.86	6.54	3.18	3.15	3.08	3.78	3.41

A seguir, calculamos o valor do teste de Iman-Davenport seguindo as equações (4.7) e (4.8).

$$\chi_F^2 = \frac{12 \cdot N}{k \cdot (k+1)} \left[\sum_j R_j^2 - \frac{k \cdot (k+1)^2}{4} \right] \quad (4.7)$$

$$F_F = \frac{(N-1) \cdot \chi_F^2}{N \cdot (k-1) - \chi_F^2} \quad (4.8)$$

Utilizando os valores da Tabela 4.1 e as Equações (4.7) e (4.8), calculamos o seguinte valor para o teste:

$$\begin{aligned} \chi_F^2 &= \frac{12 \cdot 1200}{7 \cdot (7+1)} \cdot \left[(4.86^2 + 6.54^2 + 3.18^2 + 3.15^2 + 3.08^2 + 3.78^2 + 3.41^2) - \frac{7 \cdot (7+1)^2}{4} \right] \\ &= 2522.49 \\ F_F &= \frac{(1200-1) \cdot 2522.49}{1200 \cdot (7-1) - 2522.49} = 646.60 \end{aligned}$$

O valor do teste calculado para F_F segue a distribuição F , a qual tem como parâmetros os graus de liberdade, $k-1$ e $(k-1) \cdot (N-1)$. Logo, o valor crítico pode ser obtido seguindo tal distribuição. Para esse exemplo em específico, o valor crítico é 2.09. Como o valor encontrado pelo teste, F_F , é maior do que o crítico, podemos rejeitar a hipótese nula.

Usando o pacote **scmamp**, podemos calcular o teste de Iman-Davenport como apresentado no Código 4.4. O resultado do teste pode ser acessado através do atributo *statistic* do objeto retornado pela função que implementa o teste.

```
> res <- imanDavenportTest(df)
> res[['statistic']]
Corrected Friedman's chi-squared
646.60
```

Código 4.4: Cômputo do teste de Iman-Davenport usando o pacote **scmamp**

Em R, podemos calcular o valor crítico segundo a distribuição F com a função **qf**. Como pode ser visto no Código 4.5, passamos como parâmetros o nível de confiança (0.95) e os graus de liberdade $k-1$ (6) e $(k-1)(N-1)$ (7194).

```
> fdist <- qf(.95, k-1, (k-1)*(N-1))
> fdist
```


2.09

Código 4.5: Cômputo do teste de Iman-Davenport usando o pacote `scmamp`

Visto que a hipótese nula foi rejeitada, prosseguimos com um teste *post-hoc* para encontrar quais das heurísticas de fato diferem entre si. Usamos o teste de Nemenyi para computar a diferença crítica e identificar grupos de heurísticas que são estatisticamente equivalentes entre si. Duas heurísticas são ditas diferentes segundo esse teste se seus *rank*s médios diferem por pelo menos a diferença crítica (DC) obtida segundo a Equação (4.9).

$$DC = q_\alpha \cdot \sqrt{\frac{k \cdot (k + 1)}{6 \cdot N}} \quad (4.9)$$

Na Equação (4.9) α representa o nível de confiança, e o valor crítico q_α é definido segundo a estatística *Studentized range* dividida por $\sqrt{2}$. Os valores de tal estatística podem ser encontrados na Tabela 5(a) no artigo de Demšar [16].

Para esse exemplo em particular, temos $q_\alpha = 2.949$, aplicando a Equação (4.9), temos:

$$DC = 2.949 \cdot \sqrt{\frac{7 \cdot (7 + 1)}{6 \cdot 1200}} = 0.26$$

Usando o pacote `scmamp`, podemos calcular a DC segundo teste de Nemenyi como apresentado no Código 4.6. A DC pode ser acessada através do atributo *statistic* do objeto retornado pela função que implementa o teste.

```
> res <- nemenyiTest(df)
> res[['statistic']]
Critical difference
      0.26
```

Código 4.6: DC calculada pelo teste de Nemenyi usando o pacote `scmamp`.

O resultado do teste de Nemenyi pode ser representado graficamente, como ilustrado na Figura 4.3. O eixo horizontal corresponde aos valores de *rank* médio, em ordem crescente da esquerda para a direita. Cada heurística é representada por uma linha vertical com seu respectivo nome e valor de *rank* médio logo abaixo. Uma linha horizontal de comprimento equivalente ao valor da DC é posicionada acima do eixo horizontal principal. Heurísticas as quais o seu valor de *rank* médio não diferem mais do que a DC são conectadas por segmento horizontal em negrito, de comprimento sempre inferior ao valor da DC.

O mesmo gráfico pode ser obtido através da função `plotCD` do pacote `scmamp`, como é exibido no Código 4.7.

```
> plotCD(df)
```

Código 4.7: Função para exibição do gráfico de DC do teste de Nemenyi.

Observa-se que as heurísticas H5, H4 e H3 formam o grupo de menor valor de *rank* médio e conseqüentemente, são estatisticamente equivalentes entre si de acordo com o teste de Nemenyi.

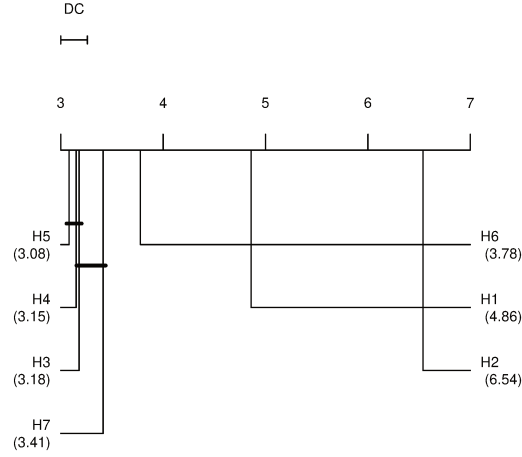


Figura 4.3: Exemplo de representação gráfica do teste de Nemenyi.

Podemos escolher uma dentre essas três heurísticas utilizando uma contagem do número de vezes em que cada heurística foi melhor do que as demais, i.e. o número de *vitórias* de cada heurística, como apresentado na Tabela 4.2.

Tabela 4.2: Contagem de vitórias entre as três heurísticas de menor *rank* médio.

Heurísticas	H5	H4	H3
H5	-	143	153
H4	116	-	143
H3	104	127	-

Pela contagem de vitórias, observamos que a heurística H5 teve desempenho superior ao das outras duas em uma quantidade maior de instâncias. Logo, podemos selecioná-la como melhor heurística para esse conjunto de instâncias.

Outro teste utilizado nesse trabalho foi o de Wilcoxon, o qual aplica-se quando queremos verificar se existe **DES** entre os resultados de dois métodos, sendo mais poderoso que o teste de Iman-Davenport para esse cenário. Basicamente, tal teste primeiro computa os *ranks* das diferenças entre os resultados dos dois métodos, ignorando o sinal e depois compara os *ranks* das diferenças positivas e negativas.

Para ilustrar o teste de Wilcoxon, vamos comparar as heurísticas com o menor (H5) e segundo menor (H4) *rank* médio. Seja $d_i, i \in \{1, \dots, N\}$ a diferença entre os resultados das duas heurísticas na i -ésima instância, r_{d_i} o *rank* do valor absoluto da i -ésima diferença (*rank* médio é atribuído no caso de empates). Seja R^+ a soma dos *ranks* para as instâncias nas quais a heurística H4 foi melhor do que a H5 ($d_i > 0$) e R^- a soma dos *ranks* para o caso oposto ($d_i < 0$). Os *ranks* para $d_i = 0$ são separados igualmente entre as somas, como mostrado na Equação (4.10).

$$R^+ = \sum_{d_i > 0} r_{d_i} + \frac{1}{2} \sum_{d_i = 0} r_{d_i} \qquad R^- = \sum_{d_i < 0} r_{d_i} + \frac{1}{2} \sum_{d_i = 0} r_{d_i} \quad (4.10)$$

Para o exemplo em questão, temos $R^+ = 346151$ e $R^- = 374449$. Seja $T = \min\{R^+,$

R^- }. Segundo Demšar [16], para $N \leq 25$ o valor crítico para o teste de Wilcoxon pode ser consultado na maioria dos livros de estatística, como, por exemplo, o de Devore [18]. Uma vez que tal valor é calculado, rejeitamos a hipótese nula se o valor de T é menor ou igual ao crítico. No caso de $N > 25$, computamos a estatística z , segundo a Equação (4.11).

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \quad (4.11)$$

Sendo $T = 346151$, aplicando a Equação (4.11) temos:

$$z = \frac{346151 - \frac{1}{4}1200(1200+1)}{\sqrt{\frac{1}{24}1200(1200+1)(2400+1)}} = -1.18$$

Segundo Demšar [16], com um nível de confiança de 95%, a hipótese nula pode ser rejeitada quando $z < -1.96$. Logo, para nosso exemplo, a hipótese não é rejeitada, então não existe DES entre as duas heurísticas. Nota-se que ambos testes de Wilcoxon e Nemenyi chegaram à mesma conclusão com respeito às duas heurísticas.

O teste de Wilcoxon também é implementado no pacote **scmamp**, através da função `wilcoxonSignedTest`, na qual passamos os resultados das duas heurísticas como parâmetros. O valor de T pode ser obtido através do atributo `statistic` do objeto retornado pelo teste, o qual é usado para o cálculo de z , como mostrado no Código 4.8.

```
> res <- wilcoxonSignedTest(df$H4, df$H5)
> T = res[['statistic']]
> # numerador
> num <- T - 0.25 * (N * (N + 1))
> # denominador
> den <- sqrt((N * (N + 1) * (2 * N + 1)) / 24)
> z <- num / den
> z
-1.18
```

Código 4.8: Cálculo do teste de Wilcoxon usando o pacote **scmamp**.

Outra abordagem de teste *post-hoc* é utilizando um *método de controle*. Como é argumentado por Demšar [16] esse tipo de teste geralmente tem mais poder que o teste de Nemenyi, devido ao número reduzido de comparações, $k - 1$, enquanto o de Nemenyi faz com todos os $k \cdot (k - 1)/2$ possíveis pares de métodos. Não entraremos em detalhes sobre essa abordagem nesse documento, uma vez que os resultados não foram diferentes dos de Nemenyi, considerando as amostras de resultados que obtivemos. Alguns exemplos desses testes, incluem o de Bonferroni-Dunn, Holm e Hommel. Mais detalhes de seu funcionamento podem ser encontrados no trabalho de Demšar [16].

4.4 Geração de Instâncias

Os autores Blum et al. [7] e García-Martínez et al. [27] deixaram disponíveis conjuntos de instâncias para serem utilizadas em futuros estudos computacionais do FFP. De maneira a complementar esse *benchmark*, geramos uma série de instâncias com parâmetros e modelos diferentes daqueles anteriormente empregados na literatura. Nessa seção, descrevemos como tais instâncias foram criadas.

Começamos pelas instâncias concebidas segundo o modelo de Erdős e Rényi [22], também conhecido como $G(n, m)$. Nesse modelo, m arestas são selecionadas aleatoriamente com probabilidade uniforme do conjunto de todas as arestas possíveis, sem reposição. No decorrer desse documento, nos referimos ao conjunto dessas instâncias como GER-ER.

Para garantir que o grafo resultante seja sempre conexo, antes de iniciar o algoritmo do modelo de Erdős e Rényi [22], construímos uma árvore geradora [13] na qual as arestas são adicionadas aleatoriamente. Como se vê no Algoritmo 13, a árvore é construída através de uma adaptação do algoritmo de Kruskal para o problema da árvore geradora mínima [13] de forma que, ao invés de escolher a aresta de menor peso, a cada iteração, na Linha 11, escolhemos aleatoriamente uma no conjunto de todas as arestas restantes, desde que, claro, não sejam formados ciclos com arestas selecionadas previamente.

Algoritmo 13: Algoritmo para construção de uma árvore geradora aleatória.

```

Entrada:  $n$ 
Saída: Árvore geradora aleatória  $T$ .
1  $T.V \leftarrow \{1, \dots, n\}$ 
2  $T.E \leftarrow \emptyset$ 
3  $A \leftarrow \emptyset$  ▷ Conjunto de todas as arestas possíveis.
4 para  $u = 1$  até  $n$  faça
5   | para  $v = u + 1$  até  $n$  faça
6   | |  $A \leftarrow A \cup \{(u, v)\}$ 
7  $m \leftarrow |A|$ 
   | ▷ Construção dos conjuntos disjuntos.
8 para  $v = 1$  até  $n$  faça
9   | Make-Set( $v$ )
10 para  $i = 1$  até  $m$  faça
11   |  $(u, v) \leftarrow \text{Aresta-Aleatória}(A)$ 
12   |  $A \leftarrow A \setminus \{(u, v)\}$ 
13   | se  $\text{Find-Set}(u) \neq \text{Find-Set}(v)$  então
14   | | Union( $u, v$ )
15   | |  $T.E \leftarrow T.E \cup \{(u, v)\}$ 
16 retorna  $T$ 

```

As operações de conjuntos disjuntos **Make-Set**, **Find-Set** e **Union** foram implementadas como é sugerido por Cormen et al. [13], incluindo as heurísticas de *união por rank* e *compressão de caminhos*.

O outro modelo utilizado foi o de Barabási e Albert [4], o qual produz grafos que tendem a simular relações em redes sociais. Nesses grafos, alguns poucos vértices tem

grau muito alto (são mais influentes), enquanto os demais tem um grau relativamente pequeno.

O algoritmo desse modelo funciona da seguinte forma. Um grafo G é inicializado com uma quantidade $m_0 < n$ de vértices, sem nenhuma aresta entre si. Então, a cada iteração, um novo vértice u é adicionado e $m \leq m_0$ arestas são adicionadas entre u e outros m vértices já existentes no grafo. A probabilidade $\Pi(g_v)$ de se adicionar uma aresta entre u e um outro vértice v já existente em G é dada pela Equação (4.12).

$$\Pi(g_v) = \frac{g_v}{\sum_{v' \in V_G} g_{v'}} \quad (4.12)$$

Dessa forma, os novos vértices acrescentados são conectados a vértices de maior grau com maior probabilidade. Considerando esse modelo em particular, utilizamos a implementação da biblioteca **NetworkX** [34]. Ao decorrer desse documento, nos referimos ao conjunto de instâncias gerados por esse modelo como **GER-BA**.

Em ambos conjuntos de instâncias, **GER-ER** e **GER-BA**, o foco de incêndio é um vértice de maior grau.

Capítulo 5

Resultados Computacionais

Nesse capítulo, descrevemos os experimentos que foram realizados visando avaliar a eficácia: das modificações do modelo PLI propostas na Seção 4.1, das heurísticas gulosas (Seção 4.2.2), da matheurística MATHEU (Seção 4.2) e de sua variação MATHEU-DI (Seção 4.2.6). Inicialmente, apresentamos as instâncias e o ambiente computacional no qual os experimentos foram executados, junto com os resultados que levaram à decisão de atribuir valores específicos de parâmetros da MATHEU.

Na análise das heurísticas gulosas e da MATHEU, consideramos como medida de desempenho o número de vértices salvos quando o fogo é contido. Nesse caso, comparamos nossos resultados com aqueles publicados em Blum et al. [7], Hu, Windbichler e Raidl [39] e García-Martínez et al. [27]. Com relação ao desempenho do M-FFMO, usamos como medida o tempo de computação, comparando seus resultados com os do FFMO.

5.1 Ambiente Computacional e Instâncias

As estratégias apresentadas no Capítulo 4 foram implementados em C++ e os programas compilados com o compilador GCC versão 7.1.0, usando o nível de otimização -O3. De maneira a implementar o FFMO e suas variações, o resolvidor IBM ILOG CPLEX versão 12.7.1 foi utilizado, mantendo a configuração padrão de seus parâmetros exceto pelo modo de paralelismo, que foi definido como *paralelo determinístico* [14]. Os experimentos foram realizados em um computador com um processador Intel®Xeon®CPU E5-2420 com 12 núcleos de 1.90 GHz cada e 32GB de memória RAM. Quando executamos múltiplas heurísticas gulosas, cada uma delas foi executada por uma *thread* separada, usando a estratégia de *tasks* da biblioteca de paralelização OpenMP [3].

Os conjuntos de grafos utilizados em nossos experimentos foram providos por Blum et al. [7] (denotado por BBGRL) e García-Martínez et al. [27] (denotado por GBRL), ambos discutidos no Capítulo 3. O primeiro conjunto foi usado nas comparações com os resultados de Blum et al. [7] e Hu, Windbichler e Raidl [39], enquanto o segundo foi empregado nas comparações com os resultados de García-Martínez et al. [27].

Como mencionado no Capítulo 3, denotamos a combinação de um valor de n e o parâmetro do respectivo modelo de geração de grafos (p ou r) como uma *classe do problema*. Considerando todos os conjuntos, tem-se 360 grafos no total, sendo $120 =$

$4 \times 3 \times 10$ ($\#n \times \#$ valores de $p \times \#$ grafos por classe de problema) do conjunto BBGRL e $240 = 4 \times 6 \times 10$ ($\#n \times \#$ valores de p e $r \times \#$ grafos por classe de problema) do conjunto G-GBRL. Para cada grafo, os valores de D , o número de brigadistas disponíveis, utilizados foram $D \in \{1, 2, \dots, 10\}$, levando a um total de 3600 instâncias. Adotando convenção de trabalhos experimentais presentes na literatura, o conjunto B dos focos de incêndio é formado pelo vértice de índice um em todas as instâncias.

Ainda seguindo a metodologia adotada nos trabalhos anteriores, definimos um tempo limite de $n/2$ segundos para a execução da MATHEU, onde n é o número de vértices do grafo. De acordo com *benchmarks* padrões para análise de desempenho de processadores fornecidos pela SPEC (Standard Performance Evaluation Corporation¹), a máquina na qual executamos nossos experimentos é 1.5 vezes mais rápida que a de Hu, Windbichler e Raidl [39] e, uma vez que os autores já haviam decrescido o tempo de computação para $n/4$ para lidar com diferenças de *hardware* quando compararam com os resultados de Blum et al. [7], definimos nosso tempo limite como $n/6$ para o mesmo propósito. Com respeito ao trabalho de García-Martínez et al. [27], nossa máquina é 1.4 vezes mais rápida, então, seguindo a mesma lógica, definimos um tempo limite de $n/2.8$ ao usar seus resultados como base de comparação.

Ao analisar a matheurística, de forma a comparar nossos resultados com os da literatura, seguimos a mesma metodologia dos trabalhos anteriores, apresentando os resultados sumarizados como a média aritmética dos valores de função objetivo por classe do problema e por valor de D .

Quando comparamos os resultados de duas ou mais estratégias em múltiplas instâncias, usamos uma análise com uma série de testes estatísticos não paramétricos, seguindo a metodologia de Demšar [16], como discutido na Seção 4.3. Nossa hipótese nula quando comparando duas ou mais estratégias é de que não existe DES entre as mesmas. Todos os testes foram feitos com um nível de confiança de 95%.

5.2 Melhorias no Modelo FFM0

Na Seção 4.2.3, mostramos que a busca local da MATHEU é feita através da resolução de um modelo PLI. Para tornar tal estratégia viável, o modelo correspondente a uma instância restrita do FFP tem que ser resolvido rapidamente. Com esta finalidade, propomos uma formulação modificada, o M-FFM0, na qual técnicas de agregação de restrições e preprocessamento foram aplicadas no FFM0. Essa seção é dedicada à discussão dos resultados em termos do *speedup* obtido pelo M-FFM0 em relação ao FFM0, os quais suportam nossa decisão de utilizar o M-FFM0.

O *speedup* é calculado pela fração $\frac{\text{Tempo de computação do FFM0}}{\text{Tempo de computação do M-FFM0}}$. Definimos $T = \lceil \frac{n}{D} \rceil$ de modo a garantir a obtenção de soluções ótimas. As instâncias utilizadas para realizar esse experimento em específico foram aquelas para as quais a solução ótima pôde ser encontrada em no máximo uma hora pelo FFM0. Considerando esta restrição, os testes limitaram-se às instâncias com $n \in \{50, 100\}$ dos conjuntos BBGRL e GBRL, totalizando $1800 = 2 \times 9 \times 10 \times 10$ ($\#n \times \#$ valores de p e $r \times \#$ grafos por classe de problema \times

¹www.spec.org/cpu2006

#brigadistas) instâncias com $D \in \{1, 2, \dots, 10\}$.

Um *Speedup* de pelo menos 1.1 foi obtido em 1306 instâncias do total de 1800. A Tabela 5.1 apresenta estatísticas para essas instâncias. A primeira coluna indica a qual conjunto de instâncias os resultados se referem e a segunda coluna refere-se ao número de vértices correspondente. As colunas restantes são estatísticas dos valores de *speedup*.

Tabela 5.1: Estatísticas das 1306 instâncias com *speedup* de pelo menos 1.1.

conjunto	n	mediana	média	desvio padrão	max
BBGRL	50	1.73	1.90	0.78	5.42
BBGRL	100	1.61	2.05	1.11	5.19
G-GBRL	50	1.77	1.81	0.50	4.27
G-GBRL	100	1.79	2.02	1.15	14.45
A-GBRL	50	1.68	1.89	0.89	9.78
A-GBRL	100	1.58	1.74	0.64	5.20

Os resultados mostram um *speedup* de aproximadamente 2 com uma mediana em torno de 1.7, i.e., 70% mais rápido do que o FFMO. *Speedups* de pelo menos 2 foram obtidos em 363 instâncias, cerca de 20% do total.

A Tabela 5.2 apresenta os resultados para as 144 instâncias nas quais o FFMO consumiu pelo menos cinco minutos de computação para obter uma solução ótima. Foco foi dado a essas instâncias de maneira a evitar tirar conclusões que levam em consideração tempos de computação pequenos, os quais são mais sensíveis a erros de medição.

Tabela 5.2: Estatísticas de *speedup* das instâncias nas quais o FFMO consumiu pelo menos 5 minutos de computação.

conjunto	n	min	mediana	média	desvio padrão	max
BBGRL	100	1.12	1.47	1.57	0.43	2.97
G-GBRL	100	1.62	2.73	3.73	3.22	14.45
A-GBRL	100	1.11	1.52	1.65	0.58	5.20

Como pode ser visto, os resultados na Tabela 5.2 são similares aos da Tabela 5.1, com a exceção de que o *speedup* para as instâncias do conjunto G-GBRL com $n = 100$ apresenta valores de média e mediana maiores. Para todas as instâncias, um *speedup* de pelo menos 1.1 foi atingido. Um caso extremo com mais de 14 de *speedup* foi obtido no conjunto de instâncias G-GBRL com $n = 100$.

Com tais resultados, chegamos à conclusão de que nossas modificações no FFMO aceleram a computação de soluções ótimas. Tal melhoria é uma contribuição desta dissertação para a busca por algoritmos exatos para o FFP. Para os algoritmos heurísticos propostos neste trabalho, a importância deste fato se torna ainda mais evidente, uma vez que em um passo da MATHEU, precisamos resolver uma versão mais restrita do M-FFMO várias vezes.

5.3 Heurísticas Gulosas

A primeira fase da **MATHEU** consiste em obter um conjunto S de soluções a partir de várias execuções de alguma heurística em modo probabilístico (como discutido na Seção 4.2.2) controlado por um valor do parâmetro α , o qual determina o tamanho da LRC (como discutido na Seção 2.4). Os experimentos descritos nesta seção visam identificar a heurística a ser utilizada nessa fase, bem como o respectivo valor de α . As heurísticas consideradas para comparação foram as propostas na Seção 4.2.2 junto das que foram apresentadas no trabalho de García-Martínez et al. [27] e tidas como as melhores pelos autores, sendo essas as heurísticas gulosas **ThDegree** e **ThDesc**, ambas descritas em detalhes no Capítulo 3. Implementamos ambas heurísticas de García-Martínez et al. [27] e os experimentos foram conduzidos com nossa implementação, uma vez que desejávamos obter os resultados individuais por instância para fins de comparação e os mesmos não foram disponibilizados pelos autores. Para maior clareza, na análise dos resultados utilizamos o prefixo **GBRL** para nos referirmos a essas heurísticas.

Os valores de α considerados nos experimentos foram $\{0, 0.1, 0.2, \dots, 0.8, 0.9\}$ e as instâncias foram aquelas do conjunto **GBRL** (ambos **A-GBRL** e **G-GBRL**) com $n \in \{50, 100, 500, 1000\}$. O tempo limite utilizado foi de $n/2$ segundos para cada execução, exceto para o caso de $\alpha = 0$, uma vez que este corresponde à execução em modo determinístico. Os valores testados para a quantidade de brigadistas disponíveis foram $D \in \{1, \dots, 5\}$, totalizando $1200 = 4 \times 6 \times 10 \times 5$ ($\#n \times \#$ valores de p e $r \times \#$ grafos por classe de problema $\times \#$ brigadistas) execuções por valor de α . A semente do gerador de números pseudo aleatórios foi uma diferente para cada execução.

Primeiramente, investigamos o desempenho de diferentes heurísticas para todos os dez valores de α . Para cada valor de α , utilizamos o teste de Iman-Davenport com todas as heurísticas e quando uma **DES** é identificada por esse teste, procedemos com o teste de Nemenyi para procurar por grupos de heurísticas equivalentes entre si.

Os resultados dos testes são apresentados na Tabela 5.3, na qual cada linha corresponde aos resultados do valor de α exibido na primeira coluna. A segunda coluna é o valor de teste de Iman-Davenport, seguido pelo respectivo **Valor Crítico** de acordo com a distribuição F e a quarta coluna é o valor da **Diferença Crítica** retornado pelo teste de Nemenyi. As últimas colunas indicam se a respectiva heurística está presente no grupo com menor valor de *rank* médio, ou seja, entre as heurísticas estatisticamente equivalentes entre si que possuem os menores valores de *rank* médio. Quando a heurística está presente, o símbolo \times é adicionado na respectiva coluna.

Observa-se que para todos os valores de α a hipótese nula foi rejeitada, uma vez que o valor computado pelo teste de Iman-Davenport é maior do que o crítico. As heurísticas **Th**, **ThDynDegree**, **GBRL-ThDegree** e **GBRL-ThDesc** foram as que estiveram presentes pelo menos uma vez no grupo com menor valor de *rank* médio. Assim, nos testes seguintes consideramos apenas esse subgrupo.

A Figura 5.1 apresenta uma outra perspectiva sobre os resultados das heurísticas, através de seus *ranks* cumulativos, sendo cada subfigura correspondente a um valor de α , indicado logo acima. O eixo x corresponde aos valores de *rank* cumulativo, enquanto o eixo y indica ao número de instâncias (de um total de 1200) que a heurística correspondente

Tabela 5.3: Resultados dos testes estatísticos por valor de α .

α	ID	VC	DC	Th	GBRL-ThDesc	GBRL-ThDegree	ThDynDegree
0.0	974.73	002.09	000.26	×	×	×	×
0.1	646.59	002.09	000.26	×	×	×	
0.2	516.40	002.10	000.26	×	×	×	
0.3	423.29	002.10	000.26	×	×	×	
0.4	354.34	002.10	000.26	×	×	×	×
0.5	392.90	002.10	000.26	×	×	×	
0.6	380.97	002.10	000.26	×	×	×	×
0.7	385.36	002.10	000.26	×	×	×	
0.8	403.31	002.10	000.26	×	×	×	×
0.9	428.10	002.10	000.26	×	×	×	

obteve aquele valor de *rank*. Dessa forma, quanto mais instâncias com valores de *rank* menores, melhor é a heurística. Nesse estudo, atribuímos o *rank* mínimo no caso de empates. Isso significa que, por exemplo, se um subconjunto de heurísticas tem o mesmo valor de função objetivo e este é o melhor valor, todas recebem *rank* um.

Analisando o gráfico, com respeito à outra heurística que desenvolvemos, **ThDescBFS**, a qual não esteve no grupo de menores *rank*s, observa-se que a mesma ficou em quinta posição considerando os valores de *rank* cumulativo, para todos os valores de α . Nota-se também que a combinação dos nossos critérios gulosos com a heurística **Th**, as heurísticas **ThDescBFS** e **ThDynDegree**, produziram melhores resultados do que as suas versões originais, **DescBFS** e **DynDegree**, respectivamente, para todos os valores de α .

Em seguida, procuramos identificar um subgrupo de valores de α com melhores resultados, considerando as quatro heurísticas da Tabela 5.3. De maneira a fazer isso, seguimos com a mesma metodologia, utilizando os testes estatísticos. Os resultados dos testes são exibidos na Tabela 5.4, onde cada linha corresponde aos resultados de uma heurística. As demais colunas apresentam os mesmos dados da Tabela 5.3, exceto que a última coluna mostra somente o valor de α com **Menor** valor de *Rank* médio.

Tabela 5.4: Resultados dos testes estatísticos das heurísticas para diferentes valores de α .

Heurística	ID	VC	DC	MR
Th	0.25	1.88	0.39	0.3
ThDesc	0.13	1.88	0.39	0.5
ThDegree	0.15	1.88	0.39	0.5
ThDynDegree	0.39	1.88	0.39	0.0

Observamos que a hipótese nula não foi rejeitada para nenhum valor de α , visto que os resultados do teste de Iman-Davenport sempre foram menores que os respectivos valores críticos. Decidimos, então, prosseguir nos testes com os valores $\alpha \in \{0.0, 0.3, 0.5\}$, que tiveram os menores valores de *rank* médio.

Para cada um destes três valores escolhidos de α , procuramos encontrar qual das quatro heurísticas selecionadas tinha o melhor desempenho. O resultado do teste de

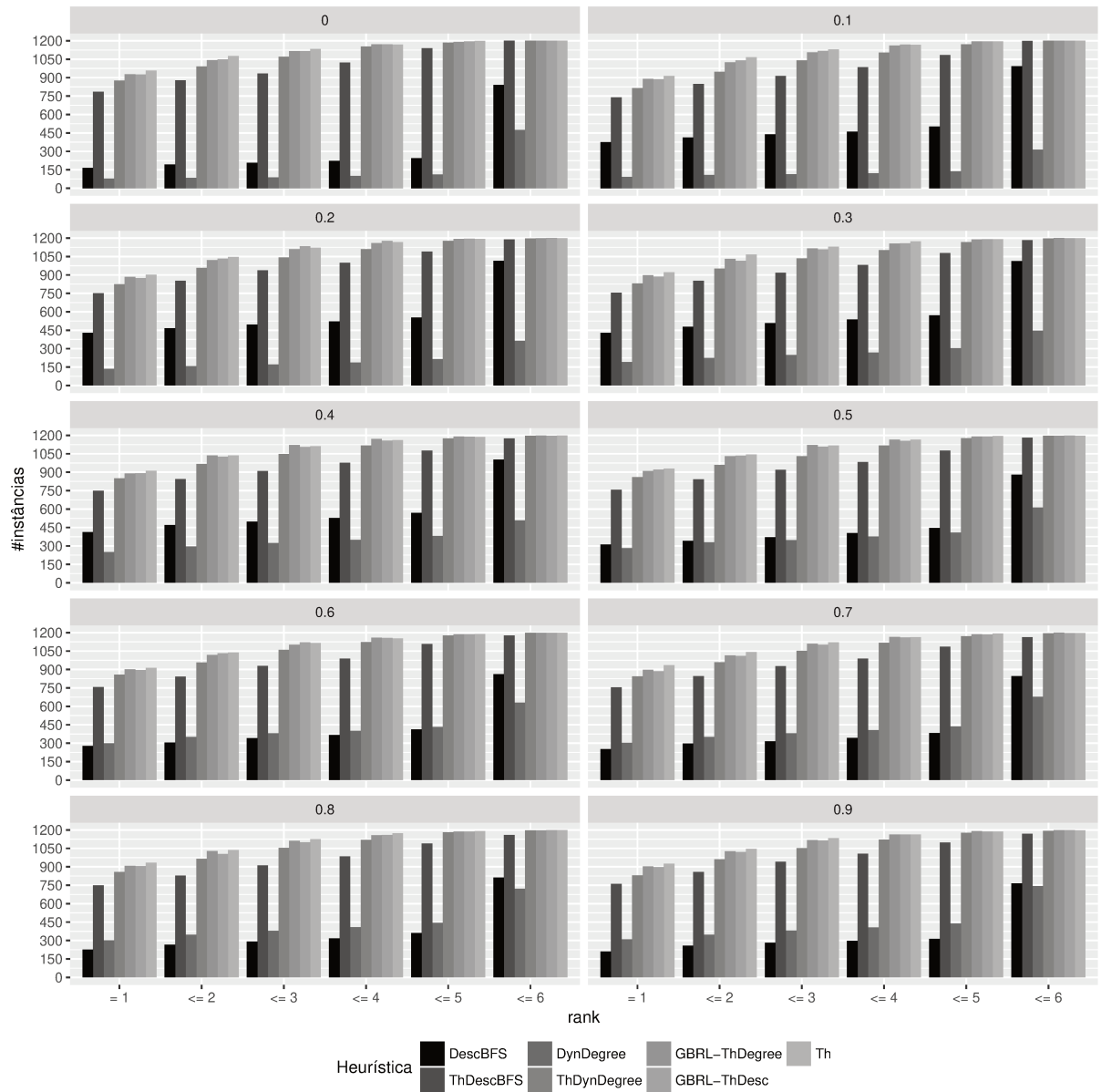


Figura 5.1: *Ranks* cumulativos das heurísticas, considerando todos os valores de α .

Nemenyi pode ser visto graficamente na Figura 5.2. Lembrando que o formato desse gráfico é discutido na Seção 4.3; A legenda de cada subfigura mostra os valores do teste de Iman-Davenport (ID), o respectivo valor crítico segundo a distribuição F (VC) e a diferença crítica computada pelo teste de Nemenyi (DC).

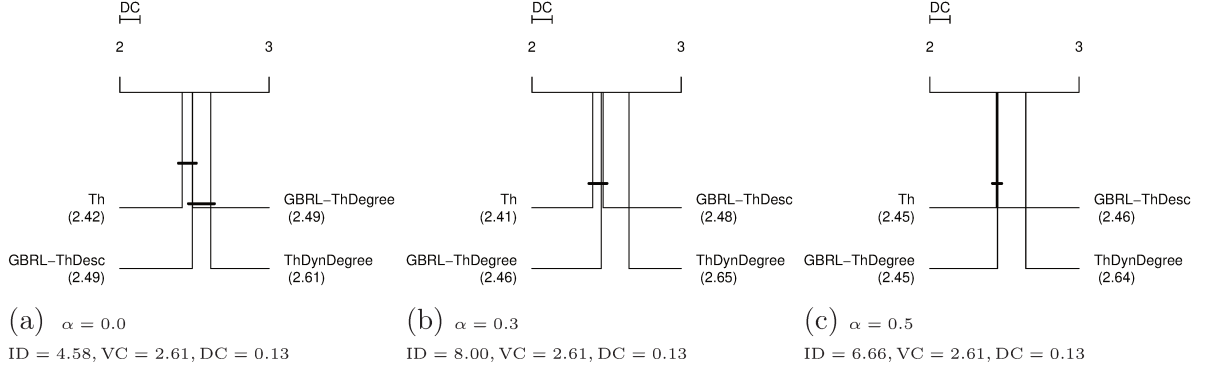


Figura 5.2: Testes de Iman-Davenport e Nemenyi entre as heurísticas para cada valor de α .

Para todos os valores de α , o teste de Iman-Davenport encontrou DES entre as heurísticas, uma vez que o valor do teste é maior do que o respectivo valor crítico. Observamos que, para todos os três valores de α , a heurística **Th** foi a com o menor valor de *rank* médio e também foi estatisticamente equivalente a **GBRL-ThDegree** e **GBRL-ThDesc**.

Baseados nos resultados do teste de Nemenyi da Figura 5.2, selecionamos as heurísticas **Th**, **GBRL-ThDegree** e **GBRL-ThDesc** como candidatas a melhor. De maneira a escolher uma dentre as três, analisamos os valores do *rank* cumulativo das mesmas, como apresentado na Figura 5.3. Cada subfigura na Figura 5.3 corresponde aos *ranks* cumulativos de cada valor de α , indicado logo acima. A heurística **Th** foi superior às outras duas para $\alpha = 0.0$ e $\alpha = 0.3$. Considerando $\alpha = 0.5$ a heurística **Th** ficou abaixo da **GBRL-ThDesc** para o valor de *rank* = 1, ficando acima para *ranks* ≤ 2 .

Os resultados para $\alpha = 0.5$ impediram que fosse caracterizada em definitivo a supremacia da heurística **Th** sobre as concorrentes. Optamos, assim, por considerar o número de vitórias e derrotas entre as duas heurísticas, **Th** e **GBRL-ThDesc**, como critério para escolher uma dentre as duas. É considerada vitória quando uma heurística é melhor do que a outra em uma instância e uma derrota caso contrário. Dito isso, a heurística **Th** obteve 134 vitórias e 125 derrotas, com empates observados no restante das 1200 instâncias. Note que o uso dos testes estatísticos nos levou à conclusão que as heurísticas **Th**, **GBRL-ThDesc** e **GBRL-ThDegree** são estatisticamente equivalentes para valores de $\alpha \in \{0.0, 0.3, 0.5\}$, assim, optamos pela análise por *ranks* cumulativos e número de vitórias para escolher uma dentre as três heurísticas. Ou seja, as duas abordagens têm conclusões similares, mas a última nos diz qual, dentre as duas estratégias, obteve melhores resultados.

Vistos tais resultados, selecionamos a heurística **Th** para ser utilizada na primeira fase da **MATHEU**. Uma vez que o teste de Iman-Davenport não encontrou DES entre os valores de α , selecionamos o valor de menor *rank* médio com relação à heurística **Th**, ou seja $\alpha = 0.3$.

Em 600 das 1200 das instâncias, o valor ótimo era conhecido (para $n \in \{50, 100\}$). Isso permitiu observar que em 369 destas 600 instâncias a heurística **Th** com $\alpha = 0.3$ obteve

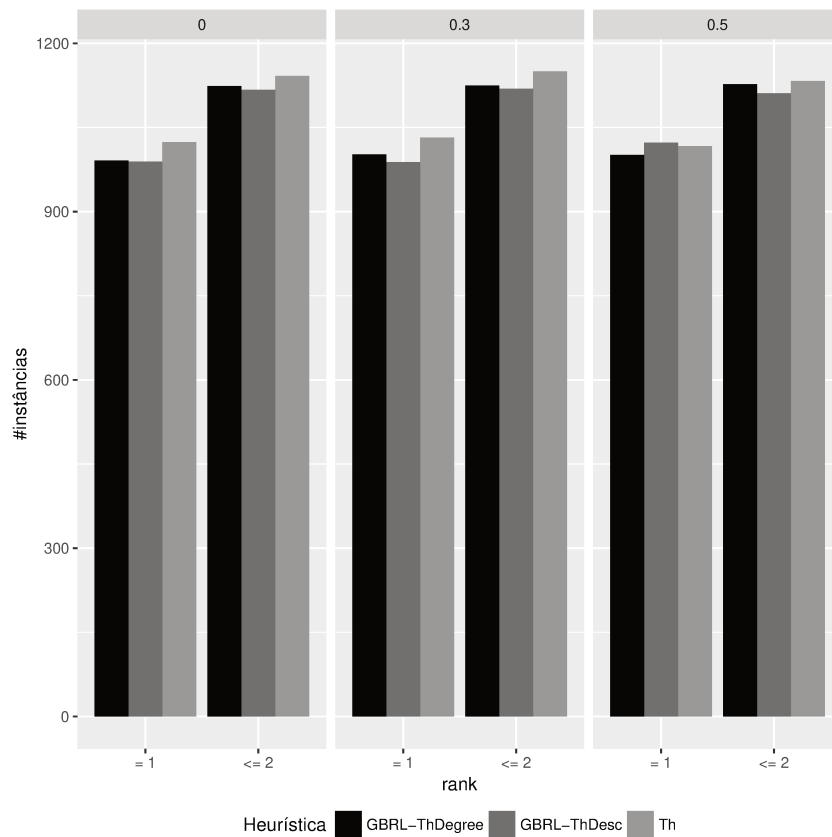


Figura 5.3: *Ranks* cumulativos das heurísticas, para cada valor de α .

o valor ótimo (cerca de 61% do total), sendo 195 vezes no conjunto G-GBRL e 174 no conjunto A-GBRL. Nas instâncias nas quais o ótimo não foi obtido, a Tabela 5.5 apresenta o *gap*, em porcentagem, entre o valor da função objetivo das soluções encontradas pela heurística e o respectivo valor ótimo.

Tabela 5.5: *Gap* (%) entre a solução encontrada pela heurística Th com $\alpha = 0.3$ e o valor ótimo, para as instâncias com $n \in \{50, 100\}$.

conjunto	n	min	média	mediana	desvio padrão	max
G-GBRL	50	2.63	18.51	16.67	15.15	62.07
G-GBRL	100	1.20	29.73	21.53	23.05	82.81
A-GBRL	50	2.94	07.63	07.14	03.09	20.00
A-GBRL	100	3.85	09.93	09.09	03.88	20.00

Observamos que o *gap* médio é maior para as instâncias do conjunto G-GBRL. Para as instâncias do conjunto A-GBRL, o *gap* é relativamente menor, com um valor máximo de 20%. Tais dados empíricos mostram que a heurística Th apresenta resultados inferiores para as instâncias do conjunto G-GBRL quando comparado ao conjunto A-GBRL. Acreditamos que esse desempenho inferior decorre do fato de que esses grafos possuem uma estrutura agrupada, na qual grupos de vértices são conectados por poucas arestas entre si. Com essa estrutura, uma estratégia que intuitivamente levaria a uma boa solução seria defender os vértices nos quais essas arestas são incidentes. Uma vez que defendemos vér-

tices selecionados aleatoriamente dentre os que estão ameaçados, algumas dessas escolhas podem não incluir esses casos particulares.

Outra informação investigada por esse experimento foi o número de vezes necessárias de se executar a heurística **Th** em modo probabilístico, denotado por η . Como foi observado empiricamente que uma única execução da heurística consome uma porção pequena do tempo limite dado, notamos que é feito um grande número de execuções e que o número de execuções finais sem melhoria na solução obtida corresponde a cerca de 89% do total. A partir daí, procuramos por um valor para η que reduzisse o número de execuções finais sem melhoria. Isso é importante para o desempenho da matheurística, uma vez que um descrésimo no tempo gasto obtendo o conjunto de soluções distintas S permite que mais tempo seja investido no procedimento de busca local.

Usando a heurística **Th** e $\alpha = 0.3$, observamos que em aproximadamente 80% das instâncias utilizadas nesse experimento, o valor de $\eta = 11000$ foi maior ou igual do que o valor da última iteração na qual ocorreu uma melhoria. Assim, para garantir que a primeira fase da **MATHEU** não consuma inutilmente uma parte expressiva do tempo limite, utilizamos como critérios de parada para rodar a heurística em modo probabilístico o número máximo de iterações em $\eta = 11000$ e o tempo de computação não superior à metade do tempo limite total, o que ocorrer primeiro.

Como apresentado na Seção 4.2.2, no modo probabilístico de execução de uma heurística, a primeira iteração consiste na execução do modo determinístico e as demais seguem o critério guloso aleatorizado. Nesse mesmo experimento, foi observada a eficácia do modo probabilístico, visto que em cerca de 82% das instâncias a solução gerada pela execução determinística foi melhorada por alguma das execuções subsequentes da execução probabilística.

5.4 Matheurística

Começamos essa seção apresentando os parâmetros da **MATHEU** e seus respectivos valores, junto da justificativa da escolha desses valores. A Tabela 5.6 mostra um sumário de tais parâmetros.

Tabela 5.6: Parâmetros da matheurística.

T	ϵ	α	k	σ_0	η	ρ	f
$(1 + \epsilon) \cdot T_{s_{Th}}$	0.5	0.3	2	0.5	11000	4	<i>f</i> -Desc (Utilizando o Algoritmo 8)

Parâmetro T e ϵ . Como dito anteriormente, o valor do parâmetro tem um forte impacto nos modelos PLI. Definindo $T = \lceil \frac{n}{D} \rceil$, garante-se que o ótimo é encontrado, mas sua resolução é custosa. De forma a alcançar um compromisso entre tempo de computação e qualidade de solução, o valor de T deve ser grande o suficiente para permitir que uma solução ótima seja encontrada e, ao mesmo tempo, pequeno o bastante para possibilitar que o modelo PLI seja resolvido rapidamente. Para lidar com essa situação, na **MATHEU** esse parâmetro é dado pela formula $(1 + \epsilon) \cdot T_{s_0}$ como descrito na Seção 4.2.1.

Para encontrar um valor apropriado para ϵ , realizamos experimentos usando o M-FFMO com a técnica de *warm start*. Nesse caso, a solução inicial passada para o resolvidor PLI é a melhor entre as geradas pelas heurísticas descritas na Seção 4.2.2 e as de García-Martínez et al. [27], todas executadas em modo determinístico. Além disso, na expressão $(1 + \epsilon) \cdot T_{s^0}$, T_{s^0} foi definido como o número de rodadas necessárias para conter o fogo nessa melhor solução. As opções de ϵ exploradas foram $\{0.2, 0.3, 0.4, 0.5, 0.6\}$. Os testes foram realizados nos grafos dos conjuntos GBRL e BBGRL com $n \in \{50, 100\}$ e o ótimo conhecido. No total, foram selecionadas 1800 instâncias com $D \in \{1, 2, \dots, 10\}$.

Nosso objetivo com esse teste era encontrar um valor de ϵ que levasse a boas soluções em pouco tempo. Para mensurar a eficácia de um valor escolhido, contamos o número de vezes que o ótimo foi alcançado. O tempo de execução do modelo PLI com $T = \lceil \frac{n}{D} \rceil$, nomeado como OPT-M-FFMO, foi usado como base para avaliar a eficiência.

Os resultados indicaram que o modelo com $\epsilon = 0.5$, denotado por T1.5-M-FFMO, provê um bom balanço entre qualidade de solução e *speedup*. Valores menores de ϵ levaram a modelos mais rápidos ao custo de perder em qualidade de solução. Por outro lado, valores maiores não aprimoraram significativamente a qualidade da solução e demandaram tempos de computação consideravelmente altos. No total, o ótimo foi encontrada em todas exceto uma das 1800 instâncias. Em adição, notamos que em 57% dos casos, a heurística Th foi responsável pela melhor solução inicial.

Em termos de *speedup*, concentramos nossa análise nas 155 instâncias nas quais o modelo OPT-M-FFMO consumiu pelo menos cinco minutos de computação, de forma a evitar conclusões distorcidas por tempos de execução pequenos. Um sumário das estatísticas coletadas para essas instâncias pode ser visto na Tabela 5.7. Nós observamos um *speedup* de 31.18 em média com um desvio padrão de 47.17 e uma mediana de 7.23. O valor máximo alcançado foi 249.5 e, em todos os casos, um *speedup* de pelo menos 1.1 foi obtido.

Tabela 5.7: Estatísticas de *speedup* do T1.5-M-FFMO em relação ao OPT-M-FFMO.

n	min	mediana	média	desvio padrão	max
100	1.10	7.23	31.18	47.17	249.50

A partir destes testes, decidimos definir $\epsilon = 0.5$ e $T_{s^0} = T_{s_{\text{Th}}}$ na continuação de nossos experimentos com a matheurística.

Parâmetro k . O valor do parâmetro k , usado na busca local para indicar o comprimento máximo do caminho necessário para considerar se dois vértices são vizinhos, foi escolhido baseando-se no conceito de distância entre soluções introduzido no Capítulo 2. Para um grupo seletivo de instâncias, investigamos a distância entre a solução obtida pela heurística Th (s_{Th}) executada em modo determinístico e uma solução ótima (s^*). Acredita-se que o valor máximo de distância entre os conjuntos de vértices defendidos pelas duas soluções, denotados por $\mathcal{D}_{s_{\text{Th}}}$ e \mathcal{D}_{s^*} , respectivamente, medida nos testes é um bom indicativo de em qual k -vizinhança de \mathcal{D}_{s^*} os vértices de $\mathcal{D}_{s_{\text{Th}}}$ estão em geral. Uma vez que estávamos interessados em obter soluções o mais próximas possível da ótima, utilizamos esse valor

de distância máxima como uma estimativa para k .

Realizamos esse experimento nas instâncias do conjunto BBGRL com $n = 100$ e $D \in \{1, \dots, 5\}$, desconsideradas as que o respectivo conjunto $N(B)$ é menor ou igual do que o número de brigadistas disponíveis. Isso porque, nessas instâncias, chamadas de *triviais*, a solução ótima pode ser obtida simplesmente defendendo todos os vértices no conjunto $N(B)$ na primeira rodada, contendo-se, assim, o fogo. No total, foram utilizadas 90 instâncias.

Os resultados desses experimentos são apresentados na Figura 5.4, onde o eixo x corresponde aos valores de distância e o eixo y ao número de instâncias onde aquela distância foi verificada. Observamos que o valor da distância máxima é três, o que significa que um vértice defendido em uma solução s_{Th} está no máximo na 3-vizinhança do vértice mais próximo no conjunto \mathcal{D}_{s^*} . Outra observação foi que o número de instâncias com distância igual a três foi significativamente menor do que para as demais distâncias. Visto isso, inferimos que a probabilidade de que um vértice defendido em s^* esteja a uma distância maior ou igual a três de um vértice defendido em s_{Th} é pequena. Então, definimos $k = 2$ de maneira a construir vizinhanças que provavelmente incluiriam todos os vértices defendidos em uma solução ótima.

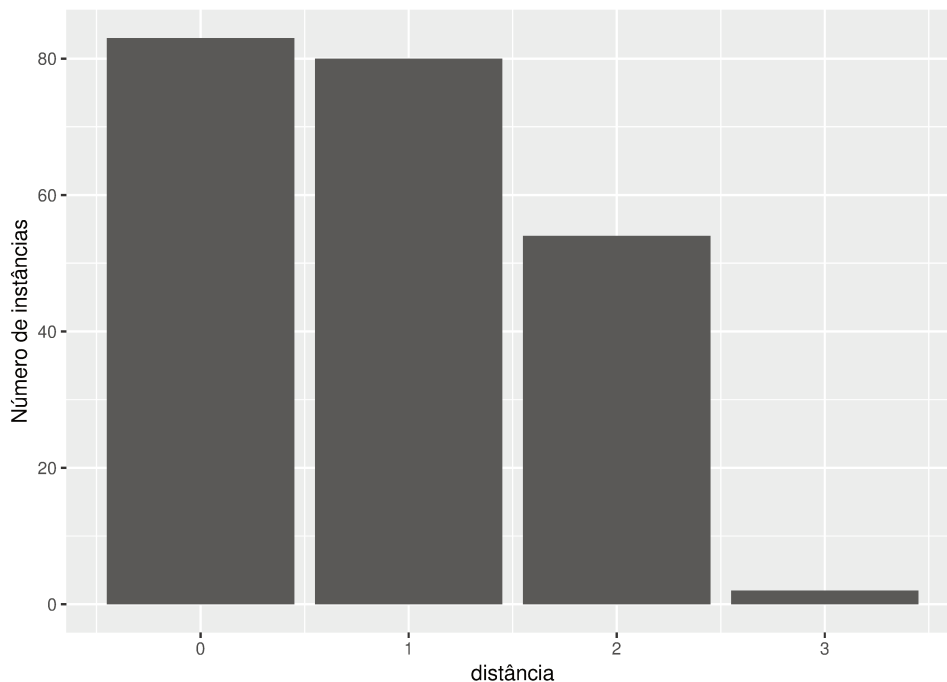


Figura 5.4: Distância entre soluções.

Parâmetro ρ . Para o tamanho do *pool* de soluções, denotado por ρ , um valor inicial de 25 foi utilizado. Esta estimativa foi baseada na observação de que cerca de $1/25$ do tempo limite restante era consumido quando a busca local era aplicada somente na melhor solução produzida pela heurística. Posteriormente, foi feito um experimento com todos os 120 grafos do conjunto BBGRL e $D \in \{1, 2, \dots, 10\}$, totalizando 1200 instâncias. Definimos o valor inicial de $\rho = 25$. Um tempo limite de $n/2$ foi imposto. Nesse experimento, a busca local foi aplicada em cada solução do *pool* e a melhor solução encontrada no final

era retornada. Nosso objetivo era estimar um limite superior para o índice da solução do *pool* que leva à melhor solução final, uma vez que o mesmo estava sempre ordenado segundo a qualidade das soluções.

A Figura 5.5 apresenta o número de instâncias (eixo y) nas quais a solução de respectivo índice do *pool* (eixo x) levou à melhor solução final. Cada gráfico corresponde a um valor de D , indicado logo acima.

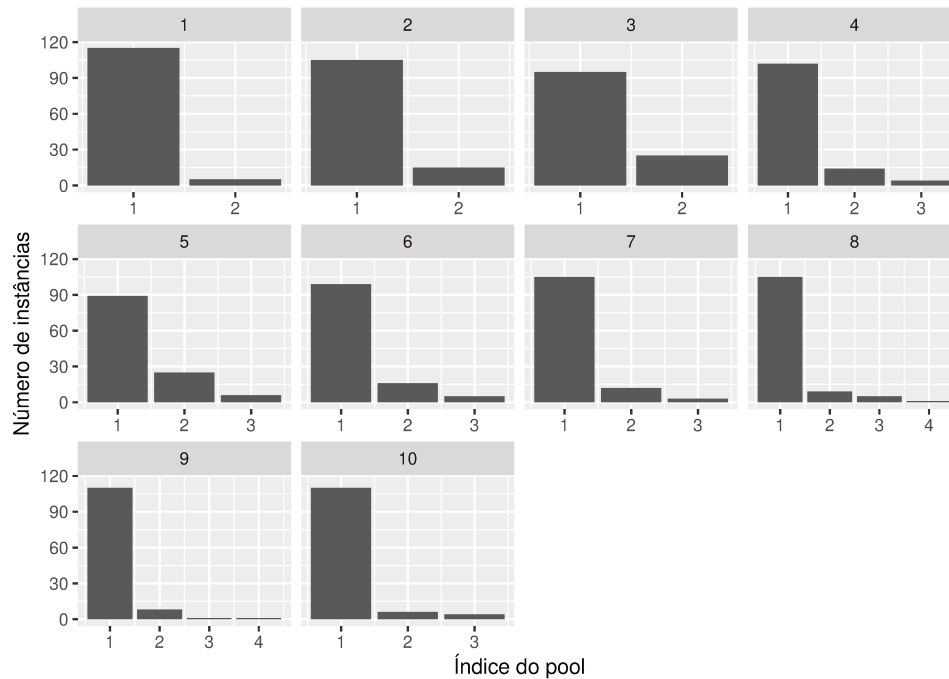


Figura 5.5: Número de instâncias nas quais o respectivo índice do *pool* levou à melhor solução final.

Notamos que a solução do *pool* que levava à melhor solução final após a busca local era no máximo a quarta. Também observamos que, na maioria dos casos, a primeira solução do *pool* foi a que levou à melhor solução final após aplicar a busca local. Visto isso, definimos $\rho = 4$.

Parâmetro f . Completamos o ajuste de parâmetros da matheurística comparando as propostas de funções f apresentadas na Seção 4.2.3, as quais são utilizadas para atribuir prioridades aos vértices ao construir uma vizinhança. Novamente realizamos os experimentos em um grupo seletivo de instâncias, desta vez constituído das instâncias não triviais dos conjuntos BBGRL e GBRL com $n = 500$ e $D \in \{1, \dots, 5\}$, totalizando 421 instâncias. Para análise dos resultados, primeiro aplicamos o teste de Iman-Davenport seguido do teste de Nemenyi, considerando os valores da função objetivo obtidos quando utilizando cada uma das diferentes funções. Pelo teste de Iman-Davenport, a hipótese nula foi rejeitada, ou seja, as funções não são equivalentes, uma vez que o resultado do teste foi 63.24 e o valor crítico é 2.37. O valor da diferença crítica pelo teste de Nemenyi foi 0.3 e o resultado do mesmo pode ser visualizado na Figura 5.6.

A função f -Desc foi a de menor *rank* médio, não foi encontrada DES entre as funções f -DegDesc, f -Deg e f -NeiDeg, enquanto a função f -Prox foi a de maior *rank* médio e

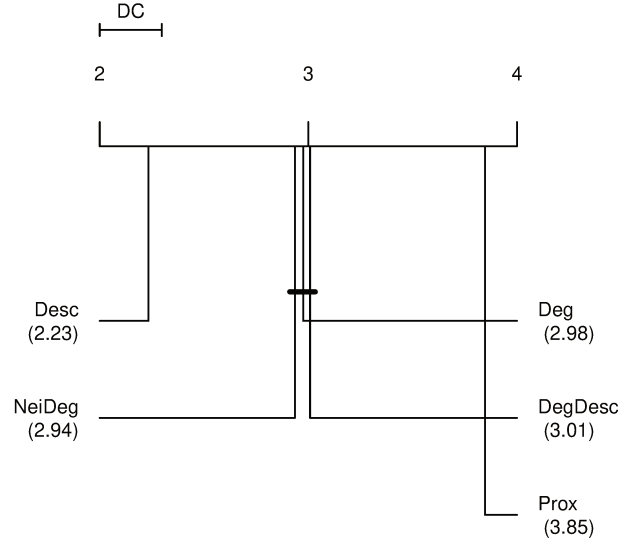


Figura 5.6: Resultados do teste de Nemenyi entre as funções utilizadas para construção da vizinhança.

consequentemente, a de pior desempenho nesse conjunto de instâncias. Dessa forma, prosseguimos com a função f -Desc para analisar a eficiência da nossa relaxação no algoritmo de construção de uma vizinhança apresentado na Seção 4.2.3 (Algoritmo 8). Lembramos que esta relaxação consiste em incluir as k -vizinhanças estritas do conjunto \mathcal{D}_{s_0} inteiras enquanto o tamanho desejado da vizinhança \overline{N}_{s_0} não seja excedido. Quando ocorre, ordenamos somente os elementos das vizinhanças restantes segundo a função f e então, selecionamos os vértices de maior prioridade. Por praticidade, denotamos a função f -Desc com o Algoritmo 8 por f -Desc-Relax.

O teste de Wilcoxon entre a função f -Desc com o algoritmo original de construção de vizinhança e da combinação f -Desc-Relax mostrou que não foi encontrada DES, com o resultando do teste sendo $z = -1.11$. Ademais, observamos que a combinação f -Desc-Relax foi melhor em 74 instâncias e pior em 58 (de um total de 421). Visto isso, decidimos usar essa estratégia de construção da vizinhança no procedimento de busca local no restante dos testes.

Comparação com resultados da literatura. Uma vez que encontramos valores apropriados para os parâmetros da MATHEU, continuamos com os experimentos finais de maneira a comparar nossos resultados com os que estão presentes em trabalhos publicados previamente.

Iniciamos com as comparações com as estratégias baseadas em *Ant Colony Optimization* de Blum et al. [7] (incluindo os resultados do CPLEX também relatados pelos autores) e a estratégia VNS de Hu, Windbichler e Raidl [39]. Segundo o teste de Iman-Davenport, as estratégias não são equivalentes, já que o resultado do teste foi 57.44 e o valor crítico é 2.39. A diferença crítica computada pelo teste de Nemenyi foi 0.56, como pode ser visto na Figura 5.7.

Segundo a diferença crítica computada pelo teste de Nemenyi, dois grupos de estratégias sem DES entre si podem ser formados: um pela MATHEU e a VNS e outro pela VNS

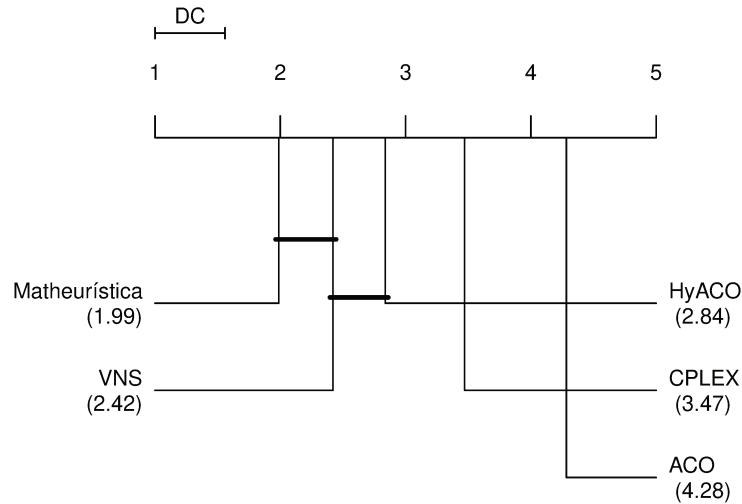


Figura 5.7: Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de Blum et al. [7] e Hu, Windbichler e Raidl [39] usando o conjunto BBGRL de instâncias.

e o HyACO. Considerando o primeiro grupo, o de estratégias de menor *rank* médio, o teste de Wilcoxon entre as duas estratégias mostrou que existe **DES** entre as mesmas com $z = -5.18$. Uma vez observado que as duas estratégias não são estatisticamente equivalentes, consideramos a contagem de vitórias e derrotas entre as duas, de maneira a avaliar qual das duas teve um melhor desempenho. A MATHEU é superior em 61 classes de problema e pior em 16 (de um total de 120). Considerando essa análise, podemos concluir que a MATHEU foi a melhor dentre todas as estratégias comparadas, nesse conjunto de instâncias.

Os autores Blum et al. [7] gentilmente providenciaram os resultados individuais para cada instância de seu trabalho e, uma vez que chegaram à conclusão que a estratégia HyACO foi a de melhor desempenho (em instâncias maiores com $n \in \{500, 1000\}$), comparamos seus resultados com a MATHEU. O teste Wilcoxon entre as duas estratégias mostra que existe **DES** entre as mesmas, com $z = -11.39$. A MATHEU foi melhor em 324 instâncias e pior em 59 (de um total de 1200).

A Tabela 5.8 apresenta o *gap*, em porcentagem, entre a solução da heurística Th na primeira fase da MATHEU e a obtida pela busca local posteriormente, o qual dá um indicativo do quanto a busca local foi capaz de melhorar a solução heurística.

Tabela 5.8: *Gap* (%) entre a solução obtida pela busca local e a solução inicial da heurística Th para as instâncias do conjunto BBGRL.

conjunto	n	min	média	mediana	desvio padrão	max
BBGRL	50	3.03	09.85	09.85	09.64	16.67
BBGRL	100	0.00	15.38	14.50	10.45	39.33
BBGRL	500	0.00	12.15	06.25	19.65	95.18
BBGRL	1000	0.00	13.16	06.07	21.05	97.19

O *gap* médio ficou entre 9 e 15.5%. Nas instâncias com $n \in \{500, 1000\}$ o *gap* máximo foi acima de 90% para ambos os casos. Houve instâncias nas quais a busca lo-

cal não foi capaz de melhorar a solução da heurística, especificamente nos casos com $n \in \{100, 500, 1000\}$, uma vez que o *gap* mínimo é de 0.00%.

Em seguida, comparamos nossos resultados com os de García-Martínez et al. [27] no conjunto de instâncias GBRL. Os autores chegaram à conclusão de que quatro dentre todas as estratégias propostas são as melhores, sendo essas: $ILP(L,T,+1)$, $ILP(2,F,+1)$, $H(ThDesc,Now)$ e $H(ThDegree,Now)$. Então, utilizamos as mesmas para nossas comparações. O teste de Iman-Davenport entre todas as estratégias encontrou que todas as estratégias não são equivalentes, uma vez que o seu resultado foi 53.31 e o valor crítico é 2.38. A diferença crítica computada pelo teste de Nemenyi foi 0.39, como pode ser visto na Figura 5.8.

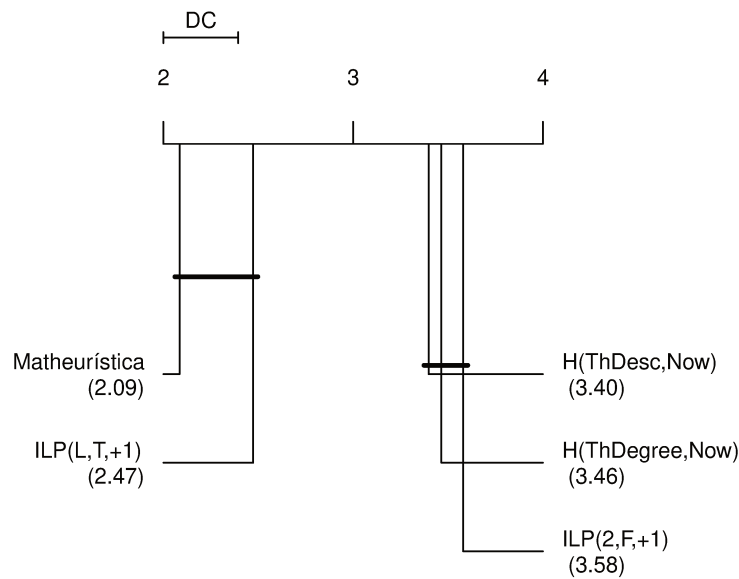


Figura 5.8: Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o conjunto GBRL de instâncias.

Segundo a diferença crítica computada pelo teste de Nemenyi, dois grupos de estratégias sem DES entre si podem ser formados, de maneira que o grupo com menor *rank* médio é formado pela MATHEU e a estratégia $ILP(L,T,+1)$. O teste de Wilcoxon entre estas duas estratégias mostrou que existe DES entre as mesmas com $z = -2.65$. Considerando a contagem de vitórias e derrotas entre as duas estratégias, a MATHEU é superior em 75 classes de problemas e pior em 54 (de um total de 240).

Uma vez que no trabalho de García-Martínez et al. [27] é feita distinção entre as instâncias, constituindo os dois conjuntos A-GBRL e G-GBRL, prosseguimos realizando uma análise separada para cada um destes conjuntos. Iniciando com o conjunto A-GBRL, o resultado do teste de Iman-Davenport mostrou que as estratégias não são equivalentes, uma vez que o resultado do teste foi 135.22 e o valor crítico é 2.39. A diferença crítica computada pelo teste de Nemenyi foi 0.55, como pode ser visto na Figura 5.9.

Observamos que a MATHEU ficou no grupo de menor *rank* médio, um pouco atrás da estratégia $ILP(L,T,+1)$. O teste de Wilcoxon entre as duas estratégias encontrou DES com $z = -2.00$. Considerando a contagem de vitórias e derrotas entre as duas estratégias, a MATHEU é superior em 28 classes de problema e pior em 48 (de um total de 120).

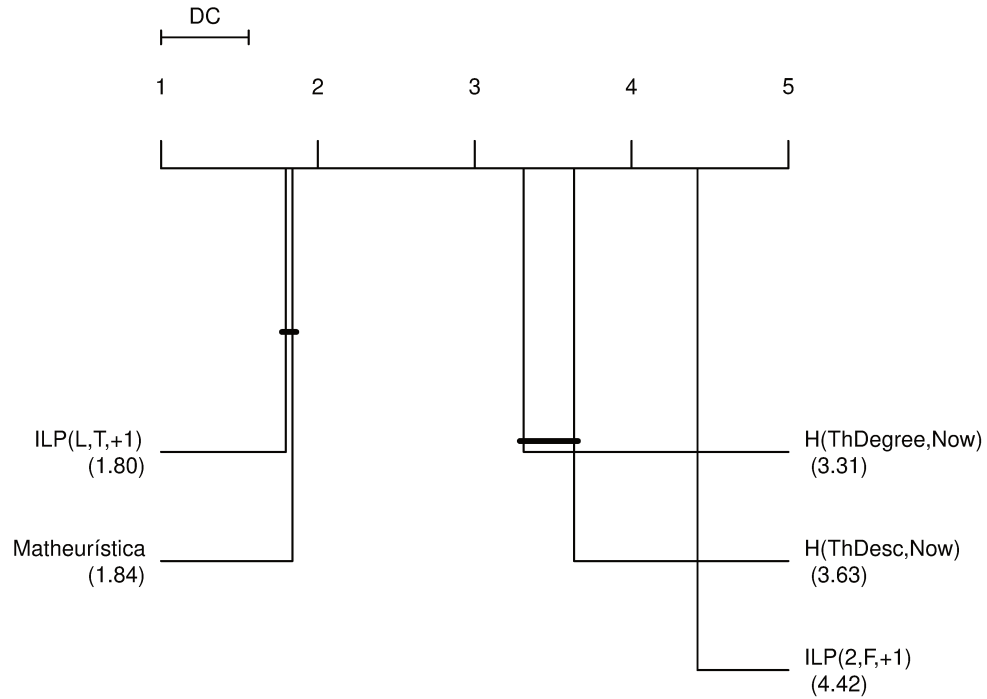


Figura 5.9: Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o subconjunto A-GBRL de instâncias.

A Tabela 5.9 apresenta estatísticas para a diferença entre os resultados de média de valores da função objetivo por classes de problema da estratégia ILP(L,T,+1) e a MATHEU. Foram examinadas as 48 classes de problemas nas quais da MATHEU teve desempenho inferior, sendo essas as com $n \in \{100, 500, 1000\}$. Notamos que a diferença foi no máximo 1.9 vértices com uma média 0.69 e um desvio padrão de 0.48.

Tabela 5.9: Estatísticas das diferenças entre os resultados de nossa matheurística e ILP(L,T,+1), quando os da primeira foram piores.

min	mediana	média	desvio padrão	max
0.10	0.60	0.69	0.48	1.90

As mesmas estatísticas são indicadas na Tabela 5.10 para as 23 classes de problema onde a matheurística apresentou desempenho superior, com respeito aos mesmo valores de $n \in \{100, 500, 1000\}$. Observamos que nossa matheurística, no máximo, salvou 531.9 vértices a mais com uma média de 44.33 e um desvio padrão de 143.86.

Tabela 5.10: Estatísticas das diferenças entre os resultados de nossa matheurística e ILP(L,T,+1), quando os da primeira foram melhores.

min	mediana	média	desvio padrão	max
0.10	0.70	44.33	143.86	531.90

Considerando as instâncias do conjunto G-GBRL, as estratégias não são equivalentes segundo o teste de Iman-Davenport, uma vez que o valor retornado pelo teste foi 12.38

e o valor crítico é 2.39. Uma diferença crítica de valor 0.55 foi computada pelo teste de Nemenyi, formando três grupos de estratégias sem DES entre si, como pode ser visto na Figura 5.10.

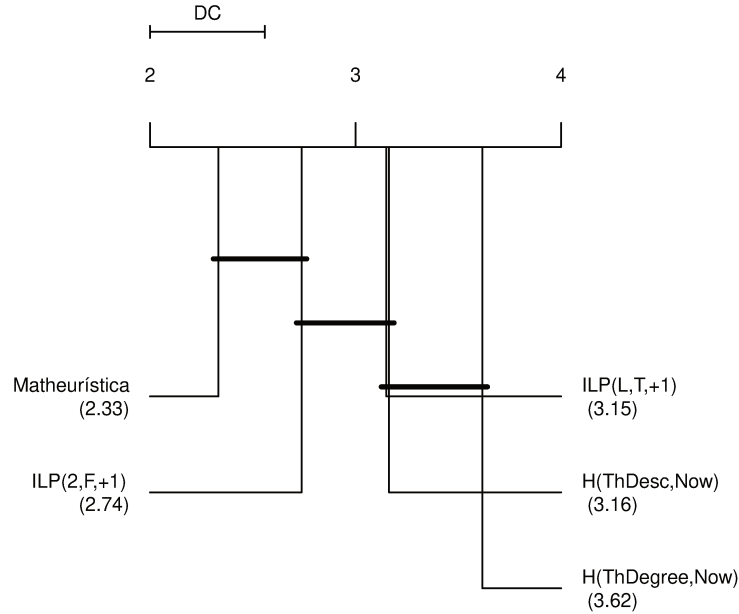


Figura 5.10: Resultados do teste de Nemenyi comparando a MATHEU com as estratégias de García-Martínez et al. [27] usando o subconjunto G-GBRL de instâncias.

O grupo de estratégias com menor *rank* médio é formado pela MATHEU e a estratégia ILP(2,F,+1). Calculamos o teste de Wilcoxon entre as duas e DES foi identificada com $z = -3.11$. Considerando a contagem de vitórias e derrotas entre as duas estratégias, a MATHEU é superior em 39 classes de problemas e foi pior em 14 (de um total de 120).

A Tabela 5.11 apresenta o *gap*, em porcentagem, entre a solução obtida pela heurística Th na primeira fase da MATHEU e a solução retornada pela busca local.

Tabela 5.11: *Gap* (%) entre a solução obtida pela busca local e a solução inicial da heurística Th para as instâncias do conjunto GBRL.

conjunto	n	min	média	mediana	desvio padrão	max
G-GBRL	100	5.56	24.83	17.60	22.24	60.00
G-GBRL	500	0.00	06.71	00.00	18.41	90.93
G-GBRL	1000	0.00	09.78	00.00	23.10	94.28
A-GBRL	100	0.00	13.04	11.11	08.74	34.48
A-GBRL	500	0.00	07.67	04.26	08.38	37.50
A-GBRL	1000	0.00	07.82	04.00	08.48	33.33

O *gap* médio ficou de 6 a 25%, com valores relativamente altos de desvio padrão. Nas instâncias do conjunto G-GBRL tanto o *gap* médio quanto o máximo foi estritamente maior do que o observado nas instâncias do conjunto A-GBRL, provavelmente devido ao fato de que a heurística Th apresentou um desempenho inferior nas instâncias do conjunto G-GBRL, deixando mais oportunidade para que a busca local melhorasse a solução. Em

quase todos os casos, exceto no conjunto **G-GBRL** com $n \in \{100\}$, em algumas instâncias a busca local não foi capaz de melhorar a solução da heurística, uma vez que o *gap* mínimo é de 0.00%.

Considerando os resultados apresentados, podemos inferir que a **MATHEU** obteve desempenho superior, na maioria das instâncias, em comparação com as estratégias de Blum et al. [7] e Hu, Windbichler e Raidl [39] e também no caso geral em relação às estratégias de García-Martínez et al. [27]. No que diz respeito ao subconjunto **A-GBRL** de García-Martínez et al. [27], a **MATHEU** obteve resultados levemente inferiores aos da estratégia ILP(L,T,+1).

5.5 Teste de Robustez

Nessa seção, apresentamos o que chamamos de *Teste de Robustez* da **MATHEU**, o qual indica se a estratégia é robusta ou não baseando-se na distribuição dos resultados obtidos em múltiplas execuções em uma mesma instância. Múltiplas execuções são utilizadas visto que a **MATHEU** possui uma fase que depende de escolhas aleatórias e pode produzir diferentes resultados para uma mesma instância, dado que utilizamos uma semente diferente para o gerador de números pseudo-aleatórios em cada execução. A **MATHEU** é dita robusta se a variação dos resultados das execuções é pequena, significando que os dados não estão muito dispersos. Daí, podemos inferir que resultados extremamente diferentes em uma mesma instância vão ocorrer com baixa probabilidade.

Os experimentos para o teste de robustez foram realizados em um conjunto seletivo de 12 instâncias, as quais são apresentadas na Tabela 5.12. A coluna **n** mostra o número de vértices da instância, a coluna **modelo** indica qual modelo foi utilizado para gerar a instância, seguida pela coluna **parâmetro** que descreve qual o parâmetro que foi usado no modelo. A coluna **contador** apresenta o número da instância entre as 10 que foram geradas com a respectiva combinação de **n** e **parâmetro**. O conjunto ao qual a instância pertence é indicado na coluna seguinte e, por fim, a coluna **id** corresponde aos identificadores únicos que foram atribuídos às instâncias e serão utilizados durante a análise.

Essas instâncias foram selecionadas baseando-se nos valores de função objetivo obtidos pela nossa estratégia. Instâncias triviais e aquelas nas quais quase todos os vértices são queimados não foram consideradas. A razão dessa decisão é que, na prática, qualquer estratégia razoavelmente boa para defender os vértices do grafo vai encontrar essencialmente a mesma solução nessas instâncias. Então, elas não são adequadas para o teste de robustez. Dessa forma, decidimos realizar os testes em instâncias nas quais o número de vértices queimados fica entre esses dois casos extremos. Os valores para a quantidade de brigadistas disponíveis utilizados foram $D \in \{1, \dots, 5\}$.

A Figura 5.11 apresenta *boxplots* dos valores de função objetivo normalizados pela quantidade de vértices da respectiva instância. O eixo x corresponde aos identificadores das instâncias descritos na Tabela 5.12, enquanto o eixo y indica os valores normalizados. Os gráficos são separados por valor de D , apresentado logo acima dos mesmos. Em cada *boxplot*, a caixa branca indica o primeiro quartil (linha horizontal de baixo), segundo quartil ou mediana (linha horizontal do meio) e o terceiro quartil (linha horizontal de

Tabela 5.12: Instâncias selecionadas para o teste de robustez.

n	modelo	parâmetro	contador	conjunto	id
50	Grafos Aleatórios	0.2000	10	BBGRL	1
100		0.1000	7	BBGRL	2
500		0.0150	7	BBGRL	3
1000		0.0075	10	BBGRL	4
50	Grafos Geométricos Aleatórios	0.3340	1	G-GBRL	5
100		0.1950	1	G-GBRL	6
500		0.0930	3	G-GBRL	7
1000		0.0650	7	G-GBRL	8
50	Grafos Aleatórios	0.1500	3	R-GBRL	9
100		0.0750	8	R-GBRL	10
500		0.0150	9	R-GBRL	11
1000		0.0075	8	R-GBRL	12

cima). O valor mínimo é representado pela linha vertical que se estende logo abaixo da caixa e o valor máximo pela linha vertical acima. *Outliers* são indicados por pontos alinhados verticalmente com a caixa. Cada *boxplot* apresenta os dados das 30 execuções da respectiva instância.

O *boxplot* provê uma noção de distribuição dos dados, uma vez que quão mais próximos entre si estão os valores dos quartis (quão mais “achatado” está o *boxplot*), menos dispersos estão os dados.

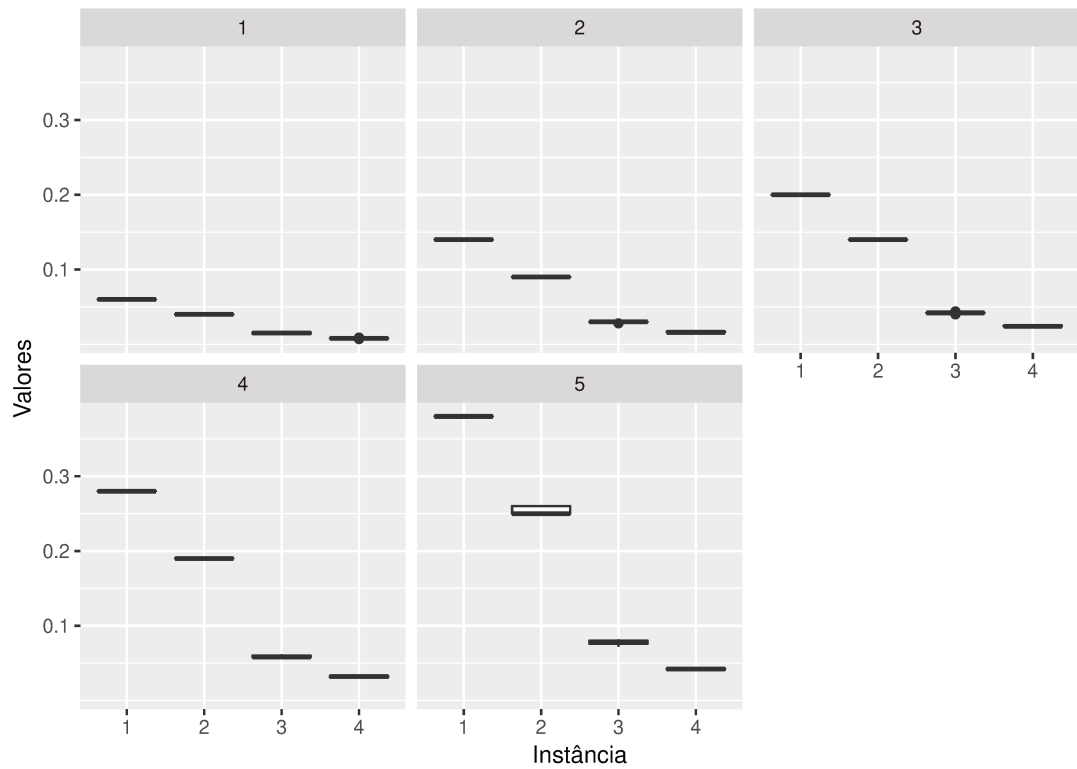
Considerando as instâncias geradas pelo modelo de grafos aleatórios, pouca variância é observada e também uma pequena ocorrência de *outliers*, em ambos conjuntos de instâncias BBGRL e GBRL. Então, tais resultados sugerem que a MATHEU é robusta para esse caso particular.

Variância maior é observada nas instâncias geradas com o modelo geométrico aleatório, especialmente nos casos com uma maior quantidade de vértices ($n \in \{500, 1000\}$). Por exemplo, no caso de $D = 5$ com a instância de id 8, a diferença entre o primeiro e o terceiro quartil é aproximadamente 0.25, o que em termos de vértices salvos equivale a cerca de 250 vértices de um total de 1000. Como mencionado anteriormente, acreditamos que tal variação maior decorre do fato de que esses grafos possuem uma estrutura agrupada, na qual a heurística Th aleatorizada não possui um bom desempenho. Então, para grafos geométricos aleatórios, a MATHEU não se mostrou tão robusta.

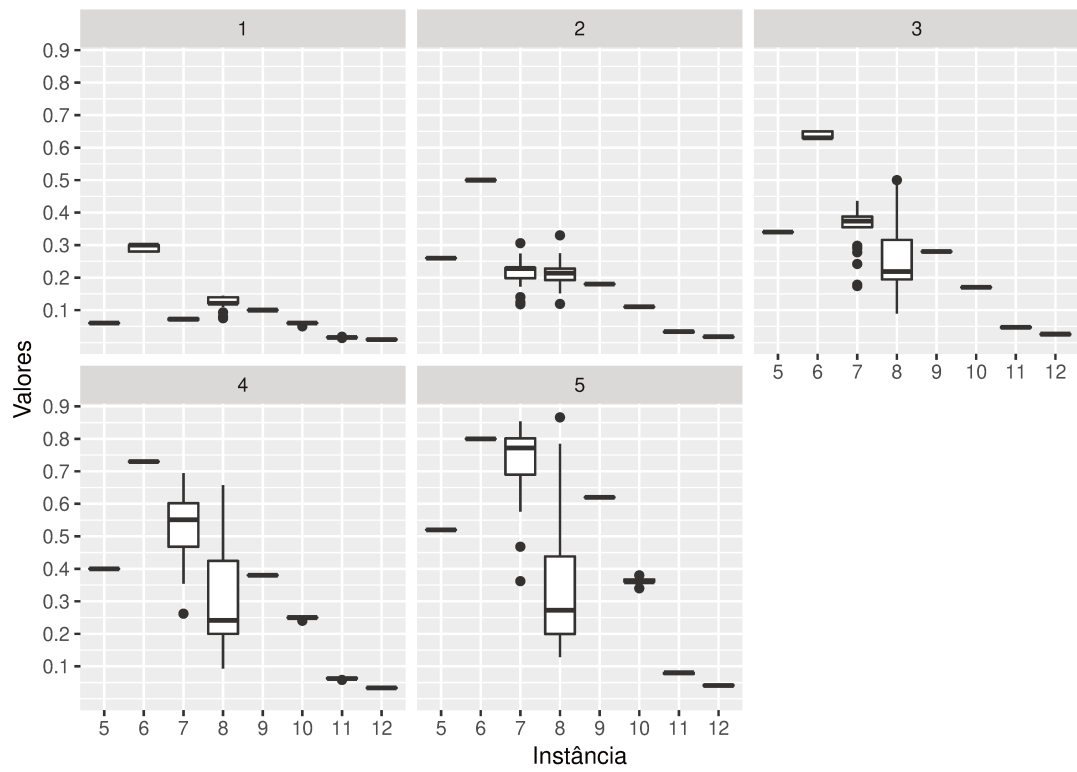
5.6 Busca Local com Diversificação e Intensificação

De maneira a avaliar a eficácia da MATHEU-DI da Seção 4.2.6, comparamos seus resultados com os da MATHEU, apresentada na Seção 4.2. Utilizamos um tempo limite de dez minutos, com o mesmo ambiente computacional descrito na Seção 5.1.

Para esse experimento, utilizamos as instâncias dos conjuntos BBGRL e G-GBRL, junto das que foram geradas segundo a metodologia discutida na Seção 4.4 (GER-ER e GER-BA).



(a) Instâncias do conjunto BBGRL.



(b) Instâncias do conjunto GBRL.

Figura 5.11: Resultados do teste de robustez.

Não utilizamos o conjunto **A-GBRL** nesse experimento devido ao alto tempo de computação (dado o considerável número de instâncias) e porque o mesmo foi gerado utilizando o mesmo modelo do conjunto **BGRL**.

Usando o modelo $G(n, m)$ de Erdős e Rényi [22] discutido na Seção 4.4, geramos 120 instâncias com valores de n e m apresentados na Tabela 5.13.

Tabela 5.13: Tamanhos e parâmetros das instâncias do conjunto **GER-ER**.

n	m	Grau Médio
50	{180, 246, 309}	{7.20, 9.84, 12.36}
100	{374, 496, 625}	{7.48, 9.92, 12.40}
125	{450, 614, 771}	{7.20, 9.82, 12.34}
150	{561, 744, 930}	{7.48, 9.92, 12.40}

Os valores de m foram escolhidos seguindo a mesma metodologia de García-Martínez et al. [27], de maneira a obter grafos com graus médios aproximados de {7.5, 10, 12.5}.

Utilizando o modelo de Barabási e Albert [4], geramos 180 instâncias, com valores de n e m_0 apresentados na Tabela 5.14. Como foi discutido na Seção 4.4, o parâmetro m_0 corresponde à quantidade de vértices iniciais no algoritmo do modelo de Barabási e Albert [4].

Tabela 5.14: Tamanhos e parâmetros das instâncias do conjunto **GER-BA**.

n	m_0	Grau Médio
50	{4, 6, 7}	{7.36, 10.56, 12.04}
100	{4, 5, 7}	{7.68, 9.50, 13.02}
125	{4, 5, 7}	{7.74, 9.60, 13.22}
150	{4, 5, 7}	{7.79, 9.67, 13.35}
200	{4, 5, 6}	{7.84, 9.75, 11.64}
300	{4, 5, 6}	{7.89, 9.83, 11.76}

Para essas instâncias, também desejávamos que o grau médio ficasse em torno de {7.5, 10, 12.5}, então, definimos o valor de m_0 do seguinte modo. De forma a estimar o número de arestas m que o grafo gerado deveria ter, utilizamos a Equação (5.1), na qual \bar{g} indica o grau médio desejado. A divisão por 2 é feita uma vez que estamos lidando com grafos não direcionados.

$$m = \left\lceil \frac{n \cdot \bar{g}}{2} \right\rceil \quad (5.1)$$

Dado o valor de m , podemos estimar o valor de m_0 , usando a Equação (5.2). No modelo de Barabási e Albert [4], um vértice é adicionado por vez, assim, são $n - m_0$ inserções, cada qual acrescentando m_0 arestas ao grafo.

$$\begin{aligned}
m &= m_0 \cdot (n - m_0) \\
m &= -m_0^2 + m_0 \cdot n \\
-m_0^2 + m_0 \cdot n - m &= 0
\end{aligned} \tag{5.2}$$

Após a resolução do polinômio da Equação (5.2), duas raízes são obtidas. Seleccionamos o teto da menor entre as duas para ser o valor de m_0 .

Utilizando todos os conjuntos de instâncias, BBGRL, GBRL, GER-ER e GER-BA, temos um total de 5400 instâncias com $D \in \{1, 2, \dots, 10\}$. Os resultados deste experimento podem ser vistos na Tabela 5.15. A primeira coluna indica o conjunto de instâncias, a segunda exibe o resultado do teste de Wilcoxon entre as duas estratégias. A terceira e quarta coluna reportam em quantas instâncias (de um total indicado na quinta coluna) a estratégia correspondente obteve melhor desempenho.

Tabela 5.15: Comparação dos resultados das estratégias MATHEU e MATHEU-DI.

Conjunto	Wilcoxon	MATHEU	MATHEU-DI	Total de Instâncias
BBGRL	-0.069	104	105	1200
G-GBRL	-0.437	70	62	1200
GER-ER	-0.551	30	41	1200
GER-BA	-1.490	42	5	1800

Observamos que para todos os conjuntos de instâncias, o teste de Wilcoxon não identificou DES entre as duas estratégias. Nos conjuntos G-GBRL e GER-BA, a MATHEU obteve melhores resultados em mais instâncias, com uma diferença de desempenho maior no conjunto GER-BA. Quanto a MATHEU-DI, esta foi melhor nos demais conjuntos, com uma diferença maior observada no conjunto GER-ER. Concluimos então que para essas instâncias e seguindo esta metodologia de análise, a MATHEU-DI não demonstrou resultados com DES em relação a MATHEU.

Capítulo 6

Considerações Finais

O objetivo principal deste trabalho foi a realização de um estudo computacional do FFP, através do desenvolvimento e avaliação de técnicas de solução que incluem algoritmos exatos (através de Programação Linear Inteira) e métodos heurísticos.

Considerando os métodos exatos, realizamos melhorias, em termos de *speedup*, no modelo PLI tradicional do FFP proposto por Develin e Hartke [17] (denotado por **FFMO**), feitas através de técnicas de fixação de variáveis e agregação de restrições. Em instâncias nas quais a solução ótima pôde ser encontrada em no máximo uma hora (em grafos com 50 e 100 vértices), conseguimos um *speedup* de aproximadamente 2 em média. Observamos que tais modificações podem levar a geração de soluções infactíveis, mas conseguimos demonstrar que é possível torná-las factíveis em tempo polinomial. Também desenvolvemos um novo modo de ajustar o parâmetro T do modelo **FFMO**, o qual é uma estimativa do número de rodadas necessárias para se obter uma solução ótima. Nossa estratégia de ajuste se mostrou efetiva para adquirir soluções sem garantia de otimalidade em um tempo de computação pequeno, o que foi fundamental para a eficácia do nosso procedimento de busca local.

Em relação aos métodos heurísticos, desenvolvemos uma *matheurística* (denotada por **MATHEU**), uma técnica que se baseia na interoperação entre meta-heurísticas e programação matemática. Alguns trabalhos presentes na literatura já haviam feito estudos computacionais do FFP utilizando métodos heurísticos, sendo esses os de Blum et al. [7], García-Martínez et al. [27] e Hu, Windbichler e Raidl [39]. Visto isso, comparamos nossos resultados com os desses autores, através do uso de testes estatísticos não paramétricos. Por meio dessa análise, foi observado que existe diferença estatisticamente significativa entre a **MATHEU** e as demais estratégias, ao mesmo tempo que nossos resultados foram superiores aos da literatura, na maioria das instâncias. Isso é um indicativo de que nosso método pode ser considerado como o melhor dentre os que foram comparados. Também propomos uma variação da **MATHEU**, a qual não se mostrou ser significativamente diferente da versão original do ponto de vista estatístico.

De modo a complementar o *benchmark* disponível para o FFP, constituído das instâncias de Blum et al. [7] e García-Martínez et al. [27], geramos aleatoriamente grafos a partir dos modelos de Barabási e Albert [4] e Erdős e Rényi [22]. Tais instâncias e todos os nossos resultados estão disponíveis em www.ic.unicamp.br/~cid/Problem-instances/Firefighter-in-Graphs.

Realizamos neste trabalho um *teste de robustez*, para verificar se as soluções da MATHEU exibem pouca variação quando múltiplas repetições são executadas em um mesmo conjunto de instâncias. Observamos que, para a maioria das instâncias testadas, a MATHEU se mostrou robusta, exceto nas instâncias de García-Martínez et al. [27] geradas aleatoriamente por um modelo baseado em conceitos geométricos. Segundo nosso conhecimento, esse foi o primeiro trabalho a fazer tal teste em um estudo computacional relacionado ao FFP.

Como trabalhos futuros, dado que criamos uma estratégia de construção de vizinhança de uma solução e busca local, existem oportunidades de se aplicar as mesmas ideias em meta-heurísticas como *Tabu Search* [32] e *Simulated Annealing* [44], entre outras. Visto que construímos um *pool* de soluções, uma estratégia natural de se utilizar seria o *Path Relinking* [31], discutido na Seção 2.4.

As técnicas desenvolvidas neste trabalho podem ser aplicadas a outras variantes do FFP como o *Politician's Firefighting* [57] ou as que foram apresentadas por Anshelevich et al. [2]. Também podem ser consideradas vertentes as quais estejam mais próximas do problema do mundo real, considerando, por exemplo, velocidade de propagação do fogo, quantidade dinâmica de brigadistas e pesos nos vértices os quais indicam a prioridade de se salvar o respectivo vértice.

Bibliografia

- [1] Equipe Sintecsys Agtech. *Maiores causas de incêndios florestais*. Jul. de 2017. URL: <http://sintecsys.com/maiores-causas-de-incendios-florestais/> (acesso em 23/01/2018).
- [2] Elliot Anshelevich et al. “Approximability of the firefighter problem”. Em: *Algorithmica* 62.1-2 (2012), pp. 520–536. DOI: 10.1007/s00453-010-9469-y.
- [3] OpenMP ARB. “OpenMP application program interface, v. 4.0”. Em: *OpenMP Architecture Review Board* (2013).
- [4] Albert-László Barabási e Réka Albert. “Emergence of scaling in random networks”. Em: *Science* 286.5439 (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509.
- [5] Cristina Bazgan, Morgan Chopin e Michael R. Fellows. “Parameterized complexity of the firefighter problem”. Em: *Algorithms and Computation*. Yokohama, Japan: Springer, 2011, pp. 643–652. DOI: 10.1007/978-3-642-25591-5_66.
- [6] BBC. *BBC Earth – Forest fire videos – See how fire started on Earth*. Out. de 2015. URL: http://www.bbc.co.uk/science/earth/natural_disasters/forest_fire (acesso em 23/01/2018).
- [7] Christian Blum et al. “The firefighter problem: application of hybrid ant colony optimization algorithms”. Em: *Evolutionary Computation in Combinatorial Optimisation*. Granada, Spain: Springer, 2014, pp. 218–229. DOI: 10.1007/978-3-662-44320-0_19.
- [8] Marco A. Boschetti et al. “Matheuristics: Optimization, Simulation and Control”. Em: *Hybrid Metaheuristics: 6th International Workshop, HM 2009, Udine, Italy, October 16-17, 2009. Proceedings*. Ed. por María J. Blesa et al. Berlin, Heidelberg: Springer, 2009, pp. 171–177. ISBN: 978-3-642-04918-7. DOI: 10.1007/978-3-642-04918-7_13.
- [9] Leizhen Cai, Elad Verbin e Lin Yang. “Firefighting on trees: $(1 - 1/e)$ -approximation, fixed parameter tractability and a subexponential algorithm”. Em: *Algorithms and Computation*. Gold Coast, Australia: Springer, 2008, pp. 258–269. DOI: 10.1007/978-3-540-92182-0_25.
- [10] Marco Caserta e Stefan Voß. “Metaheuristics: intelligent problem solving”. Em: *Matheuristics*. Springer, 2009, pp. 1–38.

- [11] Janka Chlebíková e Morgan Chopin. “The Firefighter Problem: A Structural Analysis”. Em: *Parameterized and Exact Computation: 9th International Symposium, IPEC 2014, Wrocław, Poland, September 10-12, 2014. Revised Selected Papers*. Ed. por Marek Cygan e Pinar Heggernes. Cham: Springer, 2014, pp. 172–183. ISBN: 978-3-319-13524-3. DOI: 10.1007/978-3-319-13524-3_15.
- [12] Janka Chlebíková e Morgan Chopin. “The firefighter problem: Further steps in understanding its complexity”. Em: *Theoretical Computer Science* 676 (2017), pp. 42–51. DOI: 10.1016/j.tcs.2017.03.004.
- [13] Thomas H. Cormen et al. *Introduction to Algorithms, 3rd Edition (MIT Press)*. The MIT Press, 2009. ISBN: 978-0-262-03384-8.
- [14] IBM Corporation. *IBM ILOG CPLEX Optimization Studio CPLEX User’s Manual - Version 12 Release 6*. IBM Corporation. 2015.
- [15] Marek Cygan, Fedor V. Fomin e Erik Jan van Leeuwen. “Parameterized Complexity of Firefighting Revisited.” Em: *IPEC*. Vol. 7112. Springer. Saarbrücken, Germany: Springer, 2011, pp. 13–26. DOI: 10.1007/978-3-642-28050-4_2.
- [16] Janez Demšar. “Statistical comparisons of classifiers over multiple data sets”. Em: *The Journal of Machine Learning Research* 7 (2006), pp. 1–30.
- [17] Mike Develin e Stephen G. Hartke. “Fire containment in grids of dimension three and higher”. Em: *Discrete Applied Mathematics* 155.17 (2007), pp. 2257–2268. DOI: 10.1016/j.dam.2007.06.002.
- [18] Jay L Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2011.
- [19] Reinhard Diestel. *Graph Theory*. Springer, 2017. DOI: 10.1007/978-3-662-53622-3.
- [20] Marco Dorigo, Vittorio Maniezzo e Alberto Coloni. “Ant system: optimization by a colony of cooperating agents”. Em: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.
- [21] Rodney G Downey e Michael R Fellows. *Parameterized complexity*. Springer, 2012. DOI: 10.1007/978-1-4612-0515-9.
- [22] Paul Erdős e Alfréd Rényi. “On random graphs. I”. Em: *Publicationes Mathematicae (Debrecen)* 6 (1959), pp. 290–297.
- [23] Thomas A. Feo e Mauricio G. C. Resende. “Greedy Randomized Adaptive Search Procedures”. Em: *Journal of Global Optimization* 6.2 (1995), pp. 109–133. DOI: 10.1007/BF01096763.
- [24] Paola Festa e Mauricio GC Resende. “GRASP: An annotated bibliography”. Em: *Essays and surveys on metaheuristics* 6 (2002), pp. 325–367.
- [25] Stephen Finbow e Gary MacGillivray. “The Firefighter Problem: a survey of results, directions and questions”. Em: *Australasian Journal of Combinatorics* 43 (2009), pp. 57–78. URL: http://ajc.maths.uq.edu.au/pdf/43/ajc_v43_p057.pdf.

- [26] Stephen Finbow et al. “The firefighter problem for graphs of maximum degree three”. Em: *Discrete Mathematics* 307.16 (2007), pp. 2094–2105. DOI: 10.1016/j.disc.2005.12.053.
- [27] Carlos García-Martínez et al. “The firefighter problem: Empirical results on random graphs”. Em: *Computers & Operations Research* 60 (2015), pp. 55–66. DOI: 10.1016/j.cor.2015.02.004.
- [28] Michael R. Garey e David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.
- [29] Nathan T Georgette. “Predicting the herd immunity threshold during an outbreak: a recursive approach”. Em: *PLoS ONE* 4.1 (2009), e4168. DOI: 10.1371/journal.pone.0004168.
- [30] E. N. Gilbert. “Random Graphs”. Em: *The Annals of Mathematical Statistics* 30.4 (dez. de 1959), pp. 1141–1144. DOI: 10.1214/aoms/1177706098.
- [31] F. Glover, M. Laguna e R. Marti. “Fundamentals of scatter search and path relinking”. English. Em: *Control and Cybernetics* Vol. 29, no 3 (2000), pp. 653–684.
- [32] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. Em: *Computers & Operations Research* 13.5 (1986), pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1.
- [33] Fred Glover. “Tabu Search - Part I”. Em: *INFORMS Journal on Computing* 1.3 (1989), pp. 190–206. DOI: 10.1287/ijoc.1.3.190. URL: <https://doi.org/10.1287/ijoc.1.3.190>.
- [34] Aric A. Hagberg, Daniel A. Schult e Pieter J. Swart. “Exploring network structure, dynamics, and function using NetworkX”. Em: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, ago. de 2008, pp. 11–15.
- [35] Pierre Hansen, Vittorio Maniezzo e Stefan Voß. “Special issue on mathematical contributions to metaheuristics editorial”. Em: *Journal of Heuristics* 15.3 (2009), p. 197.
- [36] Bert Hartnell. “Firefighter! An application of domination. Presentation”. Em: *25th Manitoba Conference on Combinatorial Mathematics and Computing, University of Manitoba in Winnipeg, Canada*. 1995.
- [37] Bert Hartnell e Qiyang Li. “Firefighting on trees: How bad is the greedy algorithm?”. Em: *Congressus Numerantium* (2000), pp. 187–192.
- [38] Sture Holm. “A simple sequentially rejective multiple test procedure”. Em: *Scandinavian journal of statistics* (1979), pp. 65–70.
- [39] Bin Hu, Andreas Windbichler e Günther R. Raidl. “A New Solution Representation for the Firefighter Problem”. Em: *Evolutionary Computation in Combinatorial Optimization - 15th European Conference, EvoCOP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*. Ed. por Gabriela Ochoa e Francisco Chicano. Vol. 9026. Lecture Notes in Computer Science. Springer, 2015, pp. 25–35. ISBN: 978-3-319-16467-0. DOI: 10.1007/978-3-319-16468-7_3.

- [40] Ronald L. Iman e James M. Davenport. “Approximations of the critical region of the Friedman statistic”. Em: *Communications in Statistics - Theory and Methods* 9.6 (1980), pp. 571–595. DOI: 10.1080/03610928008827904.
- [41] Laboratório de Incêndios Florestais. *Técnicas de Combate*. URL: <http://www.floresta.ufpr.br/firelab/tecnicas-de-combate/> (acesso em 23/01/2018).
- [42] Yutaka Iwaikawa, Naoyuki Kamiyama e Tomomi Matsui. “Improved approximation algorithms for firefighter problem on trees”. Em: *IEICE TRANSACTIONS on Information and Systems* 94.2 (2011), pp. 196–199. DOI: 10.1587/transinf.E94.D.196.
- [43] Andrew King e Gary MacGillivray. “The firefighter problem for cubic graphs”. Em: *Discrete Mathematics* 310.3 (2010), pp. 614–621. DOI: 10.1016/j.disc.2009.05.007.
- [44] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi et al. “Optimization by simulated annealing”. Em: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.
- [45] Cai Leizhen e Wang Weifan. “The Surviving Rate of a Graph for the Firefighter Problem”. Em: *SIAM Journal on Discrete Mathematics* 23.4 (2009), pp. 1814–1826. DOI: 10.1137/100791130.
- [46] Gary MacGillivray e Ping Wang. “On the firefighter problem”. Em: *Journal of Combinatorial Mathematics and Combinatorial Computing* 47 (2003), pp. 83–96.
- [47] Vittorio Maniezzo, Thomas Sttze e Stefan Vo. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. 1ª ed. Springer, 2009. ISBN: 978-1-4419-1306-7.
- [48] *Matheuristics conferences page*. URL: astarte.csr.unibo.it/matheuristics/.
- [49] Krzysztof Michalak. “Auto-adaptation of genetic operators for multi-objective optimization in the firefighter problem”. Em: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2014, pp. 484–491. DOI: 10.1007/978-3-319-10840-7_58.
- [50] Krzysztof Michalak. “ED-LS-A Heuristic Local Search for the Multiobjective Firefighter Problem”. Em: *Applied Soft Computing* (2017). DOI: 10.1016/j.asoc.2017.05.049.
- [51] Krzysztof Michalak. “Estimation of Distribution Algorithms for the Firefighter Problem”. Em: *Evolutionary Computation in Combinatorial Optimization: 17th European Conference, EvoCOP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings*. Ed. por Bin Hu e Manuel López-Ibáñez. Cham: Springer, 2017, pp. 108–123. ISBN: 978-3-319-55453-2. DOI: 10.1007/978-3-319-55453-2_8.
- [52] George L Nemhauser e Laurence A Wolsey. “Integer programming and combinatorial optimization”. Em: *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin* 20 (1988), pp. 8–12.

- [53] Mauricio G.C. Resende e Celso C. Ribeiro. “Metaheuristics: Progress as Real Problem Solvers”. Em: Boston, MA: Springer, 2005. Cap. GRASP with Path-Relinking: Recent Advances and Applications, pp. 29–63. ISBN: 978-0-387-25383-1. DOI: 10.1007/0-387-25383-1_2.
- [54] Mauricio GC Resende e Celso C Ribeiro. *Optimization by GRASP*. Springer, 27 de out. de 2016. ISBN: 978-1-4939-6528-1.
- [55] Neil Robertson e Paul D Seymour. “Graph minors. I. Excluding a forest”. Em: *Journal of Combinatorial Theory, Series B* 35.1 (1983), pp. 39–61. DOI: 10.1016/0095-8956(83)90079-5.
- [56] Ganesh Ram Santhanam et al. “Verifying Intervention Policies to Counter Infection Propagation over Networks: A Model Checking Approach.” Em: *AAAI*. 2011, pp. 7–11.
- [57] Allan E. Scott, Ulrike Stege e Norbert Zeh. “Politician’s firefighting”. Em: *Algorithms and Computation*. Kolkata, India: Springer, 2006, pp. 608–617. DOI: 10.1007/11940128_61.
- [58] Weifan Wang, Stephen Finbow e Ping Wang. “The surviving rate of an infected network”. Em: *Theoretical Computer Science* 411.40-42 (2010), pp. 3651–3660. DOI: 10.1016/j.tcs.2010.06.009.