MARCOS JOSÉ PIMENTEL SALLES

IMPLEMENTAÇÃO DE UM MODELO MATEMÁTICO PARA O PROBLEMA DE PARTICIONAMENTO DE ARESTAS EM DISTRITOS EULERIANOS

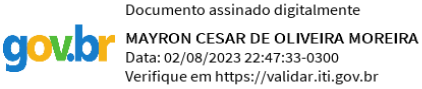
LAVRAS – MG

2023

MARCOS JOSÉ PIMENTEL SALLES

IMPLEMENTAÇÃO DE UM MODELO MATEMÁTICO PARA O PROBLEMA DE PARTICIONAMENTO DE ARESTAS EM DISTRITOS EULERIANOS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências para a obtenção do título de Bacharel em Ciência da Computação



Prof. DSc. Mayron César de Oliveira Moreira

Orientador

LAVRAS – MG

2023

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Salles, Marcos José Pimentel

Implementação de um modelo matemático para o problema de particionamento de arestas em distritos Eulerianos / Marcos José Pimentel Salles. 2^a ed. rev., atual. e ampl. – Lavras : UFLA, 2023.

52 p. : il.

TCC–Universidade Federal de Lavras, 2023.

Orientador: Prof. DSc. Mayron César de Oliveira Moreira.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

MARCOS JOSÉ PIMENTEL SALLES

IMPLEMENTAÇÃO DE UM MODELO MATEMÁTICO PARA O PROBLEMA DE PARTICIONAMENTO DE ARESTAS EM DISTRITOS EULERIANOS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências para a obtenção do título de Bacharel em Ciência da Computação

APROVADA em 30 de Julho de 2023.

Prof. DSc. Mayron César de Oliveira Moreira	UFLA
Prof. DSc. Júlio César Alves	UFLA
Prof. DSc. Vinícius Vitor dos Santos Dias	UFLA

Prof. DSc. Mayron César de Oliveira Moreira
Orientador

LAVRAS – MG
2023

Dedico este trabalho a todos os que me ajudaram ao longo desta caminhada.

AGRADECIMENTOS

Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso.

Toda ciência seria supérflua se a aparência externa e a essência das coisas coincidissem diretamente.
(Marx)

RESUMO

O trabalho considera um modelo de grafos planar, não-direcionado, ponderado e conexo, e aborda o problema de parcelamento de arestas. Trata-se de um problema presente em contextos como entrega de correspondências, coleta de lixo e recolhimento de neve. O objetivo consiste em gerar k subgrafos induzidos (ou distritos) que sejam equilibrados quanto à distribuição de demanda, representada pelo peso das arestas. Além disso, a quantidade de vértices de grau ímpar em cada partição é limitada, tornando cada subgrafo gerado o mais Euleriano possível. Tomando um vértice referência (ou vértice que indica um "depósito", dependendo da aplicação) para cada subgrafo, a função de otimização trabalha a compacidade dos subgrafos, minimizando o somatório da distância de cada aresta designada a um subgrafo para seu respectivo vértice referência. A literatura desse problema já possui um algoritmo exato, baseado na geração iterativa de cortes de conexidade. No entanto, tal implementação não encontra-se disposível para acesso. Diante disso, propõe-se, neste trabalho, a codificação de tal algoritmo, utilizando *frameworks* modernos tal como o GUROBIPY, implementando a inserção dos cortes de forma dinâmica através de funções *callback*. Os resultados computacionais atestam a eficácia da formulação implementada.

Palavras-chave: Parcelamento de arestas. Distritos Eulerianos. Modelo matemático. Gurobi. Pesquisa operacional.

ABSTRACT

The paper discusses a model of a planar, undirected, weighted, and connected graph. It addresses the issue of edge partitioning, which arises in various contexts like mail delivery, waste collection, and snow removal. The objective is to generate k induced subgraphs, also known as districts, that achieve a balanced distribution of demand represented by the edge weights. Additionally, there is a constraint on the number of odd-degree vertices in each partition, aiming to maximize the Eulerian property of the subgraphs. To accomplish this, each subgraph is associated with a reference vertex, which can be interpreted as a "depot"depending on the application. The optimization function evaluates the compactness of the subgraphs by minimizing the total distance between each edge assigned to a subgraph and its corresponding reference vertex. Although an exact algorithm for this problem exists in the literature, which utilizes iterative generation of connectivity cuts, it is currently unavailable for access. Thus, we propose in this study the implementation of the algorithm using modern frameworks such as GUROBIPY. The implementation includes the dynamic insertion of cuts through callback functions. Our computational experiments demonstrate the effectiveness of the formulated approach.

Keywords: Edge partitioning. Eulerian Districts. Mathematical model. Gurobi. Operational research.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

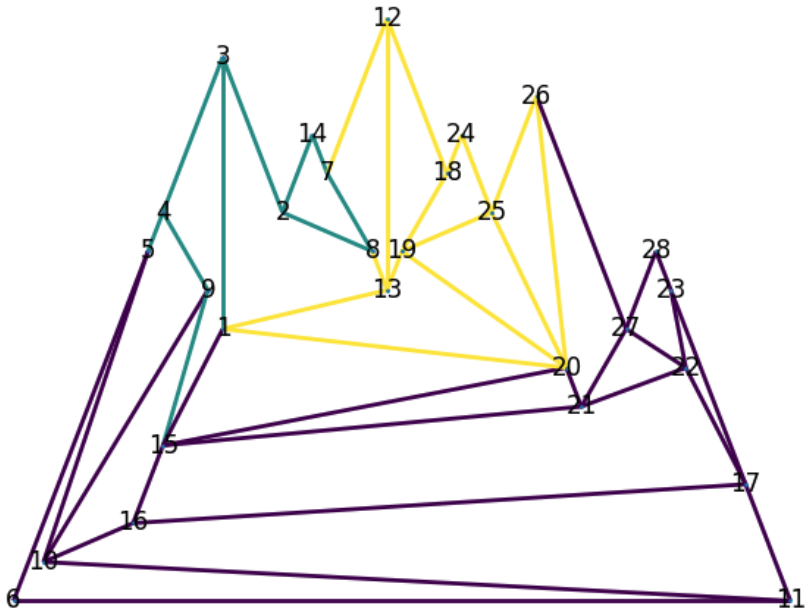
O problema de particionamento de arestas é comumente modelado através de grafos ponderados, conexos, não-direcionados e planares. Trata-se de um problema presente em contextos como entrega de correspondências (BODIN; LEVY, 1991), coleta de lixo (MOURãO; NUNES; PRINS, 2009; GOMES; PALHANO; REIS, 2023) e recolhimento de neve (LABELLE; LANGEVIN; CAMPBELL, 2002). As arestas ou ruas possuem um valor de demanda pré-determinado.

O *Problema de Parcelamento de Arestas em Distritos Eulerianos* (PPADE) pode ser definido da seguinte forma. Considera-se um conjunto de k depósitos, um para cada partição (ou distrito), em que o objetivo consiste em gerar distritos conexos que minimizem a soma das distâncias de suas arestas para seus respectivos depósitos. Neste estudo, determina-se que uma solução viável deve ser equilibrada quanto à demanda distribuída em seus distritos e a quantidade de vértices ímpares resultante do particionamento de arestas. Essa última característica faz com que cada distrito gerado seja o mais Euleriano possível, reduzindo caminhamentos desnecessários no grafo (popularmente conhecidos como *deadhead*). A Figura 1.1 representa um exemplo de solução viável para o problema de distritamento para esse grafo. Os dados completos da instância podem ser obtidos no link a seguir: <https://github.com/Marcos-Pimentel/TCC>. A Tabela 1.1 mostra que a solução apresentada pela figura previamente mencionada respeita as tolerâncias de perda de paridade ($\tau_2 = 0,1$) e de desvio da média das demandas estabelecidas ($\tau_1 = 0,1$). Esses parâmetros são abordados de forma mais profunda na seção 2.1.1.

Tabela 1.1 – Lpr-a-01-3C-7B ($\tau_1 = 0,1 \quad \tau_2 = 0,1$)

Disparidade de demanda	**Coeficiente de imparidade**	***Deadhead***
0.0920	0.0357	498

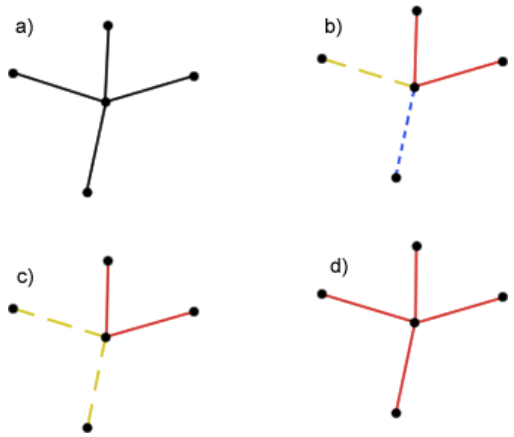
Figura 1.1 – Lpr-a-01-3C-7B ($\tau_1 = 0.10 \quad \tau_2 = 0.05$)



A Figura 1.2 mostra um exemplo de possíveis partições de um vértice de grau 4. Dentre elas, a partição *a)* representa o grafo original, a partição *b)* indica como é um particionamento indesejado e as partições *c)* e *d)* mostram exemplos de partições desejadas. Uma partição desejada é caracterizada por manter a paridade de um vértice, contribuindo assim para a euleridade dos distritos formados. De forma análoga, uma partição não desejada é uma partição que tira a paridade de um vértice par, ou que divide um vértice ímpar de modo a mais de um distrito receber uma quantidade ímpar de arestas. Isso é indesejado, pois torna os distritos menos eulerianos.

A literatura sobre o problema, levando em conta essa última década, apresenta 3 principais trabalhos, sendo eles (BUTSCH; KALCSICS; LAPORTE, 2014), (ASSIS; FRANCA; USBERTI, 2014) e (GARCÍA-AYALA et al., 2016). O primeiro trabalho elabora uma heurística construtiva com busca local para a cons-

Figura 1.2 – Exemplos de partições de um vértice de grau 4



trução dos distritos. Ele apresenta uma função objetivo que leva em conta criar distritos com demandas balanceadas entre si, que sejam compactos e que se gaste pouco tempo revisitando arestas. O segundo aborda o problema da perspectiva de uma empresa energética de São Paulo, e portanto desenvolve uma metaheurística baseada em GRASP (Greedy Randomized Adaptive Search Procedure), que priorizam o balanceamento das demandas, a criação de distritos contínuos e a similaridade com as rotas originais. Por fim, o terceiro trabalho tem como objetivo criar distritos o mais semelhantes o possível a um grafo Euleriano. Para tal, utiliza-se de uma modelagem matemática inteira-mista que prioriza a criação de distritos compactos, com a garantia que os mesmos continuem conexos e mantenha na medida do possível a paridade dos vértices do grafo. As restrições que garantem a conectividade dos distritos são relaxadas na implementação do modelo, uma vez que apresentam complexidade exponencial. Portanto, sempre que uma solução encontra distrito desconexo, são adicionadas restrições para garantir que a resposta seja considerada inválida. Além dos trabalhos que se destacam na literatura, há também um trabalho recente realizado por (ESPíNDOLA, 2022) que utiliza-se de uma heurística construtiva baseada no algoritmo de busca em largura para a criação dos distritos, a partir do depósito. Essa heurística, portanto garante a criação

de distritos conexos, e a escolha das arestas durante a expansão ocorre a partir de um critério de qualidade previamente escolhido.

Ao que consta, o trabalho de (GARCÍA-AYALA et al., 2016) é o único que propõe um algoritmo exato PPADE. Os autores desenvolvem uma formulação matemática e para sua resolução, relaxam as restrições de conexidade de distritos. Caso os distritos gerados sejam conexos, o algoritmo para e a solução ótima é retornada. Caso contrário, realiza-se uma busca em grafos para detectar as componentes desconexas e ligá-las através de cortes iterativos à formulação atual. Tomando 142 instâncias que variam de 28 a 401 vértices, o algoritmo proposto resolve aproximadamente 98% das instâncias na otimalidade.

1.0.1 Objetivos

O trabalho de (GARCÍA-AYALA et al., 2016) é o estado da arte para o PPADE. No entanto, sua implementação encontra-se inacessível a pesquisadores. Os autores do trabalho não disponibilizam o código abertamente, e não se manifestaram favoráveis à sua liberação, mesmo após solicitações formais por e-mail. Assim, considerando os benefícios que o acesso aos artefatos científicos podem trazer à continuação das pesquisas, este trabalho possui dois objetivos:

- Implementar e disponibilizar o modelo de (GARCÍA-AYALA et al., 2016) de forma aberta, utilizando o *framework* GUROBIPY, uma das referências para codificação de modelos matemáticos (Gurobi Optimization, LLC, 2023).

- Obtenção de limites superiores de outros problemas teste, propostos no trabalho de (MOREIRA, 2023), em decorrência da falta de acesso às instâncias utilizadas no artigo de (GARCÍA-AYALA et al., 2016).

1.0.2 Contribuições

Este trabalho contribui com a literatura ao aprofundar uma abordagem do problema de distritamento. Sendo essa abordagem a de criação de distritos Eulerianos, focando em garantir a formação de distritos com pequena distância de *deadhead*. Para tal, foi utilizado um software comercial que fornece ferramentas para tornar a resolução do problema mais eficiente. Uma das principais ferramentas fornecidas para tal tarefa é o *callback*, que permite a execução de um modelo mais simples e a adição de novas restrições em situações adequadas, sem a necessidade de reiniciar a execução do modelo.

Outra contribuição é a execução de uma extensa bateria de testes, e a disponibilização desses casos de teste, assim como dos resultados e do algoritmo implementado. Isso permite que esse problema possa ser mais facilmente analisado em estudos futuros.

1.0.3 Estrutura do Trabalho

O restante do trabalho apresenta a seguinte estrutura. A Seção 2 apresenta a modelagem matemática proposta por (GARCÍA-AYALA et al., 2016) , e detalhes da implementação da formulação através do *framework* GUROBIPY. A Seção 3 reporta os testes realizados e os analisa a partir de um conjunto de métricas. Por fim, a Seção 4 contém a conclusão do trabalho e propostas de pesquisas futuras.

2 DESENVOLVIMENTO

Este capítulo apresenta o desenvolvimento da modelagem matemática, que foi inicialmente proposta por (GARCÍA-AYALA et al., 2016), e o algoritmo em Python que adapta essa modelagem para a biblioteca do GUROBIPY. A Seção 2.1 apresenta todos os passos para formalizar o modelo matemático do PPADE. Detalhes referentes à implementação realizada no *framework* GUROBIPY são descritos na Seção 2.1.4.

2.1 Modelo Matemático

A Seção 2.1.1 define todos os parâmetros necessários para a formalização do problema. Na sequência, a Seção 2.1.2 apresenta as variáveis de decisão do modelo, enquanto a Seção 2.1.3, mostra a versão completa do modelo matemático.

2.1.1 Parâmetros

Seja $G = (V,E)$ um grafo ponderado, conexo, não-direcionado e planar, em que V e o conjunto de vértices e E o conjunto de arestas. Cada aresta $e = (u,v) \in E$ representa uma rua conectando os vértices $u \in V$ e $v \in V$. Cada aresta $e \in E$ possui comprimento l_e e uma demanda d_e.

Considere $P \subseteq V$ o conjunto dos vértices depósitos. Cada depósito $p \in P$ será o responsável por abastecer um determinado distrito, e portanto, esse distrito pode ser referido como distrito p. Para melhor modelar o problema, também é importante ter acesso à vizinhança de arestas que incidem no nó, ou que são vizinhas às arestas analisadas no momento. Para tanto, seja $\delta(u) \subseteq E$, $\forall u \in V$, o conjunto das arestas que incidem no vértice u, e $\sigma(S) \subseteq E$, $\forall S \subset E$, o conjunto de arestas que apresentam uma extremidade em $V(S)$ e outra extremidade em $V \setminus V(S)$, em que $V(S)$ é o conjunto de vértices associados a arestas em S.

Define-se $b_{pe} = \min\{\alpha_{pu}, \alpha_{pv}\}$, $\forall p \in P$, $\forall e = (u,v) \in E$, como a menor distância possível entre o depósito p e a aresta e, tal que α_{pu} (α_{pv}) é o valor do

caminho mais curto de p a u (v). A média de demanda por distrito é dada por $D = \sum_{e \in E} d_e / |P|$. As tolerâncias adotadas para valores de demanda e paridade de cada distrito em relação à média são dadas por $\tau_1 \in [0,1]$ e $\tau_2 \in [0,1]$, respectivamente. A perda de paridade necessita da definição do conjunto $V^e \subset V$, que indica o conjunto de vértices de grau par em G. Por fim, a constante M que é um valor suficientemente grande para garantir a redundância de algumas restrições em casos específicos.

2.1.2 Variáveis

A formulação matemática do PPADE trabalha com seis conjuntos de variáveis. Seja x_{pe}, $\forall p \in P$, $\forall e = (u,v) \in E$, é uma variável binária igual a 1 se a aresta e está associada ao distrito p, e w_{pu}, $\forall p \in P$, $\forall u \in V$, é uma variável binária igual a 1 se o vértice u está associado distrito p. O controle de paridade dos vértices de cada distrito é feito por meio de três grupos de variáves.

A variável z^0_{up}, $\forall u \in V$, $\forall p \in P$, é uma variável binária igual a 1 (0) se o vértice $u \in V$ tem grau ímpar (par) quando associado às arestas do distrito p. Considere z_{up}, $\forall u \in V$, $\forall p \in P$, uma variável natural auxiliar que tem por função garantir que z^0_{up} receba o valor correto em cada caso. Seja r_u, $\forall u \in V$, uma variável binária igual a 1 se o vértice u perde ou não paridade.

2.1.3 Formulação

Uma adaptação do modelo matemático proposto por (GARCÍA-AYALA et al., 2016) é apresentada abaixo.

$$\text{Min} \quad g(x) = \sum_{p \in P} \sum_{e \in E} b_{pe} x_{pe} \tag{2.1}$$

sujeito a:

$$\sum_{p\in P} x_{pe} = 1 \qquad \forall e \in E \tag{2.2}$$

$$\sum_{s\in\sigma(S)} x_{ps} - \sum_{s\in S} x_{ps} \geq x_{pe} - |S| \qquad \forall p \in P, \forall e \in E, \forall S \subset E\backslash\sigma(e) \tag{2.3}$$

$$\bar{D}(1-\tau_1) \leq \sum_{e\in E} d_e x_{pe} \qquad \forall p \in P \tag{2.4}$$

$$\sum_{e\in E} d_e x_{pe} \leq \bar{D}(1+\tau_1) \qquad \forall p \in P \tag{2.5}$$

$$\sum_{e\in\delta(u)} x_{pe} \leq M w_{pu} \qquad \forall p \in P, \forall u \in V \tag{2.6}$$

$$w_{pu} \leq \sum_{e\in\delta(u)} x_{pe} \qquad \forall p \in P, \forall u \in V \tag{2.7}$$

$$\sum_{e\in\delta(u)} x_{pe} = 2z_{up} + z^0_{up} \qquad \forall p \in P, \forall u \in V \tag{2.8}$$

$$r_u \leq \sum_{p\in P} z^0_{up} \qquad \forall u \in V^e \tag{2.9}$$

$$|P| r_u \geq \sum_{p\in P} z^0_{up} \qquad \forall u \in V^e \tag{2.10}$$

$$r_u \leq \sum_{p\in P} z^0_{up} - 1 \qquad \forall u \in V^o \tag{2.11}$$

$$|P| r_u \geq \sum_{p\in P} z^0_{up} - 1 \qquad \forall u \in V^o \tag{2.12}$$

$$\frac{1}{|V|} \sum_{u\in V} r_u \leq \tau_2 \tag{2.13}$$

$$w_{pu} \in \{0,1\} \qquad \forall p \in P, \forall u \in V \tag{2.14}$$

$$x_{pe} \in \{0,1\} \qquad \forall p \in P, \forall e \in E \tag{2.15}$$

$$z^0_{up} \in \{0,1\} \qquad \forall u \in V, \forall p \in P \tag{2.16}$$

$$r_u \in \{0,1\} \qquad \forall u \in V \tag{2.17}$$

$$z_{up} \in \mathbb{N} \cup \{0\} \qquad \forall p \in P, \forall u \in V. \tag{2.18}$$

A função objetivo (2.1) busca minimizar o somatório das distâncias de cada aresta até o depósito do distrito que ela foi designada, desse modo aumentando a compacidade do distrito. As restrições (2.2) garantem a designação de cada aresta para obrigatoriamente um único distrito. As restrições (2.3) fazem com que os distritos sejam contínuos, uma vez que força que cada aresta para ser designada ao distrito p, deve apresentar um caminho até o depósito p que passe apenas por arestas que foram designadas ao mesmo distrito. Para uma explicação mais aprofundada, pode-se observar como são implementados os cortes, explicados na subseção 2.1.4, e considerar que os cortes são adicionados para todas as combinações de arestas possíveis, de modo a garantir conectividade.

As restrições (2.4) e (2.5) delimitam o limite inferior e superior, respectivamente, da demanda permitida para cada distrito. As restrições (2.6) e (2.7) garantem respectivamente que se uma aresta $e = (u, v)$ foi designada ao distrito p, então u e v estão ambos associados ao depósito p; e se um vértice u foi associado a um distrito, então uma das arestas que u está associado foi designada para o distrito p. As restrições (2.8) definem a paridade do vértice para cada um dos distritos. As restrições (2.9) (2.12) definem a perda de paridade de um vértice $u \in V$, de modo que garantem que caso tenha de fato perdido a paridade, atribuindo a r_u o valor adequado. A restrição (2.13) serve para controlar a perda de paridade, garantindo que não ultrapasse o limite estipulado pela variável de tolerância τ_2. A perda de paridade é calculada dividindo a quantidade de nós que perderam a paridade pela quantidade total de vértices do grafo. Por exemplo, se apenas 3 vértices pares no grafo original passaram a ser considerados vértices ímpares em algum distrito, e o total de vértices do grafo original são 150, então apenas 2% dos vértices perderam a paridade nesse caso. As restrições (2.14) (2.17) delimitam que as variáveis em questão são binárias. Por fim, a restrição (2.18) delimita o domínio da variável para o conjunto dos números naturais, juntamente com o número 0.

2.1.4 Detalhes de Implementação

Para a criação e execução da modelagem do problema, foi utilizado o *software* comercial Gurobi, mais especificamente o *framework* GUROBIPY, para a linguagem Python. O GUROBI é um *software* desenvolvido para resolver problemas de programação de inteiros mista (MIP). Ou seja, ele resolve problemas que possam ser modelados utilizando apenas um conjunto de equações ou inequações com variáveis não exponenciais. O programa é disponibilizado de forma gratuita para instituições de ensino, e apresenta uma licensa paga para empresas.

Para o desenvolvimento e execução do algoritmo utilizou-se bibliotecas que serviram para auxiliar em diferentes partes do processo. Para lidar com a leitura dos casos de teste e a escrita posterior dos resultados, foi necessária a biblioteca *json*, pois foi esse o formato escolhido para a entrada e saída de dados. Ainda a respeito do processo de leitura e escrita, utilizou-se a biblioteca *walk*, do pacote *os*, que armazena os nomes dos arquivos de entrada em uma lista para a leitura posterior dos mesmos. A última biblioteca foi a *re*, responsável por fazer um *regex* com os dados da entrada para poder executar, posteriormente, o algoritmo de *Floyd-Warshall*(CORMEN et al., 2022). *Regex* é uma abreviação para expressão regular, que no caso, consiste em analisar o padrão de uma variável de texto e decompor ela em variáveis úteis, no caso deste trabalho, montar o grafo no formato necessário para executar o algoritmo de *Floyd-Warshall*.

Em seguida, utilizou-se a biblioteca *networkx* armazenamento do grafo em memória, e juntamente com *matplotlib.pyplot*, permitiram o desenho dos grafos para a saída. Já a biblioteca *defaultdict*, do pacote *collections*, auxilia na criação de dicionários já inicializados com um valor padrão para cada chave. Após a importação das bibliotecas, tem-se as funções auxiliares do programa, sendo uma delas o algoritmo de *Floyd-Warshall*, reponsável por fornecer o caminho mais curto entre cada par de vértices de um grafo. Para que o modelo do GUROBI possa funcionar, temos primeiramente que declarar o modelo e iniciar as variáveis

do mesmo. As variáveis desse problema são criadas a partir de uma lista, que o GUROBIPY converte em chaves de um dicionário.

O primeiro grupo de variáveis a ser declarado chama-se x, definido como um dicionário no qual as chaves são todas os possíveis pares formado por depósito e uma aresta (p,e), $\forall p \in P, \forall e \in E$. O segundo conjunto de variáveis do modelo é denominado w, representando a relação entre depósitos e vértices, sendo as chaves todos os pares possíveis entre depósitos e vértices (p,u), $\forall p \in P, \forall u \in V$.

Em seguida, declara-se o conjunto de variáveis z, único de valor natural, que auxilia no processo de definir a paridade de um vértice quando relacionado a um determinado depósito (u,p), $\forall u \in V, \forall p \in P$. A paridade é indicada pelo conjunto de variáveis binárias z_0, que assim como z, relaciona um par de vértice e depósito com um valor. A última sequência de variáveis, r_u, faz uma relação de cada vértice $u \in V$ com um valor binário, que indica se o vértice perdeu ou não sua paridade.

As restrições (2.3) foram relaxadas, visto que a enumeração de todos os subconjuntos de vértices S é inviável na resolução para as instâncias testadas (veja Seção 3.1). Para contornar esse problema, primeiramente é executado a modelagem sem a restrição, e após retornar um resultado, este é analisado para verificar que a resposta apresenta apenas distritos conexos. Caso algum distrito seja desconexo, são adicionadas restrições de corte na modelagem que garantem que pelo menos alguma aresta vizinha do conjunto de arestas desconexas passe a ser designada ao distrito que esse grupo de arestas for designada, e a modelagem é executada novamente.

Detalhando mais essa abordagem, primeiramente foi criado uma função de *callback*, de modo que toda vez que uma solução é encontrada, essa função a analisa para assegurar que todos os distritos formados são conexos. Caso positivo, a resposta está pronta e termina a execução do modelo. Caso algum distrito não seja conexo, a primeira coisa a se fazer é encontrar as arestas que não estão conec-

tadas ao depósito que foram designadas. Para encontrar essas arestas, analisamos cada depósito de forma individual, e monta-se um grafo que representa a resposta encontrada pelo modelo para representar esse distrito. Com o grafo construído, verifica-se quais arestas podem ser alcançadas a partir do depósito. Para cada aresta que não foi alcançada, ela é agrupada às arestas que está conectada, uma vez que um grupo de arestas pode estar conexa entre si, mas não apresentam nenhuma conexão ao depósito. Finalmente, para cada grupo de arestas desconexas, são adicionadas restrições que garantem que alguma aresta vizinha a esse grupo tenha que ser designada ao mesmo depósito, e o modelo é executado novamente. As restrições adicionadas são as seguintes:

$$\sum_{s \in \sigma(S)} x_{ps} - \sum_{e \in S} x_{pe} >= 1 - |S| \tag{2.19}$$

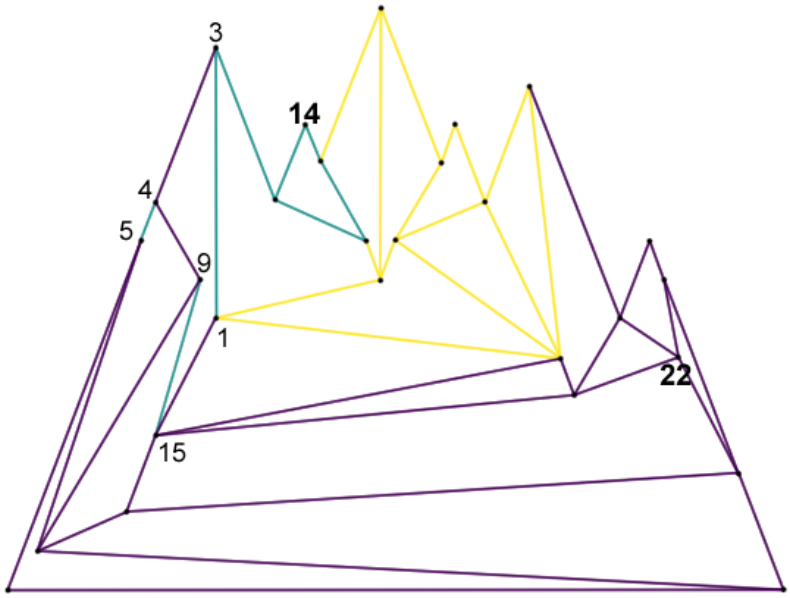
de modo que S representa o grupo de arestas desconexas. Para melhor compreensão do fluxo de execução do algoritmo de adição de cortes ao modelo, apresenta-se o Algoritmo 1.

Pode-se entender melhor o funcionamento desses cortes, utiliza-se o exemplo da figura (2.1). No caso dessa solução, os depósitos que nos interessam se encontram nos vértices 14 e 22. Nesse caso, o distrito 14 não está continuo, portanto a solução é inválida, pois as restrições (2.3) indicam que para a aresta $(4,5)$ ser designada ao distrito 14, uma das arestas vizinhas devem ser designada ao mesmo distrito. Isso também é válido para a aresta desconexa $(9,15)$. A Figura 2.2 mostra uma outra solução, dessa vez obedecendo as restrições explicadas anteriormente. Porém o distrito 14 continua desconexo, o que indica que a solução ainda não é factível. As restrições (2.3) forçam para que o conjunto de arestas desconexas $((4,5),(5,9),(9,15))$ seja designado ao distrito 14, uma das arestas que vizinha esse grupo deve também ser designada ao mesmo depósito. Por fim chegamos à figura de exemplo 2.3, que apresenta uma solução viável, uma vez que todos os distritos estão conexos.

Algoritmo 1: Adição dos Cortes

1: Seja P o conjunto de depósitos de uma solução encontrada pelo GUROBIPY
2: **for** $p \in P$ **do**
3: Seja $V_p \subset V$ o conjunto de vértices que fazem parte do grafo do distrito p
4: Seja $E_p \subset E$ o conjunto de arestas que fazem parte do grafo do distrito p
5: Seja $i_p \in V_p$ o vértice que representa o depósito do distrito p
6: Seja $E_v \leftarrow$ visitReachable(i_p) ▷ Retorna todas as arestas que podem ser visitadas saindo do depósito
7: Seja $U \leftarrow \varnothing$ o conjunto que armazena grupos de arestas que não fazem parte das arestas que podem ser visitadas saindo do depósito
8: **while** $E_p \backslash E_v \neq \varnothing$ **do**
9: Seja $i_r \in E_p \backslash E_v$ ▷ Pega um vértice arbitrário que não foi marcado ainda
10: Seja $B \leftarrow$ visitReachable(i_p)
11: **for** $e \in B$ **do**
12: $E_v = E_v \cup \{e\}$ ▷ Marca todas as arestas contidas em B
13: **end for**
14: $U = U \cup \{B\}$ ▷ insere em U o conjunto B como um único elemento
15: **end while**
16: **for** $S \in U$ **do** Acrescente ao modelo as restrições $\sum_{s \in \sigma(S)} x_{ps} - \sum_{e \in S} x_{pe} \geq 1 - |S|$
17: **end for**
18: **end for**

Figura 2.1 – Lpr-a-01-3C-7B ($\tau_1 = 0.10 \quad \tau_2 = 0.05$)



O GUROBI trabalha com adaptações do algoritmo de *Branch and Bound* e o uso de *callbacks* faz com que nós que representam soluções infactíveis acionem a criação de cortes na resolução atual do modelo matemático, sem que tenha que reiniciar todo o algoritmo. Isso permite que no lugar de começar uma nova execução do modelo, seja apenas adicionado um corte na árvore do *Branch and Bound*. Uma implementação de todos os cortes desejados de forma prévia à execução do modelo não é uma opção viável, pois isso demandaria um tempo exponencial para ser realizado. Portanto, os *callbacks* consistem em uma alternativa para a rsolução eficiente de abordagens baseadas em geração de cortes iterativos.

Figura 2.2 – Lpr-a-01-3C-7B ($\tau_1 = 0.10 \quad \tau_2 = 0.05$)

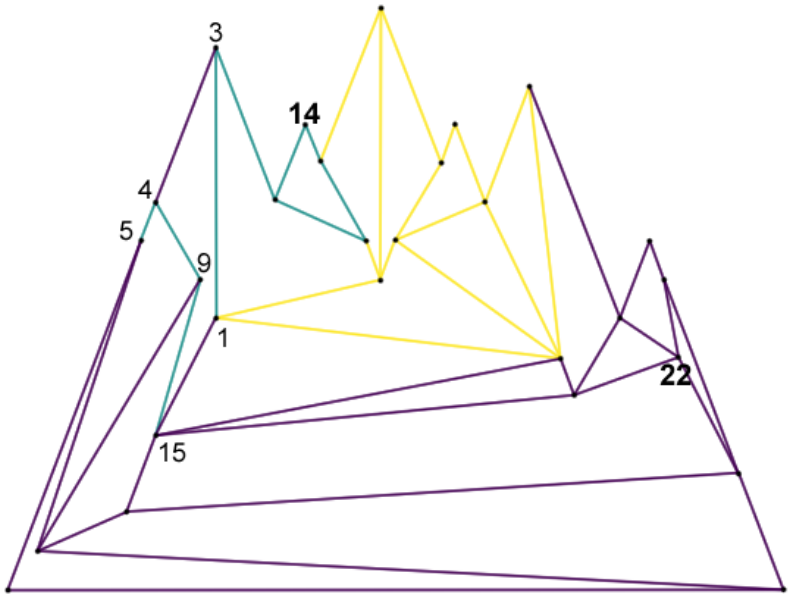
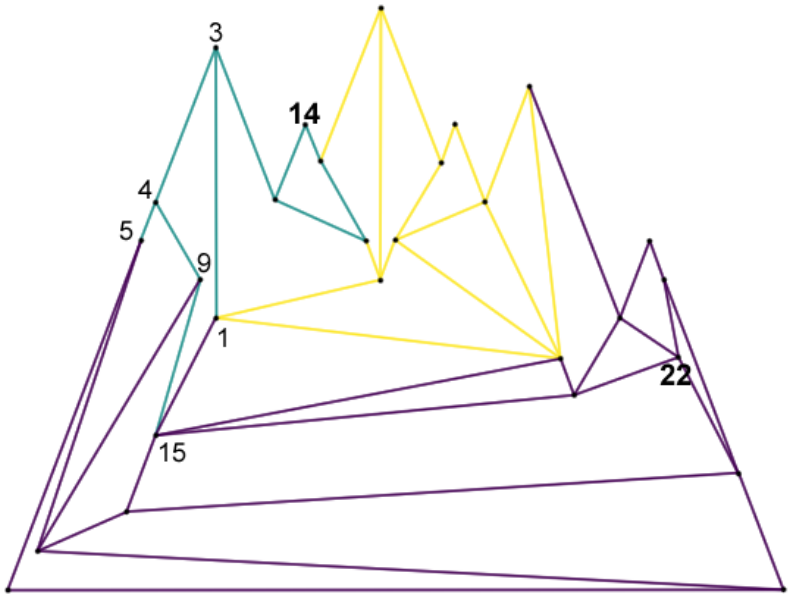


Figura 2.3 – Lpr-a-01-3C-7B ($\tau_1 = 0.10 \quad \tau_2 = 0.05$)



3 TESTES COMPUTACIONAIS

Este capítulo apresenta os experimentos computacionais para a validação do desempenho do algoritmo implementado. A Seção 3.1 descreve as instâncias utilizadas nos testes. Na sequência, a Seção 3.2 reporta uma análise dos resultados.

3.1 Benchmark

As instâncias utilizadas neste trabalho foram adaptadas a partir da base de dados descrita por (GARCÍA-AYALA et al., 2016). Nessa base original, responsável por organizar como seria o nome das instâncias, como por exemplo a instância 'Lpr-b-04-4C-5A', consta 16 mapas que apresentam nomes seguindo o seguinte padrão: 'Lpr-' seguido de 'a-' ou 'b-'. Em seguida, há um número que vai de '01-' a '05-' que indica de certo modo o tamanho do grafo e por fim, um número que indica a quantidade de depósitos presentes no grafo, acompanhado de 'C-'. As características desses 16 mapas originais podem ser observadas na tabela 3.1.

Tabela 3.1 – Características dos mapas originais

Grupo de Instâncias	N° vértices	**N° arestas**	**Densidade**	**Proporção de vértice par**
Lpr-a-01-2C	28	47	0.1243	0.5
Lpr-a-01-3C	28	47	0.1243	0.5
Lpr-a-02-2C	53	86	0.0624	0.3584
Lpr-a-03-2C	146	226	0.0213	0.4931
Lpr-a-03-3C	146	226	0.0213	0.4931
Lpr-a-03-4C	146	226	0.0213	0.4931
Lpr-a-03-5C	146	226	0.0213	0.4931
Lpr-a-04-4C	195	323	0.0170	0.4871
Lpr-a-04-5C	195	323	0.0170	0.4871
Lpr-a-05-4C	321	535	0.0104	0.4953
Lpr-b-01-2C	28	42	0.1111	0.3571
Lpr-b-01-3C	28	42	0.1111	0.3571
Lpr-b-02-2C	53	82	0.0595	0.3962
Lpr-b-03-4C	162	266	0.0203	0.5185
Lpr-b-04-4C	248	457	0.0149	0.5564
Lpr-b-05-6C	401	727	0.0090	0.5411

Utilizando esses 16 grafos como base, foram utilizados dois métodos para variar a distância e demanda de cada aresta do grafo. O primeiro desses métodos é responsável pela letra 'A' no final do nome da instância, e a maneira que ele altera os valores das arestas é de acordo com as seguintes regras: as demandas que constam como 0 não são alteradas; as demais demandas, e a totalidade das distâncias são escolhidas de forma aleatória de dentro de um intervalo que é delimitado pelos valores não nulos do grafo original. O segundo método adiciona a letra 'B' no final do nome das instâncias, o modo que as distâncias e demandas são alteradas de acordo com o padrão a seguir: as distâncias e demandas são reordenadas de forma aleatória, e em seguida, multiplicadas por um valor randômico entre $0,5$ e $1,5$.

Os nós que representam os depósitos foram escolhidos de forma similar ao trabalho de (GLIESCH; RITT; MOREIRA, 2018). Para a escolha do primeiro depósito, basta escolher um nó de forma aleatória. Para os demais depósitos, primeiramente calcula-se a raiz quadrada da quantidade de nós restantes, arredonda para o menor inteiro maior que o valor calculado, e escolhe essa quantidade de nós aleatórios e os armazenam temporariamente como candidatos. Após isso, para cada combinação possível de candidato e depósito já escolhido, calcula-se o caminho mínimo. Depois disso, para cada candidato, soma os caminhos mínimos referentes a ele, e o candidato que apesentar a maior soma é escolhido para se tornar um depósito. Esse processo é então repetido até que todos os depósitos sejam escolhidos. No total, foram elaborados 320 casos de teste, que podem ser encontrados em (MOREIRA et al., 2023). Para os testes, escolheu-se os seguintes valores de tolerâncias: $(\tau_1, \tau_2) \in \{(0,1;0,05), (0,1;0,1), (0,2;0,05)\}$. Tais valores foram selecionados por indicarem maior variabilidade dos resultados nas instâncias testadas por (GARCÍA-AYALA et al., 2016). Outro motivo para a escolha de tais valores é pela característica de não permitirem que o algoritmo proponha respostas que possam apresentar uma grande disparidade nas demandas dos distritos ou

até mesmo distritos pouco eulerianos, o que causaria um aumento da distância de *deadhead*.

Os testes foram realizados em um computador Intel(R) Core(TM) i7-7700 CPU 3.60GHz e 16GB RAM. Foi utilizada a versão 3.8.10 do Python, e a versão 10.0.1 do Gurobi Optimizer. Foi estipulado 3.600 segundos como tempo máximo permitido na execução de cada instância. Na execução do Gurobi foram utilizadas 8 threads.

3.2 Resultados

O primeiro conjunto de testes considerou $\tau_1 = 0,1$ e $\tau_2 = 0,05$. Constatou-se que em 72.18% (231 instâncias), o algoritmo exato encontrou solução factível, sendo que apenas 12 terminaram a execução por ter atingido o limite de tempo. Em $94,80\%$ das instâncias que foi encontrada uma solução factível, o algoritmo exato encontrou a solução ótima. Em outras palavras, $68,43\%$ de todas as instâncias executadas apresentaram resultado ótimo. A média de tempo de execução dos casos que terminaram por encontrar solução ótima foi de $102,91$ segundos. Considerando todos os casos, a média de tempo é de $1.210,54$ segundos. O *gap* de otimalidade médio do *solver* para as soluções factíveis que não foram provada a otimalidade foi de $0,27\%$. Esse *gap* representa a diferença entre o *upper bound* e o *lower bound* da solução encontrada. Quando esse valor é nulo, o programa identifica a solução encontrada como ótima, desse modo, valores não nulos para o *gap* aparecem em casos onde foi encontrada uma solução factível, porém a execução foi terminada por ter atingido o limite de tempo, e portanto, não garantiu ainda a otimalidade da resposta. Em média, $3,97\%$ dos vértices tiveram perda de paridade, o que respeita a variável de tolerância $\tau_2 = 0,05$ com um certo grau de folga. A Tabela 3.2 apresenta detalhes da execução desse conjunto de testes. A primeira coluna da tabela indica o mapa original do qual as instâncias do grupo foram baseados, cada grupo representam 20 casos de teste; a segunda coluna representa

a média do tempo de execução ($\bar{t}(s)$) juntamente com o desvio padrão ($\sigma(t)$); a terceira coluna contém a média do *GAP* do *solver* em porcentagem ($\overline{GAP}(\%)$) e o desvio padrão ($\sigma(GAP)$); em seguida temos a coluna que representa a porcentagem de casos em cada grupo que apresenta resultado factível; a quinta coluna mostra a porcentagem das execuções que retornaram respostas ótimas dentro de cada grupo de instâncias; por fim, a última coluna apresenta tanto a média da distância de deadhead ($\overline{Deadhead}$) quanto o desvio parão dessa distância ($\sigma(Deadhead)$).

O segundo conjunto de testes considera os seguintes valores de tolerância: $\tau_1 = 0,1$ e $\tau_2 = 0,1$. Aproximadamente 69% (222 instâncias) obtiveram resultado factível, sendo que apenas 9 terminaram a execução por ter atingido o limite de tempo. Assim, praticamente 96% das soluções que obtiveram resultado factível foram ótimos. Em comparação com todas as instâncias executadas, em 66.56% delas foram obtidos resultados ótimos. A média de tempo de execução dos casos que terminaram por encontrar solução ótima foi de $124,91$ segundos. Considerando todos os casos, a média de tempo é de $1.290,41$ segundos. O *gap* médio de otimalidade do *solver* para soluções cuja otimalidade foi inconclusiva foi de $0,32\%$. Em média, $6,11\%$ dos vértices tiveram perda de paridade, o que respeita a variável de tolerância $\tau_2 = 0,1$ com um certo grau de folga. Abaixo, a tabela 3.3 contém os resultados dos testes.

O terceiro conjunto de testes considerou tolerâncias iguais a $\tau_1 = 0,2$ e $\tau_2 = 0,05$. Em 76% das instâncias, o algoritmo obteve solução factível, sendo que apenas 15 terminaram a execução por ter atingido o limite de tempo. Logo, 94% (242 instâncias) que obtiveram resultado factível foram resolvidas na otimalidade. Esse valor representa que em 70.93% de todas as instâncias executadas o resultado obtido se encontra na otimalidade. A média de tempo de execução para as instâncias cuja otimalidade não fora provada foi de $72,48$ segundos. Ao considerar todos os casos, a média de tempo é de $1.101,06$ segundos. Já o *gap* de otimalidade médio do *solver* foi de $0,21\%$, considerando as 15 instâncias que atingiram o limite

de tempo sem otimalidade comprovada. Em média, $3,96\%$ dos vértices tiveram perda de paridade. Segue abaixo a tabela 3.4 contendo mais informações sobre a execução.

Para resultados mais detalhados dos testes realizados, segue o link: <https://github.com/Marcos-Pimentel/TCC>. Nesse repositório também estão presentes o código fonte do algoritmo implementado, o conjunto de instâncias utilizados para os testes, arquivos em formato ".json" retornados pelo GUROBIPY e imagens dos grafos antes e após o distritamento.

Os resultados obtidos serviram para mostrar que um valor maior na restrição τ_1 pode resultar em uma maior porcentagem de resultados factíveis dentro do tempo limite estipulado, porém um maior valor em τ_2 acabou por retornar uma menor quantidade de resultados factíveis. Isso pode ser pelo fato de que a varável de tolerância τ_1 está relacionada a restrições mais simples, e portanto da uma maior liberdade para o *solver* resolver o modelo, sem acarretar em um maior tempo de execução do *solver* para encontrar a solução ótima. No entanto, para uma análise mais aprofundada de o porquê desse comportamento distindo entre afrouxar τ_1 e τ_2, seria melhor um outro estudo mais focado nessa questão. Uma outra análise que pode ser feita sobre o algoritmo desenvolvido, é que o aumento na quantidade de depósitos apresenta um impacto negativo maior na quantidade de soluções factíveis encontrada do que o aumento da quantidade de vértices e arestas. Pelo menos no que diz respeito à execução do modelo com limite de tempo de 1 hora por caso de teste.

No artigo de (MOREIRA, 2023) há uma comparação entre esse modelo e a heurística construtiva de (ESPíNDOLA, 2022), nele, conclui-se que a modelagem apresentada neste trabalho retorna resultados melhores, porém a heurística apresenta a vantagem de retornar resultado factível para casos de teste maiores, enquanto que em muitos casos este trabalho não consegue resultado antes de atingir o limite de tempo de 1 hora. Desse modo, a heurística se sobressai em casos

com grafos maiores de entrada, com casos contendo uma quantidade significativa de depósitos (5 ou mais), ou com menor tempo de execução disponível para o algoritmo.

Tabela 3.2 – Resultados computacionais para $\tau_1 = 0,1$ e $\tau_2 = 0,05$.

Grupo de Instâncias	$\bar{t}(s) \pm \sigma(t)$	$\overline{GAP}(\%) \pm \sigma(GAP)$	**Factível**$(\%)$	**Ótima**$(\%)$	$\overline{\textbf{\textit{Deadhead}}} \pm \sigma(\textbf{\textit{Deadhead}})$
Lpr-a-01-2C	0.4 ± 0.6	0.0 ± 0.0	100	100	789.6 ± 127.8
Lpr-a-01-3C	0.9 ± 1.7	0.0 ± 0.0	100	100	733.0 ± 169.3
Lpr-a-02-2C	0.2 ± 0.3	0.0 ± 0.0	100	100	2791.9 ± 388.5
Lpr-a-03-2C	0.4 ± 0.5	0.0 ± 0.0	100	100	10116.0 ± 1811.0
Lpr-a-03-3C	363.9 ± 1078.7	0.0 ± 0.0	100	90	9737.1 ± 2074.0
Lpr-a-03-4C	929.0 ± 951.5	0.0 ± 0.0	85	80	9406.1 ± 2024.0
Lpr-a-03-5C	2830.8 ± 1208.6	0.1 ± 0.0	35	25	10714.4 ± 1908.2
Lpr-a-04-4C	1545.6 ± 974.8	0.0 ± 0.0	65	65	13207.9 ± 2277.7
Lpr-a-04-5C	3076.3 ± 1167.5	0.2 ± 0.0	30	15	13093.5 ± 1636.9
Lpr-a-05-4C	2523.4 ± 1238.7	0.0 ± 0.0	40	35	28322.1 ± 7650.5
Lpr-b-01-2C	0.4 ± 0.5	0.0 ± 0.0	100	100	1058.8 ± 205.7
Lpr-b-01-3C	0.5 ± 0.7	0.0 ± 0.0	100	100	981.2 ± 240.0
Lpr-b-02-2C	0.3 ± 0.4	0.0 ± 0.0	100	100	2884.7 ± 452.9
Lpr-b-03-4C	1887.3 ± 1350.8	0.0 ± 0.0	60	50	10425.5 ± 1885.6
Lpr-b-04-4C	2609.4 ± 1234.5	0.0 ± 0.0	40	35	16792.5 ± 2582.4
Lpr-b-05-6C	3600.0 ± 0.0	Inf $\pm$ Inf	0	0	Inf $\pm$ Inf

Tabela 3.3 – Resultados computacionais para $\tau_1 = 0,1$ e $\tau_2 = 0,1$.

Grupo de Instâncias	$\bar{t}(s) \pm \sigma(t)$	$\overline{GAP}(\%) \pm \sigma(GAP)$	**Factível**$(\%)$	**Ótima**$(\%)$	$\overline{Deadhead} \pm \sigma(Deadhead)$
Lpr-a-01-2C	0.0 ± 0.0	0.0 ± 0.0	100	100	895.4 ± 176.1
Lpr-a-01-3C	0.2 ± 0.1	0.0 ± 0.0	100	100	809.6 ± 160.9
Lpr-a-02-2C	0.1 ± 0.0	0.0 ± 0.0	100	100	2916.6 ± 493.6
Lpr-a-03-2C	0.2 ± 0.1	0.0 ± 0.0	100	100	10276.7 ± 2016.2
Lpr-a-03-3C	365.7 ± 799.6	0.0 ± 0.0	95	90	10118.9 ± 2204.6
Lpr-a-03-4C	1202.3 ± 1190.6	0.0 ± 0.0	80	70	10528.4 ± 2175.3
Lpr-a-03-5C	3135.1 ± 1032.9	0.2 ± 0.0	35	20	12134.9 ± 2559.2
Lpr-a-04-4C	2069.5 ± 1320.5	0.0 ± 0.0	55	50	15429.7 ± 3372.0
Lpr-a-04-5C	3077.5 ± 1147.2	0.0 ± 0.0	15	15	13666.7 ± 415.2
Lpr-a-05-4C	2440.8 ± 1207.8	0.0 ± 0.0	45	45	31225.4 ± 8278.4
Lpr-b-01-2C	0.1 ± 0.0	0.0 ± 0.0	100	100	1093.2 ± 232.1
Lpr-b-01-3C	0.1 ± 0.1	0.0 ± 0.0	100	100	1046.3 ± 225.5
Lpr-b-02-2C	0.1 ± 0.1	0.0 ± 0.0	100	100	2967.4 ± 551.5
Lpr-b-03-4C	1961.3 ± 1255.8	0.0 ± 0.0	50	50	11358.9 ± 1995.1
Lpr-b-04-4C	2793.7 ± 1262.6	0.1 ± 0.0	35	25	18424.3 ± 2264.3
Lpr-b-05-6C	3600.0 ± 0.0	Inf $\pm$ Inf	0	0	Inf $\pm$ Inf

Tabela 3.4 – Resultados computacionais para $\tau_1 = 0,2$ e $\tau_2 = 0,05$.

Grupo de Instâncias	$\bar{t}(s) \pm \sigma(t)$	$\overline{GAP}(\%) \pm \sigma(GAP)$	**Factível**$(\%)$	**Ótima**$(\%)$	$\overline{Deadhead} \pm \sigma(Deadhead)$
Lpr-a-01-2C	0.5 ± 1.7	0.0 ± 0.0	100	100	759.5 ± 120.0
Lpr-a-01-3C	0.4 ± 0.4	0.0 ± 0.0	100	100	750.1 ± 163.1
Lpr-a-02-2C	0.1 ± 0.0	0.0 ± 0.0	100	100	2816.6 ± 424.3
Lpr-a-03-2C	0.1 ± 0.0	0.0 ± 0.0	100	100	10113.4 ± 1823.8
Lpr-a-03-3C	224.7 ± 253.7	0.0 ± 0.0	95	95	9346.5 ± 1862.8
Lpr-a-03-4C	538.4 ± 521.1	0.0 ± 0.0	90	90	8938.8 ± 1777.1
Lpr-a-03-5C	2489.1 ± 1269.4	0.0 ± 0.0	40	35	10332.1 ± 1940.6
Lpr-a-04-4C	1539.4 ± 1176.8	0.0 ± 0.0	65	60	12641.2 ± 1747.5
Lpr-a-04-5C	3017.5 ± 1195.3	0.1 ± 0.0	45	20	13615.7 ± 1862.7
Lpr-a-05-4C	1990.8 ± 1337.7	0.1 ± 0.0	60	50	26529.1 ± 5886.4
Lpr-b-01-2C	0.1 ± 0.3	0.0 ± 0.0	100	100	1019.8 ± 184.5
Lpr-b-01-3C	0.1 ± 0.2	0.0 ± 0.0	100	100	940.8 ± 235.3
Lpr-b-02-2C	0.2 ± 0.4	0.0 ± 0.0	100	100	2723.6 ± 475.8
Lpr-b-03-4C	1814.5 ± 1439.7	0.1 ± 0.0	65	50	11033.6 ± 2744.1
Lpr-b-04-4C	2557.1 ± 1396.3	0.1 ± 0.0	45	30	15532.6 ± 2331.7
Lpr-b-05-6C	3444.2 ± 662.0	0.0 ± 0.0	5	5	37119.0 ± 0.0

4 CONCLUSÃO

Este estudo considera o problema de particionamento de arestas em distritos Eulerianos, introduzido na literatura por García-Ayala et al. (2016). As aplicações práticas deste problemas são diversas, como na entrega de correspondências ou até mesmo na coleta de lixo, aplicações essas que mostram a utilidade da divisão de uma região em distritos para que esses serviços possam ser melhor planejados e até mesmo simultaneamente executados. Nesse sentido, este trabalho contribui para o estudo deste problema ao aprofundar na abordagem que se preocupa em formar distritos Eulerianos e utilizar-se de ferramentas disponibilizadas por um software comercial para deixar mais eficiente a resolução desse problema via algoritmo de *Branch and Bound*.

O uso da funcionalidade do *callback*, fornecida pela biblioteca GUROBIPY, permite que durante a execução da modelagem novas restrições sejam adicionadas à formulação baseadas em soluções encontradas, a fim de aproximar-se da solução ótima do problema. A execução da bateria de testes permite também uma maior compreensão da eficiência da modelagem apresentada, assim como do algoritmo implementado. O algoritmo mostrou-se eficiente, resolvendo a maioria dos casos de teste em um intervalo de tempo relativamente pequeno.

Trabalhos futuros relacionados a este problema podem aproveitar dos testes realizados e verificar como que um maior intervalo de tempo para a execução de cada caso influencia a quantidade de respostas factíveis. Outra abordagem pode levar em conta uma modelagem alternativa do problema, encontrando um método mais eficiente de criar restrições para garantir a continuidade dos distritos, sem demandar um incremento exponencial no tempo de execução. Por fim, casos de testes mais variados podem ajudar numa compreensão melhor a respeito de quais situações o algoritmo apresenta um resultado melhor, e quais seria necessário buscar uma modelagem alternativa.

REFERÊNCIAS

ASSIS, L. S.; FRANCA, P. M.; USBERTI, F. L. A redistricting problem applied to meter reading in power distribution networks. **Computers & Operations Research**, Elsevier, v. 41, p. 65–75, 2014.

BODIN, L.; LEVY, L. The arc partitioning problem. **European journal of operational research**, Elsevier B.V, AMSTERDAM, v. 53, n. 3, p. 393–401, 1991. ISSN 0377-2217.

BUTSCH, A.; KALCSICS, J.; LAPORTE, G. Districting for arc routing. **INFORMS Journal on Computing**, INFORMS, v. 26, n. 4, p. 809–824, 2014.

CORMEN, T. H. et al. **Introduction to algorithms**. [S.l.]: MIT press, 2022. 655-662 p.

ESPíNDOLA, A. M. **Uma Heurística Construtiva para o Problema de Particionamento de Arestas em Distritos Eulerianos**. Monografia (TCC) — Universidade Federal de Lavras, 2022.

GARCÍA-AYALA, G. et al. A novel model for arc territory design: promoting eulerian districts. **International Transactions in Operational Research**, Wiley Online Library, v. 23, n. 3, p. 433–458, 2016.

GLIESCH, A.; RITT, M.; MOREIRA, M. C. A multistart alternating tabu search for commercial districting. In: SPRINGER. **Evolutionary Computation in Combinatorial Optimization: 18th European Conference, EvoCOP 2018, Parma, Italy, April 4–6, 2018, Proceedings 18**. [S.l.], 2018. p. 158–173.

GOMES, M. J. N.; PALHANO, A. W. de C.; REIS, E. C. R. Sector arc routing-based spatial decision support system for waste collection in brazil. **Waste Management & Research**, v. 41, n. 1, p. 214–221, 2023. PMID: 35892193. Disponível em: <https://doi.org/10.1177/0734242X221104366>.

Gurobi Optimization, LLC. **Gurobi Optimizer Reference Manual**. 2023. Disponível em: <https://www.gurobi.com>.

LABELLE, A.; LANGEVIN, A.; CAMPBELL, J. Sector design for snow removal and disposal in urban areas. **Socio-economic planning sciences**, Elsevier Ltd, v. 36, n. 3, p. 183–202, 2002. ISSN 0038-0121.

MOREIRA, M. C. d. O. Um novo conjunto de instâncias para o problema de particionamento de arestas em distritos eulerianos. **LV Simpósio Brasileiro de Pesquisa Operacional**, p. 1–12, 2023.

MOREIRA, M. C. O. et al. **Novo conjunto de instâncias para o problema de parcelamento de arestas em distritos Eulerianos**. 2023.

<https://github.com/Space-G/instances_sbpo2023>. [Online; acessado em 18-maio-2023].

MOURãO, M. C.; NUNES, A. C.; PRINS, C. Heuristic methods for the sectoring arc routing problem. **European Journal of Operational Research**, v. 196, n. 3, p. 856–868, 2009. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221708003834>.