UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Mateus Oliveira Dogan

Problema Generalizado da Árvore Geradora Mínima

Campinas
22/11/2024

Mateus Oliveira Dogan

Problema Generalizado da Árvore Geradora Mínima

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Orlando Lee.

Resumo

O **Problema Generalizado da Árvore Geradora Mínima (PGAGM)** é uma extensão do famoso problema da **Árvore Geradora Mínima**. Enquanto o problema clássico busca encontrar um subconjunto de arestas que conecte todos os nós de um grafo de forma otimizada, no PGAGM os nós são divididos em partições, e o objetivo é encontrar um conjunto de arestas que conecte exatamente um nó de cada partição. Este trabalho está inserido no campo da **Otimização Combinatória** e tem como objetivo introduzir ao leitor conceitos fundamentais de **teoria dos grafos**, **otimização combinatória** e **programação inteira**. Ao longo do estudo, serão apresentadas formulações matemáticas para o problema, bem como suas implementações utilizando o solver **GUROBI**. Com base nos resultados obtidos, busca-se compreender a complexidade inerente ao problema e analisar seu comportamento em diferentes instâncias.

Abstract

The **Generalized Minimum Spanning Tree Problem (GMSTP)** is an extension of the well-known **Minimum Spanning Tree Problem**. While the classic problem aims to find a subset of edges that connects all nodes in a graph in an optimized way, the GMSTP divides the nodes into partitions, and the goal is to find a set of edges that connects exactly one node from each partition. This study is part of the field of **Combinatorial Optimization** and aims to introduce the reader to fundamental concepts of **graph theory**, **combinatorial optimization**, and **integer programming**. Throughout the work, mathematical formulations of the problem will be presented, as well as its implementations using the **GUROBI** solver. Based on the results obtained, the study seeks to understand the inherent complexity of the problem and analyze its behavior in different instances.

Conteúdo

1 Introdução

No **Problema Generalizado da Árvore Geradora Mínima** (*Generalized Minimum Spanning Tree Problem*), são dados um grafo conexo $G = (V, E)$, uma função custo $c : E \mapsto \mathbb{R}^+$ e uma partição $\{V_1, \ldots, V_m\}$ de V (clusters de G), e deseja-se encontrar uma subárvore de G de custo mínimo, $\min \sum_{e \in E} c_e$ contendo exatamente um vértice de cada cluster.

Este problema é importante na modelagem de várias aplicações práticas do mundo real. Por exemplo, na área de telecomunicações, queremos identificar a melhor posição para instalar centros de serviços, tais como lojas, armazéns e centros de distribuição. Este problema generaliza o famoso **Problema da Árvore Geradora Mínima** (*Minimum Spanning Tree Problem*), o caso particular em que cada cluster tem tamanho um.

Neste projeto, pretendemos estudar e implementar **algoritmos exatos** para esse problema usando a abordagem de **Programação Linear Inteira**. Em particular, estudaremos métodos para resolver programas lineares inteiros, como **Branch-and-Bound** e **Branch-and-Cut**.

A **otimização combinatória** é uma área que envolve o estudo da **complexidade de algoritmos** e o uso de técnicas como **programação dinâmica**. Muitos problemas de **Otimização Combinatória** são *NP-difíceis*, o que implica que não se pode esperar a existência de algoritmos polinomiais eficientes para resolvê-los, supondo que $P \neq NP$.

No capítulo 2 será introduzido ao leitor as ferramentas de programação inteira junto a teoria de grafos necessária para a resolução do problema, no capítulo 3 será discutida as formulações possíveis e as consequências de cada formulação e por fim no capítulo 4 serão mostrados os resultados numéricos e a conclusão do trabalho.

2 Grafos, Árvores e Combinatória

2.1 Teoria de Grafos

Antes de mais nada, é necessário que o leitor esteja familiarizado com a linguagem e as noções básicas de grafos. Um grafo simples $G = (V, A)$ consiste de uma coleção de nós (ou vértices) pertencentes a V e um conjunto de pares não ordenados dos elementos de V, tal que $A \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$. Caso o conjunto A seja ordenado, dizemos que o grafo é direcionado, e, ao invés de arestas, chamamos de arcos, os quais possuem direção.

Seja $a = (u, v)$ uma aresta. Dizemos que a incide em u e v, que u e v são os extremos de a, e que u e v são adjacentes.

Seja $S \subseteq V$ um subconjunto de vértices de V. Definimos $\delta(S)$ como o conjunto de arestas com um extremo em S e o outro em $V \setminus S$. Se $\emptyset \neq S \neq V$, chamamos $\delta(S)$ de **corte** em V. Tambem, definimos $E(S)$ como o conjunto de arestas onde os dois extremos estao em S.

$$\delta(S) := \{(u, v) \in A \mid u \in S \text{ e } v \in V \setminus S\}$$

$$E(S) = \{e = \{i, j\} \in E \mid i \in S, j \in S\},$$

Em um grafo $G = (V, A)$, um conjunto não vazio de arcos ou arestas $P = \{a_1, a_2, \ldots, a_k\}$, onde $a_i = (v_i, v_{i+1}) \in A$ e $v_i \neq v_j$ para $i \neq j$, é chamado de **caminho** P de v_1 até v_k. Se incluirmos o arco $a_{k+1} = (v_k, v_1)$, o caminho se torna um **circuito** ou **ciclo**. Para conectar k vértices em um ciclo, são necessárias k arestas.

Para o grafo direcionado $D = (V, A)$ associado ao grafo G, onde A é o conjunto de arcos contendo (i, j) e (j, i) para cada aresta $\{i, j\} \in E$, as definições equivalentes são:

$$A(S) = \{(i, j) \in A \mid i, j \in S\},$$

$$\delta^+(S) = \{(i, j) \in A \mid i \in S, j \notin S\},$$

$$\delta^-(S) = \{(i, j) \in A \mid i \notin S, j \in S\}.$$

As seguintes notações vetoriais são utilizadas:

$$x = (x_{ij}), \quad z = (z_i), \quad w = \{x_{ij}\}$$

$$x(E') = \sum_{i,j \in E'} x_{ij}, \quad \forall E' \subseteq E, \quad z(V') = \sum_{i \in V'} z_i, \quad \forall V' \subseteq V,$$

$$w(A) = \sum_{(i,j) \in A} w_{ij}, \quad \forall A' \subseteq A,$$

E para formular as variaveis do PGAGM:

- $x_e = x_{ij} = \begin{cases} 1 & \text{se a aresta } e = \{i, j\} \text{ for escolhida,} \\ 0 & \text{caso contrário.} \end{cases}$
- $z_i = \begin{cases} 1 & \text{se o vértice } i \text{ for escolhido,} \\ 0 & \text{caso contrário.} \end{cases}$
- $w_{ij} = \begin{cases} 1 & \text{se a aresta } \{i, j\} \text{ conectar vértices de diferentes clusters,} \\ 0 & \text{caso contrário.} \end{cases}$

Um grafo é chamado de **conexo** se houver um caminho entre quaisquer dois de seus vértices; caso contrário, ele é **desconexo**. Um grafo é **acíclico** se não possuir ciclos. Um grafo acíclico é chamado de **floresta**, e, se esse grafo também for conexo, ele será uma **árvore**.

Um grafo $A = (V, E)$ que é uma **árvore** satisfaz a seguinte equação:

$$|E| = |V| - 1.$$

A demonstração pode ser realizada por indução. Inicialmente, um grafo com dois vértices precisa de exatamente uma aresta. Suponha agora que uma árvore com n vértices já satisfaça a relação. Ao adicionar um novo vértice, é necessário incluir também uma nova aresta para mantê-lo conectado. Assim, a relação continua válida para $n + 1$ vértices.

É perceptível que um grafo conexo com exatamente $|V|-1$ arestas será acíclico. Isso ocorre porque, se houvesse um circuito, seria possível remover uma aresta sem desconectar o grafo, o que reduziria o número total de arestas para $|V| - 2$, e com esse número

de arestas não seria possível conectar todo o grafo. Assim, um grafo conexo com $|V| - 1$ arestas é uma árvore.

De forma análoga, um grafo acíclico com $|V|-1$ arestas será conexo. Isso ocorre porque, ao começarmos com um grafo com n vértices e nenhuma aresta, e adicionarmos uma aresta de cada vez sem formar ciclos, conseguiremos colocar apenas $n-1$ arestas sem que um ciclo seja formado. Esse ponto de parada é o mesmo alcançado pelo algoritmo de Kruskal ao construir uma árvore geradora mínima.

Portanto, para que um grafo seja uma árvore, basta verificar apenas duas das propriedades mencionadas, pois a propriedade restante será satisfeita automaticamente. Nas formulações a seguir, garantiremos sempre o número mínimo de arestas $|V| - 1$, juntamente com alguma restrição de conectividade ou aciclicidade.

2.2 Técnicas de Programação Inteira

Para resolver o PGAGM, será necessário formular o problema como um Problema de Programação Linear Inteira (PLI), onde as restrições e a função a minimizar são lineares e as variáveis de decisão são inteiras. No caso do PGAGM, as variáveis serão as arestas e os vértices. É claro que, ao escolher as arestas, os vértices são automaticamente escolhidos. No entanto, para modelar no solver e em modelos de PLI, é necessário criar as variáveis para os vértices e estabelecer as suas restrições de dependência das arestas.

A princípio, o espaço de soluções do PGAGM pode ser representado pelo espaço $\mathbb{B}^{|E|}$, onde E é o conjunto de arestas. Cada variável $x_e \in \mathbb{B}^{|E|}$ pertence ao espaço binário $\{0, 1\}$, representando a presença ou ausência da aresta no grafo. No entanto, nem todas as soluções geradas levarão à formação de uma árvore. Para garantir que a solução seja uma árvore, é necessário adicionar restrições adicionais. Os cortes que limitam o espaço de soluções do problema formulado limitam o poliedro a um poliedro convexo. As formulações apresentadas neste trabalho serão sempre politopos.

Como primeiro modelo, temos nossa função objetivo padrão do problema de minimização, a restrição (1) de partição dos clusters, onde pegaremos um nó de cada partição, em (2), uma das condições para a resposta do problema ser uma árvore. Assim, só será necessário adicionar uma restrição de aciclicidade ou conectividade do problema. As restrições (3) e (4) garantem respostas inteiras para o sistema.

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\text{s.a. } z(V_k) = 1, \quad \forall k \in K = \{1, \ldots, M\} \tag{1}$$

$$x(E) = M - 1 \tag{2}$$

$$x_e \in \{0, 1\}, \quad \forall e \in E \tag{3}$$

$$z_i \in \{0, 1\}, \quad \forall i \in V \tag{4}$$

2.2.1 Relaxação

Como primeira técnica de **Programação Inteira (PI)**, veremos a relaxação. Problemas de PI são $\mathcal{NP}$-difíceis, portanto, não podemos aplicar um método simplex diretamente. Porém, se relaxarmos as suas variáveis inteiras para variáveis reais, ou seja, $x_e, z_i \in \mathbb{R} \quad \forall i \in V, e \in E$, e adicionarmos as restrições $0 \leq x_e, z_i \leq 1$, torna-se possível resolver o sistema com o método simplex, de pontos interiores ou algum outro modelo de solução para Programação Linear.

Entretanto, a solução do novo sistema pode ser fracionária, e muito provavelmente será. Assim, só a relaxação não resolverá o problema. No entanto, a relaxação de problemas de PI pode servir de base para algoritmos de **Branch-and-Bound** e o **método do plano-de-cortes**.

2.2.2 Plano-de-corte

No método de **plano-de-cortes**, introduzido por Gomory para resolver **PLIs**, começamos com uma relaxação do problema original e, de maneira sistemática, até que a solução obtida seja inteira, serão acrescentadas novas inequações, chamadas de **planos de cortes**. Dado um poliedro P, acrescentaremos **planos-de-cortes** tais que $P \supseteq P' \supseteq P'' \supseteq \cdots \supseteq P_I$, onde P_I é o fecho inteiro do poliedro P.

Assim, basta encontrar todos os cortes necessários para solucionar o problema por meio do simplex. Entretanto, tal método pode demorar muito, e pode ser que nem todos os cortes sejam necessários. Por isso, a forma sistemática de acrescentar as restrições será a seguinte:

Algorithm 1 Método de Plano de Corte

Entrada: Problema Inteiro
Saída: Solução Inteira
Passo 1: Relaxar as variáveis do problema
Passo 2: Encontrar a solução ótima do sistema relaxado
Enquanto variáveis não inteiras **Faça**
    Passo 3: Gerar um corte ao sistema
    Passo 4: Achar uma nova solução do sistema
end Enquanto
Passo 5: Retorne a solução inteira

Assim, o algoritmo apenas irá gerar as restrições necessárias para solucionar o sistema, não sendo necessário criar um fecho inteiro induzido do poliedro P. Outra maneira de otimizar o código é acrescentar previamente restrições que provavelmente serão usadas como um corte inicial do problema, evitando calcular exaustivamente a solução de um poliedro.

No nosso problema, o plano de corte pode gerar restrições de duas formas: gerar cortes em soluções que contenham ciclos ou em soluções desconexas. Estas serão a base das formulações de Subtour Elimination e Corte Direto, que são restrições baseadas em árvores que usaremos. Além disso, o plano de corte também servirá para eliminar as soluções fracionárias do problema, onde as arestas apresentam valores não-inteiros. Assim, o politopo resultante terá como arestas árvores.

2.2.3 Branch-and-Bound

Como última técnica de PI a ser usada no trabalho, discutiremos o Branch-and-Bound. Esse método será um processo de ramificação da solução inteira, criando uma árvore de decisão. Ao invés de cortar o problema, ele o divide em subproblemas. Caso encontre uma aresta fracionária, criaremos dois nós filhos na árvore de decisão: um onde tal aresta possui valor zero e outro onde a aresta possui valor 1.

Analisaremos cada folha da árvore atual para decidir qual será a próxima folha a ser escolhida. Para isso, criaremos uma fila de prioridade com base em dois critérios: o limitante superior e o inferior. A folha com a menor diferença entre esses dois limitantes será a próxima a ser selecionada para exploração.

O limitante superior será o resultado da relaxação do problema, um valor

do qual a solução de minimização não consegue ser menor. À medida que limitamos o problema com mais restrições, o limitante inferior tende a aumentar, enquanto o limitante superior não ultrapassa um valor máximo. Com isso, a diferença entre os limitantes superior e inferior vai diminuindo.

A princípio, o cálculo do limitante superior pode ser feito utilizando o problema dual de minimização, mas fazer isso a cada interação do **Branch-and-Bound** custaria muito. Por isso, usaremos heurísticas para encontrar um valor aceitável para o limitante superior. Se o limitante inferior for maior que o limitante superior, o problema será inviável. Se os dois limitantes forem iguais, então teremos encontrado o mínimo do sistema.

Há também a possibilidade de combinar as técnicas, formando o algoritmo de **Branch-and-Cut**. Nesse caso, além de realizar o branching das variáveis, caso seja detectada uma discontinuidade ou aciclicidade em uma solução inteira, pode-se criar um corte nela.

Algorithm 2 Algoritmo Branch and Bound

Variáveis:
`LB` (*limitante inferior*)
`UB` (*limitante superior*)
`Ativos` (*lista de nós ativos da árvore*)
Lê os dados do problema;
`LB` := 0
`UB` := ∞
`Ativos` := raiz
Enquanto LB < UB Faça
  Se não existe nenhum nó ativo na lista `Ativos` **then**
    Busca encerrada
    Fim do algoritmo
  Senão
    i := Nó ativo da lista `Ativos`
    Acertar variáveis fixadas nesse nó
    Calcular limitante inferior $\mathbf{LB}_i$ para o nó
    Se $\mathbf{LB}_i > \mathbf{LB}$ **then**
      Atualizar `LB`
    end Se
    Calcular limitante superior $\mathbf{UB}_i$ para o nó
    Se $\mathbf{UB}_i < \mathbf{UB}$ **then**
      Atualizar `UB`
    end Se
    Se $\mathbf{LB}_i > \mathbf{UB}$ **then**
      Eliminar i **da lista** `Ativos`
    Senão
      Escolher variável a ser fixada
      Retirar i da lista `Ativos`
      Criar dois novos nós, fixando a variável em 0 e 1 e os incluir na lista `Ativos`
    end Se
  end Se
end Enquanto

Observação: O algoritmo referencia se encontra na pag:103 do livro de Yoshiko, referência 2.

2.3 Complexidade Computacional

Myung[5] provou que o PGAGM é um problema $\mathcal{NP}$-difícil. Sua prova baseia-se no fato de que o PGAGM é derivado do problema de Node Cover. Um problema $\mathcal{NP}$-difícil é aquele para o qual não existe, no caso geral, um algoritmo de tempo polinomial

para resolvê-lo. No entanto, em certos casos particulares, é possível encontrar algoritmos polinomiais para o PGAGM.

Por exemplo, quando todos os clusters possuem exatamente um nó, o problema se reduz ao problema padrão da árvore geradora mínima, que pode ser resolvido utilizando algoritmos gulosos como os de Kruskal ou Prim. Outro caso particular ocorre quando o grafo possui apenas duas partições, em que basta encontrar a aresta de menor custo que conecta os dois clusters.

Pop[4] apresentou um algoritmo de programação dinâmica para resolver o PGAGM no caso em que o cluster é fixo, e não uma variável. Neste cenário, o problema pode ser resolvido em tempo polinomial, embora com um polinômio de ordem elevada, $O(m^{m-2}n^2)$, que é eficiente apenas para instâncias com um número pequeno de clusters.

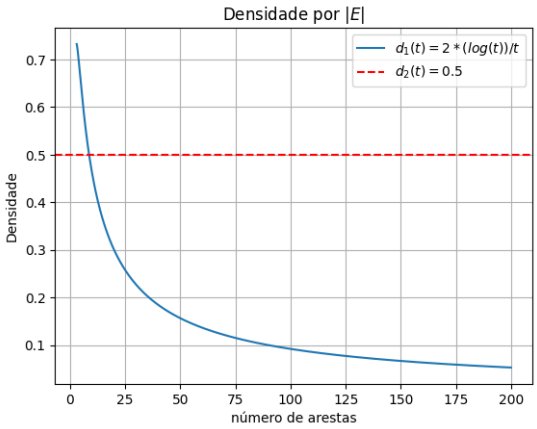
Para resolver o caso geral do problema, utilizaremos programação inteira e testaremos diferentes formulações. No PGAGM, a complexidade e o tempo de execução do algoritmo dependem tanto da quantidade de clusters da instância quanto da quantidade de arestas do grafo. Grafos mais densos tendem a ser mais demorados de serem processados por uma PLI, uma vez que cada aresta adiciona uma variável ao solver GUROBI, além de contribuir com mais camadas na árvore de Branch-and-Bound.

Com isso em mente, definiremos duas funções de densidade para os grafos a serem testados. A inspiração para testar tais modelos de grafos vem de Myung [5] para os grafos aleatórios e euclidianos. Logo em seguida, Pop [4] propôs os grafos euclidianos não-estruturados para os testes de formulações. Os grafos serão gerados de maneira aleatória com as seguintes estruturas:

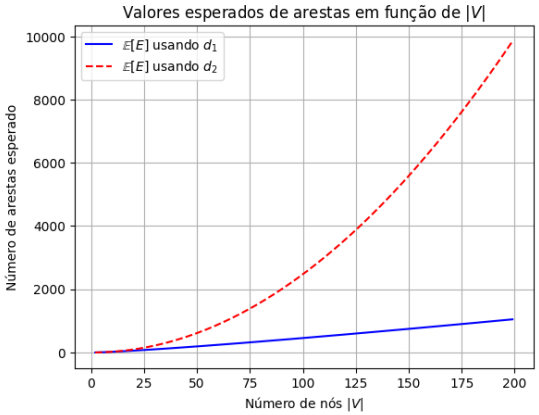
- **Grafos Euclidianos Estruturados**: Nós de uma mesma partição estão próximos uns dos outros, e os custos das arestas respeitam a desigualdade triangular euclidiana.
- **Grafos Euclidianos Não Estruturados**: As partições são escolhidas de maneira aleatória, mas os custos continuam obedecendo as restrições euclidianas.
- **Grafos Aleatórios**: Os custos das arestas são definidos por alguma função aleatória.

As funções de densidade dos grafos serão as seguintes:

- $d_1 = 0.5$: Uma densidade constante.
- $d_2(n) = \frac{2\log(n)}{n}$: A densidade definida pelo modelo de Erdős–Rényi.

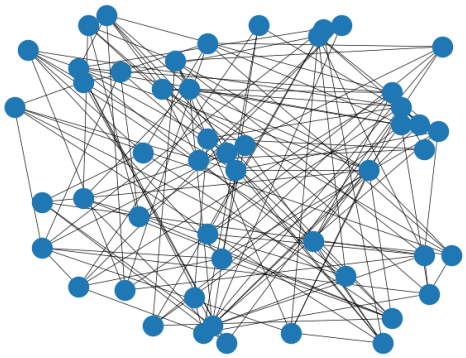


(a) Gráfico densidade por $|V|$

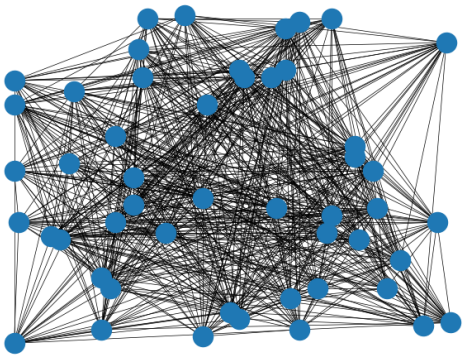


(b) $|E|$ por $|V|$

Figura 1: Gráficos densidade e tamanho do grafo.



(a) Grafo com densidade Erdős-Rényi



(b) Grafo com densindade 0.5

Figura 2: Grafos com 50 vértices e diferentes densidades

3 Formulações e poliedros do problema

Na literatura são encontradas disversas formulações com diversas maneiras de retratar as restriçoes de conetividade e aciclicidade no PGAGM, neste trabalho iremos estudar as 4 formulaçoes, duas baseadas em arvores e duas baseadas em fluxos.

3.1 Formulações baseadas em árvores

Foi utilizada como base para o início dos estudos a formulação do PGAGM, *Generalized Subtour Elimination Formulation* Myung[5]. A formulação é apresentada a seguir:

$$
\begin{aligned}
& min \sum_{e \in E} c_e \cdot x_e \\
\text{s.a. } & z(V_k) = 1, \quad \forall k \in K = \{1, ..., M\} \\
& x(E) = M - 1 \\
& x(E(S)) \leq z(S - i), \quad \forall i \in S \subset V,\ 2 \leq |S| \leq n - 1 \qquad (5) \\
& x_e \in \{0, 1\}, \quad \forall e \in E \\
& z_i \in \{0, 1\}, \quad \forall i \in V
\end{aligned}
$$

A restrição (5) tambem conhecida como *generalized subtour elimination constraints*, garante a conectividade do grafo, explicado em Myung[5].

Substituímos as restrições (3) e (4) por $0 \leq x_e, z_i \leq 1$, para todo $e \in E$ e $i \in V$.

Formulação de corte generalizado (Myung et al., 1995) As restrições de eliminação de subtour (2) podem ser substituídas pelas restrições de conectividade (6), resultando no seguinte problema de programação inteira linear:

$$
\begin{aligned}
& min \sum_{e \in E} c_e \cdot x_e \\
\text{s.a. } & (1), (2), (3), (4) \\
& x(\delta(S)) \geq z_i + z_j - 1, \quad \forall i \in S \subset V,\ j \notin S,\ 1 \leq |S| \leq n - 1 \qquad (6)
\end{aligned}
$$

3.2 Formulações baseadas em fluxos

As formulações apresentadas anteriormente possuem um número exponencial de restrições. As formulações a seguir têm um número polinomial de restrições, mas

com mais variáveis, o que farão o modelo calcular mais rapido o resultado e permitirão o modelo ser aplicado a grafos maiores. Para obter formulações compactas do problema GMST, introduzimos variáveis auxiliares de fluxo, além das variáveis binárias associadas às arestas e vértices do grafos. Todos os modelos usam variáveis de fluxo direcionado, apesar de as arestas serem não direcionadas, transformamos o grafo do problema em um grafo direcionado, no qual para cada aresta criamos um par de aresta em cada direção. A formulação de *single commodity*, por exemplo, envia uma unidade de fluxo de um cluster de origem para cada cluster destino.

single commodity flow formulation (Pop[4]) Neste modelo, o cluster de origem V_1, escolhido de maneira arbitrária, envia uma unidade de fluxo para cada cluster. O fluxo em cada aresta é denotado por f_{ij}, representando o fluxo na aresta $\{i, j\}$ da direção i para j.

$$\min \sum_{e \in E} c_e \cdot x_e$$

s.a. $(1), (2), (3), (4)$

$$\sum_{e \in \delta^+(i)} f_e - \sum_{e \in \delta^-(i)} f_e = \begin{cases} (m-1)z_i, & \text{para i} \in V_1 \\ -z_i, & \text{para i} \in V - V_i \end{cases} \tag{7}$$

$$f_{ij} \leq (m-1)x_e, \quad \forall e = \{i, j\} \in E \tag{8}$$

$$f_{ji} \leq (m-1)x_e, \quad \forall e = \{i, j\} \in E \tag{9}$$

$$f_{ij}, f_{ji} \geq 0, \quad \forall e = \{i, j\} \in E \tag{10}$$

E também existe o *Multi-Commodity Flow*, onde, ao invés de enviar um fluxo f para cada cluster, definimos no início do programa um fluxo f^k para cada cluster V_k. Definiremos um conjunto $K_1 = \{2, 3, ..., m\}$.

$$\min \sum_{e \in E} c_e \cdot x_e$$

s.a. $(1), (2), (3), (4)$

$$\sum_{a \in \delta^+(i)} f_a^k - \sum_{a \in \delta^-(i)} f_a^k = \begin{cases} z_i, & \text{para } i \in V_1 \\ -z_i, & \text{para } i \in V_k, \quad k \in K_1 \\ 0, & \text{se } i \notin V_1 \cup V_K \end{cases} \tag{11}$$

$$w_{ij} + w_{ji} = x_e, \quad \forall e = \{i, j\} \in E, \tag{12}$$

$$f_a^k \geq 0, \quad \forall a = \{i, j\} \in A, k \in K_1 \tag{13}$$

$$x, z \in \{0, 1\} \tag{14}$$

De acordo com Pop[4], o *Multi-Commodity Flow* é o que melhor se sobressai das 4 formulações. Os algoritmos baseados em fluxo têm uma grande vantagem sobre os algoritmos baseados em árvores devido à redução no número de restrições. O aumento no número de variáveis é insignificante diante dessa redução.

4 Implementação e Conclusão

Para os testes, foi implementado o modelo de *Multi-Commodity Flow* para ambos os grafos euclidianos não estruturados.

O objetivo do projeto era de se estudar o o Problema Generalizado da árvore Geradora Mínima e na implementação de algoritmos exatos usando técnicas de Programação Linear Inteira.

MCF			
		media tempo em segundos	
Clusters	nós por Cluster	modelo Erdos	modelo Constante
6	3	0.04286160469	0.05299153328
6	4	0.06279006004	0.09672670364
6	6	0.05165758133	0.1774363518
6	10	0.1041509628	0.7616911888
8	3	0.08368291855	0.0848903656
8	4	0.06979622841	0.3211507797
8	6	0.1022540569	1.149650335
8	10	0.7455811977	5.262573147
10	3	0.5712034225	0.3213274956
10	4	0.5891693592	2.250704813
10	6	1.140278769	2.95252986
10	10	1.139356136	15.17361803
12	3	0.5246535778	1.493015194
12	4	0.5129902363	2.968408346
12	6	0.9811861992	8.947144222
12	10	4.328721619	53.49946337
15	3	0.841170454	4.197636795
15	4	1.603239584	9.982121611
15	6	3.055028343	41.31168613
15	10	7.251432467	247.0664502
18	3	0.9043779373	1.305770159
18	6	3.492486	62.98112512
18	10	16.25128698	181.9699812
20	3	3.020251036	11.31416798
20	6	0.9869999886	103.9004378
20	10	40.46605015	609.1217101

Referências

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009.

[2] Ferreira, C. E. ; Y. Wakabayashi . Combinatória Poliédrica e Planos-De-Corte Faciais. 1. ed. Campinas, SP: X ESCOLA DE COMPUTACAO, 1996. 130p .

[3] Laurence A. Wolsey. Integer Programming. John Wiley & Sons, Hoboken, Estados Unidos, 1998.

[4] Petrica C Pop. The Generalized Minimum Spanning Tree Problem: An overview of formulations, solution procedures and latest advances. *European Journal of Operational Research,* 283(1):1–15, 2020.

[5] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. On The Generalized Minimum Spanning Tree Problem. Networks, 26(4):231–241, 1995.

[6] Paul Pop. Generalized Minimum Spanning Tree Problem – Formulations, Algorithms and Applications. Ph.D. Dissertation, *University of Aarhus, Faculty of Science, Department of Computer Science,* Aarhus, Denmark, 2002.

[7] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming,* 91(2):201–213, 2001.

[8] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2024. URL: `https://www.gurobi.com`.

[9] Bruce Golden, S. Raghavan, and Daliborka Stanojević. Heuristic Search for the Generalized Minimum Spanning Tree Problem. The Robert H. Smith School of Business, University of Maryland, College Park, Maryland, USA, 2005.