UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS

**Wesly Carmesini Ataide**

# MINIMIZAÇÃO DO CONTÁGIO EM UMA TOPOLOGIA COM PROGRAMAÇÃO MATEMÁTICA

**FLORIANÓPOLIS**
**2022**

WESLY CARMESINI ATAIDE

# MINIMIZAÇÃO DO CONTÁGIO EM UMA TOPOLOGIA USANDO PROGRAMAÇÃO MATEMÁTICA

**Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Ciências da Computação**

Florianópolis, março de 2022

WESLY CARMESINI ATAIDE

MINIMIZAÇÃO DO CONTÁGIO EM UMA TOPOLOGIA USANDO PROGRAMAÇÃO MATEMÁTICA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciências da Computação, sendo aprovada em sua forma final pela banca examinadora:

---

Orientador: Prof. Dr. Álvaro Junio Pereira Franco
Universidade Federal de Santa Catarina - UFSC

---

Coorientador: Prof. Dr. Rafael de Santiago
Universidade Federal de Santa Catarina - UFSC

---

Prof. Dr. Pedro Belin Castellucci
Universidade Federal de Santa Catarina - UFSC

# Resumo

Este trabalho tem por objetivo construir soluções para a minimização de contágio em uma topologia de bairros restringindo a movimentação da população entre locais. A resolução possui como foco as ferramentas da programação matemática. Desenvolvemos e implementamos um modelo de contágio, assim como uma formulação de programação não-linear equivalente a ser resolvida com ferramentas de otimização. Desenvolvemos uma instância de um grafo baseado em dados reais da COVID-19, além de um método para encontrar e melhorar soluções. Por fim, mostramos alguns resultados demonstrando a dificuldade de lidar com a complexidade do problema e a imprecisão da ferramenta.

**Palavras-chave**: Programação Matemática, Contágio, Modelos Epidemiológicos, Redes Complexas, Programação Linear, Programação Não-linear

# Sumário

# Lista de ilustrações

# 1 Introdução

Depois dos primeiros casos de COVID-19, doença causada pelo coronavírus SARS-CoV-2, serem registrados em dezembro de 2019 na China, a infecção se espalhou rapidamente, representando um grande perigo para sistemas de saúde e para a economia mundial. Desde março de 2020, quando a Organização Mundial da Saúde declarou a situação da COVID-19 como uma pandemia, vários governos implementaram diferentes medidas, de brandas a radicais, baseadas no distanciamento social, isolamento e quarentena, como recomendado pela OMS (PEIXOTO *et al.*, 2020).

As medidas adotadas por muitos países em resposta consistiram em restrições de movimento, limitando o funcionamento de estabelecimentos e incentivando seus cidadãos a ficarem em casa, com o objetivo de reduzir o contato entre indivíduos e, desse modo, frear o contágio. Com essas ações, entretanto, vem a questão de quando é seguro reabrir, quais lugares podem ser reabertos e quanta atividade pode ser permitida (CHANG *et al.*, 2021).

Para responder essas questões, modelos epidemiológicos se tornaram cruciais. Com um volume grande de dados, eles podem identificar grupos socioeconômicos mais vulneráveis, apontar classes de estabelecimentos mais propensas a espalhar a infecção e encontrar padrões para maximizar a efetividade de medidas de restrição (CHANG *et al.*, 2021) .

Medidas que diminuem o contágio são os instrumentos mais importantes na luta contra o COVID-19 numa perspectiva social, pois tanto diminuem o número de infectados como ajudam a garantir o tratamento para pessoas que se infectarem (CHANG *et al.*, 2021) . Com isso, previsões se tornaram fundamentais para reduzir os impactos negativos da pandemia tanto numa esfera social quanto numa esfera econômica (Liu *et al.*, 2020) , ao passo que guiam as políticas públicas adotadas.

Espelhando as recomendações na vida real, diferentes publicações já tentaram analisar a modelagem e efetividade do isolamento social. Para o modelo em Liu *et al.* (2020) , simulações comprovaram a eficácia do isolamento de pessoas infectadas e de bairros com taxa de infecção particularmente alta, o que está de acordo com as recomendações implementadas em diversos países.

Este trabalho propõe analisar o modelo de contágio apresentado em Franco (2021) sob a perspectiva da programação não-linear para buscar uma solução ótima. Naturalmente, isso implica em criar uma formulação do modelo que se conforme às ferramentas de otimização, além de analisar formas de melhorar o custo de encontrar essas soluções ótimas, especialmente em questões de tempo.

Sobre a organização do trabalho, essa primeira parte foi onde introduzimos o tema e apresentamos uma visão geral do trabalho. No capítulo 2, discutimos os conceitos e definições que vamos usar no corpo do trabalho. No capítulo 3, apresentamos e comparamos trabalhos com temas similares. No capítulo 4, detalhamos o modelo com o qual vamos trabalhar. No capítulo 5, damos enfoque a implementação do modelo. No capítulo 6, apresentamos alguns resultados obtidos com o modelo.

# 2 Conceitos básicos

Para que o modelo e a implementação sejam explicados mais claramente, apresentamos neste capítulo alguns dos conceitos básicos usados pelo corpo deste trabalho. Isso inclui definições sobre grafos, usadas na explicação da topologia do modelo, modelos epidemiológicos, que definem como o contágio funciona, e programação matemática ,que é um ponto crucial em nossas tentativas de solução do problema.

## 2.1 Definições básicas de grafos

Nesta seção, são apresentadas algumas definições básicas de grafos, parcialmente baseadas em Wilson (1996) .

Um **grafo simples** (ou **não-dirigido**) $G$ consiste de um conjunto finito e não vazio $V(G)$ de elementos chamados de **vértices** (ou **nós**), e um conjunto finito $E(G)$ de pares não-ordenados de elementos distintos de $V(G)$, chamados de **arestas**. Dizemos que dois vértices $v$ e $w$ de um grafo $G$ são **adjacentes** se existe uma aresta $\{v, w\}$, a qual é dita **incidente** em $v$ e $w$. Similarmente, duas arestas são adjacentes se compartilham um vértice. O **grau** de um vértice $v$ de um grafo $G$ é o número de arestas incidentes no vértice $v$. Um grafo **valorado** tem um valor não-negativo (chamado **peso**) associado com cada uma das arestas. Um **subgrafo** de um grafo $G$ é um grafo cujos vértices todos pertencem a $V(G)$ e cujas arestas todas pertencem a $E(G)$.

Sejam dois grafos $G_1 = (V(G_1), E(G_1))$ e $G_2 = (V(G_2), E(G_2))$, com $V(G_1)$ e $V(G_2)$ disjuntos. A **união** $G_1 \cup G_2$ é o grafo com o conjunto de vértices $V(G_1) \cup V(G_2)$ e o conjunto de arestas $E(G_1) \cup E(G_2)$. Um grafo é **conexo** se não pode ser expresso como a união de dois ou mais grafos distintos, e **desconexo** caso contrário. Um grafo desconexo pode ser expresso como a união de vários grafos conexos, cada um chamado de uma **componente conexa**, ou apenas **componente**.

Uma **cadeia** em $G$ é uma sequência finita de vértices de $G$ em que cada par de vértices consecutivos é adjacente. Se uma cadeia não passa mais de uma vez pelo mesmo vértice, é definida como **caminho**. Um caminho mais uma aresta que conecta o primeiro e o último vértice é um **ciclo**.

Uma **árvore** é um grafo conexo sem ciclos. Uma **árvore geradora** de um grafo conexo $G$ é uma árvore com o conjunto de vértices igual a $V(G)$ (e o conjunto de arestas contido em $E(G)$). Em especial, para um grafo valorado nas arestas $G$, uma **árvore geradora mínima** é uma árvore geradora de $G$ cuja soma dos valores das arestas que a compõem é mínima dentre todas as árvores geradoras.

Já um grafo **dirigido** tem, no lugar do conjunto das arestas, um conjunto finito $A(G)$ de pares ordenados de elementos distintos de $V(G)$, chamados de **arcos**. Os grafos dirigidos também podem ser valorados, com um valor para cada arco em vez de cada aresta. Grafos dirigidos têm uma definição própria de conexidade, mas usamos uma diferente, que é explicada mais adiante.

## 2.2 Modelo epidemiológico

Um **modelo epidemiológico** define como a infecção se comporta do ponto de vista epidêmico. O modelo que usamos neste trabalho para modelar o contágio, por exemplo, foi o **modelo SIR**, que dinamicamente divide as pessoas em três categorias (FRANCO, 2021):

- **suscetíveis** (S), que são as pessoas passíveis de serem infectadas caso elas tenham contato com algum infectado;
- **infectados** (I), que contraíram a infecção e podem transmitir para pessoas suscetíveis;
- **recuperados** (R), formado por pessoas que não podem transmitir nem serem infectadas. Tipicamente isso é por terem desenvolvido resistência depois de já terem sido infectadas e se recuperado. Alternativamente, pessoas que morreram ou foram imunizadas por vacinação poderiam fazer parte deste grupo.

O modelo SIR inclui ainda a **virulência** $v$, que indica qual a probabilidade de uma pessoa suscetível se infectar quando ela entrou em contato com uma pessoa infectada, e a **taxa de recuperação** $\mu$, que representa um tempo médio de recuperação de pessoas infectadas em um dado instante. Dependendo do que for mais apropriado para o método de transmissão da doença analisada, outros modelos podem ser escolhidos. Alguns exemplos são o modelo SIS, em que pessoas infectadas não desenvolvem imunidade e, desse modo, se tornam suscetíveis novamente, ou o modelo SEIS, que adiciona um período de latência exposto, em que um indivíduo está infectado, mas ainda não é capaz de transmitir (SANTIAGO *et al.*, 2016).

## 2.3 Programação matemática

Em geral, se $c_1, c_2, \ldots, c_n$ são números reais, então uma função com variáveis reais $x_1, x_2, \ldots, x_n$ definida por

$$f(x_1, x_2, \ldots x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n = \sum_{j=1}^{n} c_jx_j$$

é chamada de **função linear**. Se $f$ é uma função linear, e $b$ um número real, a equação

$$f(x_1, x_2, \ldots x_n) = b$$

é uma **equação linear** e as inequações

$$f(x_1, x_2, \ldots x_n) \leq b$$

$$f(x_1, x_2, \ldots x_n) \geq b$$

são chamadas **inequações lineares**. As equações e inequações lineares constituem **restrições lineares**.

Finalmente, um problema de **programação linear** é o problema de maximizar (ou minimizar) uma função linear (a **função objetivo**) sujeita a um número finito de restrições lineares (CHVÁTAL, 1983).

Eventualmente, certas restrições ou a função objetivo de um problema a ser otimizado não são lineares. Nesse caso, é necessário usar ferramentas da **programação não-linear**, que permite resolver problemas de otimização dessa forma. Outro caso ainda é o da **programação inteira**, que cuida de casos em que algumas variáveis apenas podem tomar valores inteiros.

De um problema de programação linear (ou não-linear) é possível derivar um outro problema, chamado o **dual** do problema original, o **primal**. Uma solução do primal implica em um limite para o valor ótimo do dual e vice-versa (CHVÁTAL, 1983; BOYD et al., 2004). Em particular, se $p$ é valor da função objetivo de uma solução do primal e $d$, o valor da solução induzida no dual, chamamos a diferença $p - d$ de **gap**. Se o *gap* for igual a 0, temos a confirmação de que as soluções encontradas são ótimas. Entretanto, a recíproca não é válida: é possível que soluções com um *gap* maior que zero sejam, de fato, ótimas.

Nós nos referimos a programas que resolvem problemas dessas áreas da programação matemática (ou seja, que tentam fornecer uma solução ótima) como **solvers**. Como internamente as escolhas de variáveis são analisadas como uma árvore de escolhas, também é possível medir quanto do espaço de soluções já foi verificado com o número de **nós** dessa árvore que já foram explorados.

# 3 Trabalhos relacionados

A estrutura básica do nosso modelo é apresentada em Franco (2021), que também executa alguns experimentos para analisar o contágio com topologias diferentes. Nosso trabalho se ocupa em revisitar o modelo sobre uma perspectiva de programação não-linear.

O trabalho Santiago *et al.* (2016), como o nosso, busca minimizar o número de infectados em um grafo, porém com um modelo em que os vértices representam indivíduos e com o modelo de infecção SEIS, em que um indivíduo pode passar pelos estágios suscetível, exposto, infectado e suscetível novamente. O algoritmo proposto testa soluções possíveis removendo algumas das arestas do grafo até certa condição ser atendida, e escolhendo a melhor solução dentre as testadas. Para enfrentar a dificuldade do problema e a grande quantidade de possibilidades, essa proposta é expandida no trabalho Concatto *et al.* (2017), que propõe um algoritmo genético para gerar soluções não necessariamente ótimas. Neste trabalho, nos preocupamos com uma visão um pouco mais abstrata, que modela o movimento baseada nos bairros, além de focarmos em obter uma solução ótima.

Uma formulação da redução do risco de infecção com programação inteira linear é mostrada em Popa (2020). O autor propõe um modelo com um grafo bipartido, com pessoas em uma partição e estabelecimentos em outra. Existe a possibilidade de fechar os estabelecimentos e isolar pessoas a um custo, diminuindo o contato entre pessoas e assim a chance de acontecer uma infecção. Já que o problema é NP-difícil, como provado no trabalho, é desenvolvido um algoritmo heurístico para aproximar uma solução ótima. Apesar do modelo que usamos ser similar em essência, não nos preocupamos com o custo das nossas restrições, e sim apenas com a conectividade do grafo. Além disso, a nossa formulação precisa de algumas restrições não-lineares.

O trabalho Deng *et al.* (2013) investiga problemas de prevenção de doenças e controle de epidemias otimizando decisões de vacinação de indivíduos e fechamento de estabelecimentos. É aplicado um modelo similar ao de Popa (2020), com vários indivíduos que desejam ir a certos estabelecimentos. Alguns algoritmos são desenvolvidos e comparados, inclusive um de programação inteira. Novamente, apesar do nosso modelo ser similar, nossa perspectiva de bairros acarreta restrições específicas.

O trabalho Word *et al.* (2012) procura desenvolver um framework para estimar os parâmetros de transmissão em doenças infecciosas em crianças usando equações diferenciais contínuas, que são formuladas em um problema de programação não-linear. Como o contágio é modelado com as equações diferenciais, não há a preocupação em lidar com a topologia ou a dinâmica de movimento da população, diferentemente do nosso trabalho.

Por último, o trabalho Gutin *et al.* (2021) analisa o efeito do distanciamento social

no alcance de uma epidemia. É definido um modelo SIR com distanciamento social, em que os indivíduos são limitados em número de conhecidos com os quais eles podem interagir. Isso restringe a transmissão para uma sub-rede do grafo original. Os resultados indicam que a eficácia é altamente dependente da topologia e que há a necessidade de coordenação de políticas no nível global para a eficácia das medidas. Dada a similaridade com o nosso modelo, é possível que essas conclusões também se apliquem ao nosso. Nosso foco, porém, não se restringe aos resultados de simulações, já que vamos buscar ativamente por uma solução ótima.

# 4 Definição do modelo e solução inicial

A nossa modelagem do problema, baseada em Franco (2021) consiste na minimização do contágio em uma rede com bairros interconectados por estradas, representada por um grafo em que os bairros são os vértices (com uma população associada) e cada estrada entre um bairro $i$ e um bairro $j$ é um par de arcos $(i, j)$ e $(j, i)$. Nas próximas seções, vamos entrar em detalhes sobre cada aspecto relevante do modelo.

## 4.1 População dos vértices

Como já mencionado, o modelo epidemiológico escolhido para analisarmos foi o SIR. Levando isso em conta, a população de cada vértice $i$ para cada instante $t$ consiste no número de pessoas suscetíveis $S_i^t$, infectadas $I_i^t$ e recuperadas $R_i^t$.

O movimento entre vértices também é modelado, porém é restrito de modo que pessoas do vértice $i$ apenas podem visitar vértices adjacentes de $i$, denotados por $\chi(i)$. Ainda definimos para cada arco $(i, j)$ os valores $\beta_{i\to j}$ que representam a porcentagem de pessoas que residem no vértice $i$ e visitam o vértice $j$, ou seja, que "utilizam a estrada $(i, j)$". Também é associado a cada arco $(i, j)$ uma variável binária $x_{i\to j}$ (que restringimos para ter o mesmo valor de $x_{j\to i}$), que indica que o tráfego está bloqueado se $x_{i\to j} = 0$ ou que está livre se $x_{i\to j} = 1$.

É relevante notar que essa construção faz com que o grafo seja mais facilmente analisado como um grafo dirigido da perspectiva dos valores $\beta$ e como um grafo não-dirigido da perspectiva das variáveis $x$. Desse modo, quando nos referimos a uma árvore ou a componentes conexas nesse grafo, estamos o analisando do modo não-dirigido.

As ruas bloqueadas e liberadas criam uma variação no número de pessoas em cada um dos vértices, afetando o contágio, como é explicado a seguir.

## 4.2 Dinâmica do contágio

Para analisar a dinâmica do contágio, observa-se que, em cada instante $t$, as pessoas circulando em cada vértice $i$ se dividem em dois grupos: as que vêm de um vértice vizinho a $i$ e as que moram e ficaram em $i$.

Para o primeiro grupo, para cada vértice vizinho, o número de pessoas vindo obedece às seguintes fórmulas:

$$S_{j\to i}^t = \beta_{j\to i} x_{j\to i}^t S_j^t;$$

$$I^t_{j\to i} = \beta_{j\to i} x^t_{j\to i} I^t_j;$$
$$R^t_{j\to i} = \beta_{j\to i} x^t_{j\to i} R^t_j.$$

Já o segundo grupo consiste das pessoas que não saíram do vértice $i$ e, desse modo, basta subtrair a porcentagem de pessoas que saiu:

$$S^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j} x^t_{i\to j}))S^t_i;$$
$$I^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j} x^t_{i\to j}))I^t_i;$$
$$R^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j} x^t_{i\to j}))R^t_i.$$

É relevante observar que se uma estrada é bloqueada, a porcentagem das pessoas que tinham intenção de usá-la é acrescentada no vértice onde as pessoas moram.

Com isso, o número de pessoas circulando de cada categoria é dado por:

$$\dot{S}^t_i = S^t_{i\to i} + \sum_{j\in\chi(i)} S^t_{j\to i}$$
$$\dot{I}^t_i = I^t_{i\to i} + \sum_{j\in\chi(i)} I^t_{j\to i}$$
$$\dot{R}^t_i = R^t_{i\to i} + \sum_{j\in\chi(i)} R^t_{j\to i}$$

O objetivo é minimizar o número de infectados ao longo do tempo. É necessário considerar para cada vértice e instante o número de pessoas que se recuperam e o número de novos infectados, que é uma razão do número de encontros entre pessoas suscetíveis e infectadas:

$$S^{t+1}_i = S^t_i - v\mathbb{E}^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i);$$
$$I^{t+1}_i = I^t_i - \xi I^t_i + v\mathbb{E}^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i);$$
$$R^{t+1}_i = R^t_i + \xi I^t_i.$$

Como a porcentagem de infectados das pessoas circulando em um vértice $i$ é $\dfrac{\dot{I}^t_i}{\dot{S}^t_i + \dot{I}^t_i + \dot{R}^t_i}$, tomamos esse valor como a proporção de suscetíveis que tem um encontro com um infectado nesse vértice. O número esperado de encontros para a população do vértice $i$ é, portanto,

$$\mathbb{E}^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i) = \frac{S_{i\to i}\dot{I}^t_i}{\dot{S}^t_i + \dot{I}^t_i + \dot{R}^t_i} + \sum_{j\in\chi(i)} \frac{S_{i\to j}\dot{I}^t_j}{\dot{S}^t_j + \dot{I}^t_j + \dot{R}^t_j}.$$

Temos uma limitação sobre o modo como escolhemos bloquear as ruas, entretanto: devemos garantir que seja possível acessar todos os bairros, ou seja, o subgrafo induzido pelas ruas liberadas deve ser conexo. Na próxima seção, discutimos como garantir essa restrição no nosso modelo de programação não-linear.

## 4.3 Conexidade do grafo

Quando estamos buscando uma solução, temos a restrição de que o subgrafo gerado pela escolha ótima das estradas livres precisa ser conexo, além de conter todos os vértices do grafo original. Aqui discutimos dois métodos para conseguir isso.

### 4.3.1 Método das mercadorias

Um modo de garantir a conexidade (baseado em Carvalho *et al.* (2020)) é através do fluxo com uma abordagem similar a de produtores e consumidores de mercadorias, com um vértice arbitrário $s$ escolhido como fonte e as restrições a seguir.

$$\sum_{(i,j)\in E} f_{i\to j} + 1 = \sum_{(j,i)\in E} f_{j\to i} \text{ para cada vértice } i \neq s;$$

$$\sum_{(s,j)\in E} f_{s\to j} = n - 1;$$

em que $f_{i\to j}$ representa o fluxo que passa pelo arco $(i, j)$, e $n$ é o número de vértices do grafo.

Uma escolha das variáveis $f$ que satisfaça essas condições é um fluxo $s$-arborescente. Note que cada fluxo $s$-arborescente induz uma árvore geradora no grafo ao tomarmos como arestas todo par de arcos $(i, j)$ em que $f_{i\to j} > 0$ ou $f_{j\to i} > 0$. Como nossos arcos vêm em pares, a estrutura do grafo essencialmente se mantém.

A escolha da fonte $s$ não importa, como vamos provar a seguir.

**Teorema:** Seja $G$ um grafo conexo, e $T$ uma árvore geradora de $G$. Para todo vértice $v$ em $T$, existe um fluxo $v$-arborescente que gera a árvore $T$.

*Demonstração.* Vamos proceder por indução no número de vértices $|V|$ da árvore. O caso com $|V| = 1$ (mostrado na Figura 1 é válido, aceitando que o fluxo de um vértice sem arcos vale 0.

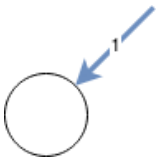


Figura 1 – O caso base da indução.

Tomamos como verdade o caso com $|V| = n$. Para o caso $|V| = n + 1$, escolhemos uma folha qualquer $l$. Seja $T'$ a árvore $T$ com o vértice $l$ removido. Separamos então em dois casos:

Se o vértice $l$ é definido como fonte de $T$, e $p$ é um vértice (único) tal que $\{p, l\}$ é uma aresta na árvore $T$, escolhemos $f_{l\to p} = (n + 1) - 1 = n$ e $f_{l\to k} = 0$ para os outros vértices $k$ que eventualmente sejam adjacentes com $l$, satisfazendo a segunda condição.

Como $T'$ possui $n$ vértices e $p$ é um de seus vértices, existe um fluxo $p$-arborescente que gera $T'$. Com a escolha $f_{p\to l} = 0$, temos que

$$\sum_{(p,j)\in E} f_{p\to j} + 1 = (n-1) + 1 = n = \sum_{(j,p)\in E} f_{j\to p},$$

satisfazendo a primeira condição e, portanto, constituindo um fluxo $l$-arborescente que gera $T$. Este caso é ilustrado na Figura 2.

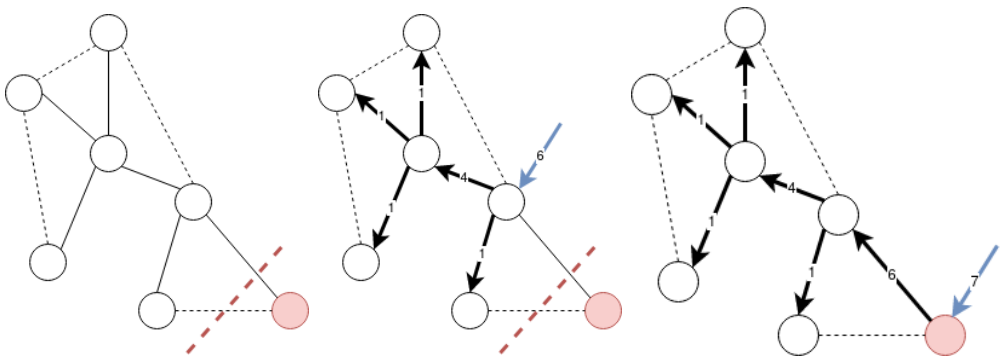

Figura 2 – O caso em que $l$ é definido como fonte de $T$.

Se algum outro vértice $s$ for definido como fonte (como na Figura 3), há um caminho $s = p_1, p_2, \ldots, p_k = l$. Tendo um fluxo $s$-arborescente que gera $T'$, garantido pela hipótese de indução para $|V| = n$, podemos construir o fluxo para $T$ ajustando os pesos do caminho na seguinte forma:

$$f_{p_i\to p_{i+1}} = f'_{p_i\to p_{i+1}} + 1, \text{ para } i = 1, 2, \ldots, k-1,$$

em que $f'_{i\to j}$ é o fluxo no arco $(i, j)$ da árvore $T'$.

Veja que todos os vértices obedecem às condições do fluxo:

- Para os vértices que não fazem parte do caminho, os fluxos de $T'$ já servem.
- Os vértices no meio do caminho recebem uma mercadoria a mais e oferecem uma mercadoria a mais, então a igualdade ainda vale.
- O vértice $s$ agora tem uma mercadoria a mais saindo, que corresponde a mercadoria que entra pelo número de vértices ter aumentado em um.
- Como o vértice $l$ não recebia nenhum fluxo, agora ele recebe exatamente um, que é o consumido. Isso já serve para satisfazer a condição, sem termos que nos preocupar em ajustar os que saem de $l$.

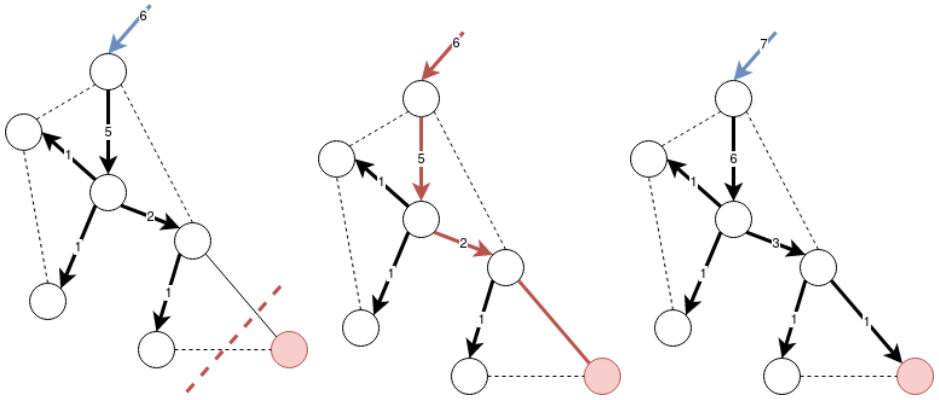

Figura 3 – O caso em que outro vértice é definido como fonte de $T$.

□

Para integrar a garantia da conectividade com o contágio, adicionamos restrições para que um par de arcos não seja bloqueado quando há fluxo sobre ele: para todo arco $(i, j)$, $nx_{i\to j} \geq f_{i\to j}$.

Esse teorema, assim como sua prova e o modelo, foi apresentado no Encontro Latino-Americano de Matemática e Aplicações (ELAMAP) 2021.

### 4.3.2 Método do fluxo

Desenvolvemos também um método alternativo, que não se preocupa com o valor específico do fluxo. Nós precisamos, entretanto, de uma fonte e potencialmente múltiplos sorvedouros. As restrições são de que, para cada vértice $i$, com exceção da fonte, o fluxo total entrando em um vértice deve ser igual ao saindo deste vértice, e pelo menos algum fluxo deve entrar em cada vértice. De outro modo, para todo vértice $i$:

$$0 < \sum_{(p,i)\in E} f_{p\to i} = \sum_{(i,p)\in E} f_{i\to p}.$$

Essas restrições são diferentes para a fonte $s$ e para qualquer sorvedouro $k$:

$$\sum_{(s,i)\in E} f_{s\to i} > 0 \text{ e } \sum_{(i,s)\in E} f_{i\to s} = 0;$$

$$\sum_{(i,k)\in E} f_{i\to k} > 0 \text{ e } \sum_{(k,i)\in E} f_{k\to i} = 0.$$

Isso não é o suficiente para garantir que a solução dada será um único grafo conexo, pois podemos ter várias componentes conexas, como no exemplo simples na Figura 4, em que há algum fluxo maior que zero passando apenas pelas arestas vermelhas, e uma fonte e um sorvedouro na componente à esquerda:

Para resolver isso, quando recebemos uma solução do *solver*, checamos os componentes conexos. Caso haja mais de um, analisamos as arestas do grafo cujos vértices estão em componentes diferentes, e adicionamos uma restrição ao nosso modelo do grafo de

Figura 4 – Um grafo obedecendo as restrições, mas que pode gerar um grafo desconexo.

modo que pelo menos uma delas terá que ser incluída (ou seja, ter algum fluxo passando por ela) na próxima solução e tentamos de novo. Repetimos esse processo até que o grafo obtido na solução seja de fato conexo.

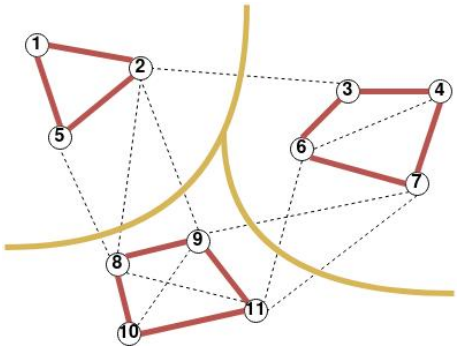


Figura 5 – Um grafo com três componentes conexas. Pelo menos uma das arestas que a curva amarela toca deve estar liberada na próxima iteração do grafo.

Na figura 5, vemos um exemplo com três componentes conexas. Nesse caso, adicionaríamos a restrição:

$$x_{2\to3} + x_{2\to8} + x_{2\to9} + x_{5\to8} + x_{6\to11} + x_{7\to9} + x_{7\to11} \geq 1.$$

Ao implementarmos o modelo em uma ferramenta de otimização, é interessante fornecer também uma solução viável inicial, que imediatamente dá ao nosso solver um limitante superior para o valor ótimo do problema. Na próxima seção, vamos discutir um modo de obter uma solução viável inicial.

## 4.4 Uma primeira solução viável e uma busca iterativa

Para iniciar o solver com um limitante superior, fornecemos uma árvore geradora inicial ao software e calculamos o número de infectados considerando tal árvore. Além disso, tentamos encontrar árvores geradoras que diminuem o número de infectados utilizando um processo de busca iterativa que vamos descrever a seguir.

Primeiramente, sendo $T$ uma árvore geradora de um grafo $G$, vamos definir uma árvore geradora adjacente de $T$ como uma árvore geradora que difere em apenas uma aresta.

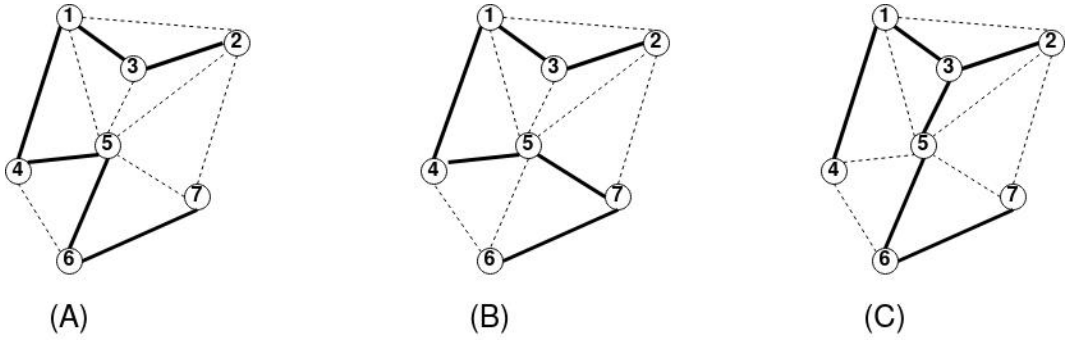


Figura 6 – Três árvores geradoras (as linhas sólidas) de um grafo (que contém também as linhas pontilhadas).

Na Figura 6, (B) é uma árvore adjacente a (A), pois elas têm todas as arestas em comum, exceto uma ($\{5, 6\}$ em (A) e $\{5, 7\}$ em (B)). Similarmente, a árvore (C) também é uma árvore adjacente a (A) (apenas $\{4, 5\}$ em (A) e $\{3, 5\}$ em (C) não têm correspondentes). Mas (B) e (C) não são adjacentes, pois $\{4, 5\}$ e $\{5, 7\}$ estão em (B), mas não em (C) e $\{3, 5\}$ e $\{5, 6\}$ estão em (C), mas não em (B).

Procedemos do seguinte modo, então:

Iniciamos com uma árvore geradora qualquer $T$. Buscamos entre o conjunto de todas as árvore geradoras adjacentes de $T$ e escolhemos a árvore $T'$ que minimiza o número de infectados. Se o valor de infectados de $T'$ for menor que o de $T$, $T'$ toma o lugar de $T$ e repetimos o procedimento. Caso contrário, paramos.

Podemos repetir esse processo por um número dado de iterações, definindo vários mínimos locais. Naturalmente, mantemos apenas o que levar ao menor número de infectados.

O processo específico de busca entre as árvores adjacentes merece um detalhamento maior. Dada uma árvore geradora $T$, escolhemos uma aresta do grafo que não esteja em $T$ e a adicionamos em $T$. Isso forma um ciclo ($T$ mais a aresta inserida). Para cada aresta nesse ciclo, a retiramos da árvore e recalculamos o número de infectados, salvando a árvore que o minimize. Repetimos o processo, escolhendo outra aresta até que todas as arestas que não estavam na árvore original tenham sido escolhidas.

Em relação à complexidade, para analisar todas as árvores adjacentes, temos $|E| - (n-1) \in O(n^2)$ opções de arestas que podemos adicionar. Para cada uma dessas, testamos a retirada de cada uma das arestas do ciclo que se formou, ou seja, até $n-1 \in O(n)$ arestas. Como calculamos o número de infectados para cada vértice, temos $O(n)$ cálculos. Com isso, para cada troca de árvore, temos $O(n^4)$ operações. O número de árvores pela quais passamos, contanto, é mais difícil de estimar e necessitaria de uma investigação mais profunda. Se $tree(G)$ é o número de árvores geradoras do grafo $G$, porém, podemos estimar grosseiramente como no máximo $O(\text{número de iterações} \times tree(G) \times n^4)$.

Com o modelo com o qual trabalhamos definido, nos preocupamos em definir as

ferramentas a serem usadas e a implementação de fato, as quais são discutidas no capítulo a seguir.

# 5 Implementação

O modelo apresentado foi implementado em Python 3, com uma classe representando o grafo do modelo e seu comportamento. Em particular, também podemos criar árvores geradoras do grafo para os testes iniciais.

A busca iterada foi implementada utilizando vários métodos auxiliares. Por exemplo, métodos para adicionar arcos no grafo, encontrar ciclos, e testar diferentes árvores em número de infectados.

Para resolver nosso problema de otimização utilizamos o solver SCIP, através do JuMP, uma linguagem de modelagem para programação matemática embutida em Julia (DUNNING *et al.*, 2017). O SCIP foi escolhido por ser um software com licença acadêmica, além de ser compatível com o JuMP e permitir restrições não-lineares.

A princípio, nossa implementação se preocupa em minimizar o número de infectados apenas de um tempo inicial até o próximo tempo. Como começamos de um tempo inicial sem recuperados, eles não interferem no resultado em apenas uma iteração, e portanto na implementação mantivemos apenas os suscetíveis e infectados. Os códigos desenvolvidos podem ser encontrados em Ataide (2022).

Como um ponto de partida, fornecemos ao *solver* uma solução: uma árvore geradora que tenha o menor número de infectados, obtida no processo de busca iterativa (passamos tanto o número de infectados quanto a estrutura). A ideia por trás disso é que, se o otimizador já tem uma solução, ele potencialmente pode evitar investigar soluções dos subproblemas que tiverem um limitante inferior maior do que a solução atual. Além disso, o *solver* consegue obter o dual, permitindo estimar a distância máxima de um solução ótima. Tendo definido os cálculos necessários para encontrar os novos infectados, adicionamos a função objetivo, que corresponde à essa expressão.

Ainda precisamos adicionar as restrições para que as soluções sejam conexas. Usamos a abordagem das mercadorias, criando as variáveis associada com o fluxo e as relacionando do modo já explicado.

Para avaliarmos nosso modelo, naturalmente precisamos de instâncias de grafos. Alguns grafos são aleatórios e outros são simplificações da topologia real de uma cidade. Os grafos que utilizamos como instância neste trabalho são detalhados na seção a seguir.

## 5.1 Grafos usados

### 5.1.1 Geração de grafos aleatórios

Os primeiros testes realizados por este trabalho foram sobre grafos conexos não-dirigidos com $n$ vértices gerados aleatoriamente. A construção de um grafo conexo aleatório $G$ foi feita da seguinte forma:

- Geramos um grafo completo $G'$ com $n$ vértices;
- Para que o grafo resultante $G$ seja conexo, rodamos um algoritmo para achar uma árvore geradora qualquer $T$ em $G'$, e copiamos tal árvore em $G$;
- Para cada aresta $e$ de $G' - T$, adicionamos $e$ em $G$ com uma probabilidade definida $p$;
- Depois, para cada aresta $e = \{i, j\}$, definimos o peso (o valor $\beta_{i,j}$) como

$$\beta_{i,j} = \frac{1}{\max(d(i), d(j)) + 1};$$

- Os números de suscetíveis e infectados são escolhidos previamente ou uma população de cada tipo é distribuída aleatoriamente pelos vértices.

A árvore geradora é definida de um modo similar ao algoritmo de Kruskal: a diferença é que as arestas são embaralhadas em uma ordem qualquer em vez de serem ordenadas por peso. As arestas são, então, analisadas uma por uma. Caso uma aresta não forme um ciclo, ela é adicionada à árvore geradora; caso contrário, ela é descartada.

O objetivo principal dos grafos aleatoriamente gerados foi testar a validade da implementação, especialmente no estágio inicial. Para obtermos resultados mais concretos, propomos construir um grafo baseado em dados reais de uma cidade. Os detalhes sobre a construção deste grafo são discutidos na próxima seção.

### 5.1.2 Grafo exemplo de Florianópolis

O grafo exemplo de Florianópolis foi baseado nos dados disponíveis no mapa em Microsoft (2021) que contém dados sobre a COVID-19 de acordo com uma divisão de bairros de Florianópolis. O mapa nos fornece números de casos confirmados, casos ativos, recuperados e óbitos. Este mapa não apresenta a população total de cada bairro.

Para obter a população de cada bairro, recorremos ao censo de 2010, mantido no banco de dados do IBGE e disponível em (TABELA, 2021). Apesar de também ser organizado em bairros, eles não correspondem exatamente aos bairros do mapa anterior.

A solução proposta foi agrupá-los de acordo com a posição geográfica aproximada correspondente. Para os bairros que não tiveram nenhum correspondente claro, estimamos a população como proporcional ao número de casos. Essa população se torna o número de pessoas suscetíveis em cada um dos bairros, enquanto o número de casos ativos em cada região se torna o número de infectados. Mais detalhes da construção do grafo podem ser encontrados em Ataide (2022).

Para diminuir o valor de vértices (e assim o número de variáveis para serem escolhidas na otimização), decidimos agrupar os 49 bairros em 12 regiões, cada uma sendo composta de bairros próximos.

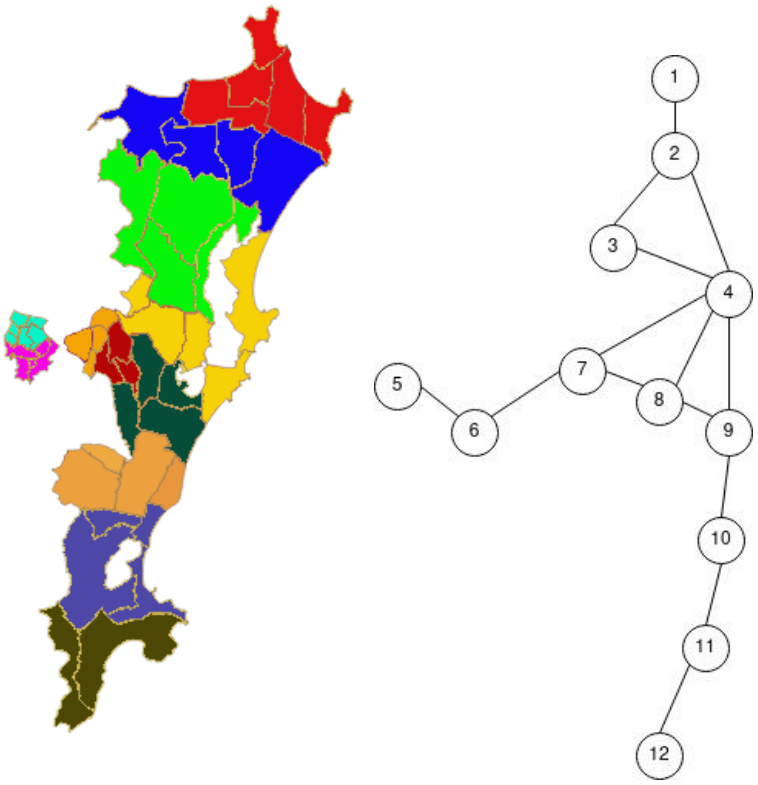


Figura 7 – O nosso grafo (adaptado de Microsoft (2021)) e um grafo esquemático.

Também criamos um grafo um pouco mais complexo, apresentado na figura 8, para testarmos até que ponto o Julia exploraria.

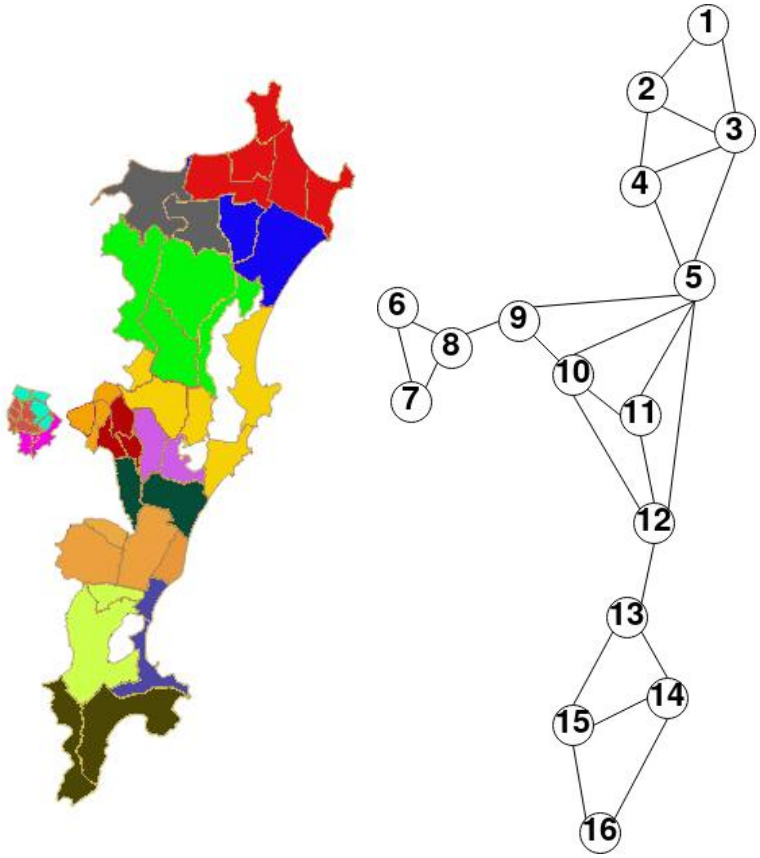


Figura 8 – O grafo com 16 regiões (adaptado de Microsoft (2021)) e um grafo esquemático.

Para definir o fluxo de pessoas, escolhemos cada valor $\beta_{i\to j}$ como a proporção entre a população de $j$ e a soma das populações de $i$ e de todos os vértices adjacentes a $i$. Isso tem a intenção de capturar o movimento maior para bairros mais populosos.

# 6 Experimentos

Neste capítulo, vamos expôr experimentos específicos que executamos, além de discutir os resultados obtidos.

## 6.1 Árvore geradora mínima

Para grafos em que os betas são simétricos, é natural pensar na árvore geradora mínima como uma possível solução, como pesos menores correspondem a menos pessoas se movimentando. No entanto, ela não oferece sempre uma solução ótima, particularmente quando um dos vértices é muito desbalanceado em termos de população, como no exemplo na Figura 10. A árvore geradora mínima, nesse caso, ocasiona um valor de infectados maior do que as outras duas árvores geradoras.

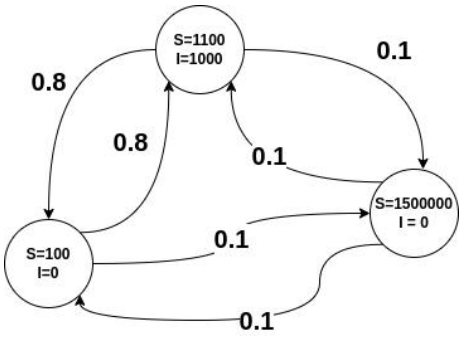


Figura 9 – Um exemplo de grafo em que a árvore geradora mínima não resulta no menor número de infectados.

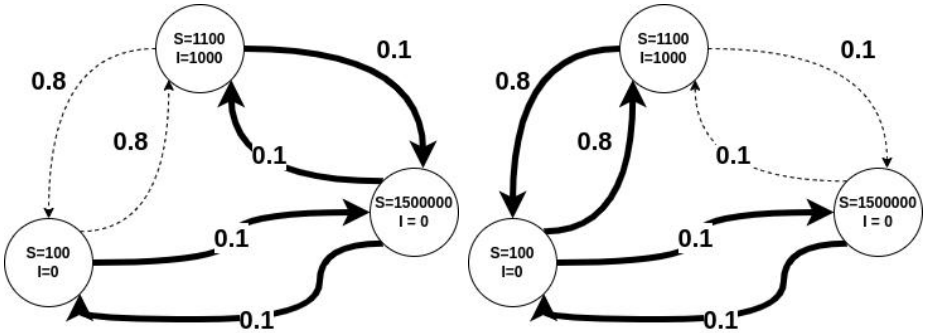


Figura 10 – A árvore a esquerda resulta em cerca de 1497 infectados, enquanto a da direita, que não é a com peso mínimo, retorna cerca de 1457.

Para grafos em que os betas não são simétricos, a definição de escolha das arestas é ainda mais complexa, visto que os valores beta de dois arcos que correspondem a uma mesma aresta podem ser bastante diferentes.

Apesar de não ser uma solução ótima, isso pode ser compensado pela facilidade de obter uma árvore geradora mínima.

## 6.2 Testes com o grafo de Florianópolis de 12 regiões

Os testes foram executados em um Notebook Aspire ES1-572 Acer, com CPU Intel(R) Core(TM) i3-6006u com frequência de 2.00GHz e 4GB de memória RAM. Em relação ao software, usamos Python versão 3.9.9 e Julia versão 1.7.0 com JuMP versão 0.22.1.

Com os bairros agrupados em 12 regiões, rodamos a busca iterativa com 2.000 iterações (ou seja, a partir de 2000 árvores geradoras aleatórias). O resultado obtido está representado na Figura 11.

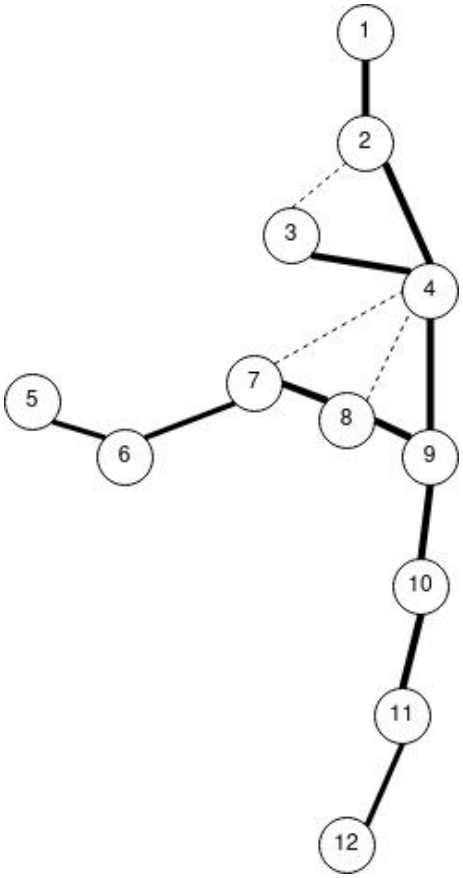


Figura 11 – A árvore que a busca iterativa retorna, com o valor de infectados esperado de 622.281867.

Apesar de ser um grafo simples, o *solver* utilizado tem dificuldade em melhorar a solução encontrada ou provar que ela é ótima.

Considerando esta instância em específico e para diminuir o número de variáveis implementadas, perceba que algumas conexões entre regiões (as correspondentes às arestas mais grossas na figura 12) precisam estar em qualquer solução viável para manter a

conexidade da cidade. Desse modo, inserimos restrições que garantem a escolha dos arcos dessas arestas e um fluxo apropriado.

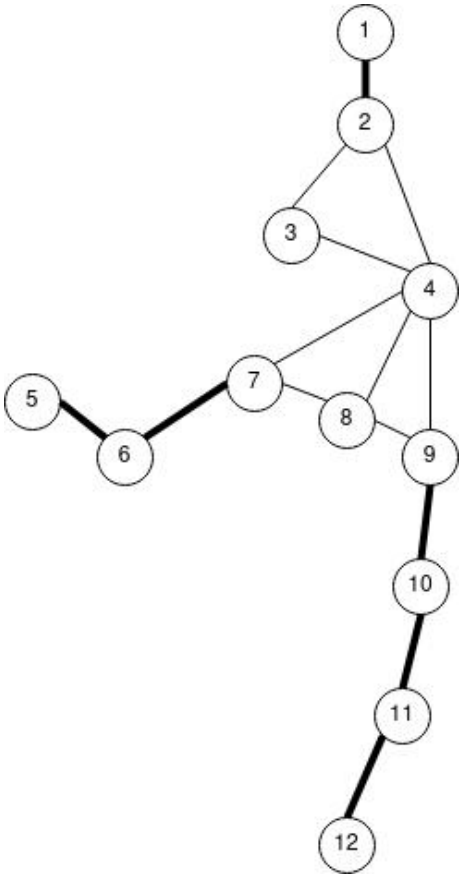


Figura 12 – O esquema do grafo de Florianópolis. As arestas $\{1, 2\}$, $\{5, 6\}$, $\{6, 7\}$, $\{9, 10\}$, $\{10, 11\}$ e $\{11, 12\}$ sempre são parte de qualquer solução.

Além disso, dado o número de iterações da nossa busca iterativa, podemos supôr que a solução já obtida é a melhor árvore e forçar o nosso programa de otimização a não ser uma árvore adicionando ciclos nas restrições. Note que, com isso, as árvores iniciais não são mais válidas. A figura 13 ilustra os quatro ciclos testados.

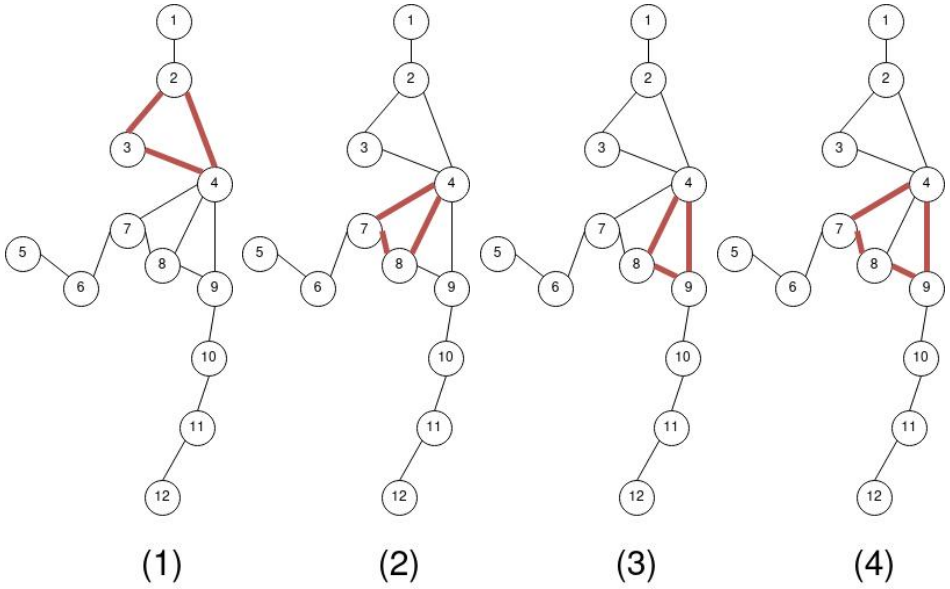


Figura 13 – Os quatro ciclos que testamos.

As quantidades de variáveis e restrições para cada caso são mostradas na tabela 1.

| Entrada | Binárias | Inteiras | Contínuas | Total | Restrições |
|---|---|---|---|---|---|
| Melhor árvore | 13 | 16 | 30 | 59 | 57 |
| Ciclo (1) | 7 | 16 | 17 | 40 | 35 |
| Ciclo (2) | 6 | 16 | 17 | 39 | 34 |
| Ciclo (3) | 6 | 16 | 17 | 39 | 34 |
| Ciclo (4) | 4 | 16 | 13 | 33 | 27 |

Tabela 1 – Tabela resumindo o número de variáveis em cada uma das instâncias que rodamos.

Os resultados obtidos com as otimizações são expressos na tabela 2. Cada instância executou até o limite de um milhão de nós explorados. O gap é dado em porcentagem aqui, e corresponde a $\frac{p-d}{p}$, sendo que $p$ é o limite encontrado do primal e $d$ o limite encontrado do dual.

É esperado que cada valor obtido tenha algum erro acumulado devido a precisão de

| Entrada | Status | Gap | Tempo | Valor obtido |
|---|---|---|---|---|
| Melhor árvore | Limite de nós alcançado | 5.07% | 63.58s | 622.236065 |
| Ciclo (1) | Limite de nós alcançado | 2.34% | 132.01s | 622.260109 |
| Ciclo (2) | Limite de nós alcançado | 3.70% | 228.73s | 622.247581 |
| Ciclo (3) | Solução ótima encontrada | 0.0% | 0.43s | 622.286727 |
| Ciclo (4) | Solução ótima encontrada | 0.0% | 5.4s | 622.284740 |

Tabela 2 – Resumo da saída do Julia para os grafos testados.

| Teste | Valor obtido | Valor empírico |
|---|---|---|
| Melhor árvore | 622.236065 | 622.281867 |
| Ciclo (1) | 622.260109 | 622.287180 |
| Ciclo (2) | 622.247581 | 622.287725 |
| Ciclo (3) | 622.286727 | 622.286727 |
| Ciclo (4) | 622.284740 | 622.284740 |

Tabela 3 – Tabela mostrando a diferença entre os valores obtidos e os valores reais, calculados com o grafo retornado pelo Julia

64 bits. Entretanto, alguns acumularam erros muito altos. Nós validamos os grafos dados como resultado pelo otimizador calculando o número de infectados independentemente. Os resultados na tabela 3 mostram que alguns valores desviaram mais de um centésimo do valor real, sendo que a melhor solução continua sendo a árvore resultante da busca iterativa.

## 6.3 Testes com o grafo de Florianópolis de 16 regiões

Já para o grafo de 16 regiões, a busca iterativa nos resultou na árvore na figura 14.

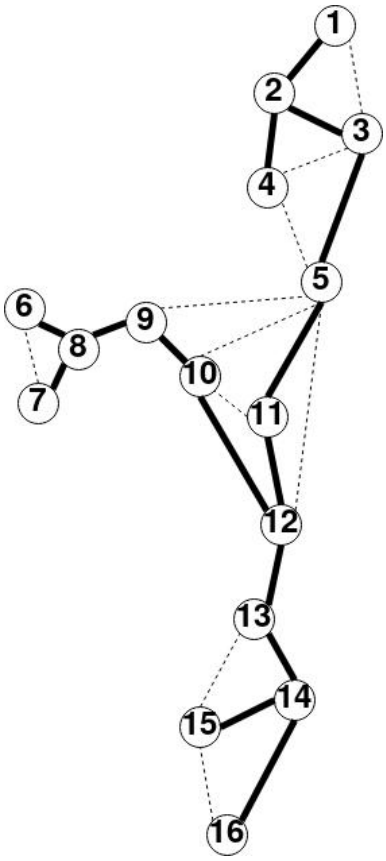


Figura 14 – A árvore obtida na busca iterativa, com número de infectados esperado 622.279198.

Passamos o grafo e essa árvore para o Julia e executamos por um total de quatro milhões de nós. A pré-otimização do Julia retornou um total de 162 variáveis (36 binárias, 49 inteiras e 77 contínuas) e 153 restrições. Depois de executar por 1073s, o programa retornou a árvore na figura 15.

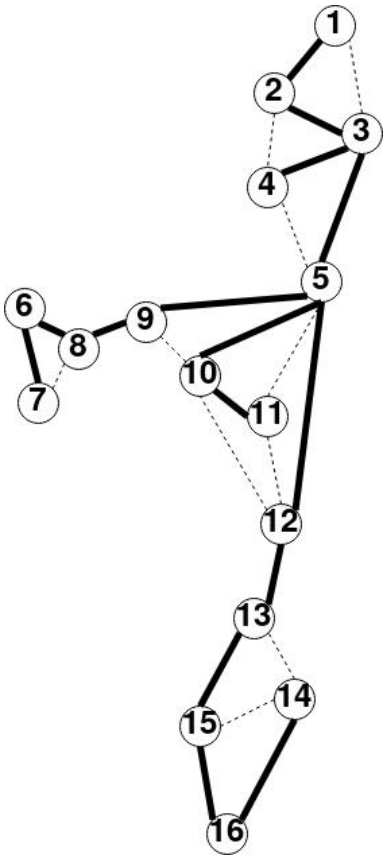


Figura 15 – Árvore retornada pelo Julia, acusando um gap de 14.24%.

O valor ótimo retornado foi 622.238778; entretanto, ao calcularmos o número de infectados da árvore da figura 14, obtemos 622.287136, novamente um valor diferente e mais alto do que o da melhor árvore.

Em geral, problemas de precisão e a grandeza do espaço de soluções dificultam a aplicação da programação não-linear tal como implementamos. Apesar de o *gap* forneça alguma informação do problema (e possa garantir que uma solução é ótima), heurísticas como a da busca iterativa se mostraram muito mais promissores em de fato encontrar uma solução relativamente boa.

# 7 Conclusões e trabalhos futuros

Neste trabalho, expandimos um modelo de contágio em uma topologia de bairros. Com o objetivo de minimizar o contágio, desenvolvemos uma formulação do modelo como um problema de programação não-linear; por fim, obtivemos alguns resultados sobre o problema usando um grafo baseado nos dados de Florianópolis. Os resultados ilustram a dificuldade em lidar com a complexidade do problema, além do esforço que existe na construção de instâncias baseadas em dados reais.

Aqui listamos algumas possibilidades de objetivos para progredir a partir deste trabalho:

- O foco neste trabalho para garantir a conexidade foi no método das mercadorias, pois o método do fluxo necessita de ferramentas mais complexas que possibilitam adicionar as restrições dinamicamente, o que não é possível com o SCIP através do JuMP. Uma implementação direta no SCIP (ou outro solver), entretanto, poderia aproveitar essas ferramentas para buscar soluções do problema com o método do fluxo.

- Buscar outras heurísticas para encontrar uma solução melhor antes de oferecer ao otimizador (por exemplo, uma que analise alguns ciclos, ou heurísticas que se baseiem na população de um vértice).

- É possível que o nosso problema seja NP-difícil, apesar de não termos uma prova disso. Seria interessante uma investigação mais profunda da complexidade, incluindo possivelmente uma redução de um problema NP-completo.

- Estamos usando algumas simplificações no programa de otimização, como eliminar os recuperados e considerar apenas uma iteração. Encontrar uma solução ótima para várias iterações requer teoricamente repensar o modelo por completo, enquanto adicionar os recuperados requer adicionar e modificar os cálculos.

# Referências


BOYD, S.; VANDERBERGHE, L. **Convex Optimization**. Cambridge, Reino Unido: Cambridge University Press, 2004.

CARVALHO, C. et al. **On the characterization of networks with multiple arc-disjoint branching flows**. [S.l.], 2020. Disponível em: %<https://hal.inria.fr/hal-03031759>.

CHANG, S. et al. Mobility network models of covid-19 explain inequities and inform reopening. **Nature**, v. 589, p. 82–87, 2021.

CHVÁTAL, V. **Linear Programming**. [S.l.]: W. H. Freeman and Company, 1983.

CONCATTO, F. et al. Genetic algorithm for epidemic mitigation by removing relationships. In: **GECCO '17**: Proceedings of the genetic and evolutionary computation conference. [S.l.: s.n.], 2017. p. 761–768.

DENG, Y.; SHEN, S.; VOROBEYCHIK, Y. Optimization methods for decision making in disease prevention and epidemic control. **Mathematical Biosciences**, v. 4, p. 213–227, 11 2013.

DUNNING, I.; HUCHETTE, J.; LUBIN, M. Jump: A modeling language for mathematical optimization. **SIAM Review**, v. 59, n. 2, p. 295–320, 2017.

FRANCO, Á. J. P. Epidemic model with restricted circulation and social distancing on some network topologies. In: **Cellular Automata**. Cham: Springer International Publishing, 2021. p. 261–264.

GUTIN, G. et al. The effect of social distancing on the reach of an epidemic in social networks. **J Econ Interact Coord**, v. 16, p. 629–647, 2021.

LIU, C. et al. A new SAIR model on complex networks for analysing the 2019 novel coronavirus (COVID-19). **Nonlinear Dyn**, v. 101, p. 1777–1787, 2020.

MICROSOFT Power BI. **Power BI**. Disponível em: %<https://app.powerbi.com/view?r=eyJrIjoiMzc5YmY0NmQtNTFkOS00ZDAxLWE2ZmQtOTZmZDkzM2M5NzAxIiwidCI6IjYyMTIxZmE1LWU3NTAtNDZlYS1hNjg0LTJhZmM2ZDIwYzYyYiJ9>. Acesso em: 28 ago. 2021.

PEIXOTO, P. S. et al. Modeling future spread of infections via mobile geolocation data and population dynamics. An application to COVID-19 in Brazil. **PLoS ONE**, v. 10, 2020.

POPA, A. **A decision support system for optimizing the cost of social distancing in order to stop the spread of COVID-19**. ArXiv. 2020.

SANTIAGO, R. et al. A new model and heuristic for infection minimization by cutting relationships. In: **International Conference on Neural Information Processing**. Barcelona, Spain: [s.n.], 2016. v. 9948, p. 500–508.

TABELA 1552: População residente, por situação do domicílio e sexo, segundo a forma de declaração de idade e a idade. **IBGE**. Disponível em: %<https://sidra.ibge.gov.br/tabela/1552>. Acesso em: 23 set. 2021.

ATAIDE, W. C. weslyca/TCC. **GitHub**, 23 fev. 2022. Disponível em: %<https://github.com/weslyca/TCC>.

WILSON, R. J. **Introduction to Graph Theory**. Harlow, Reino Unido: Longman Group, 1996.

WORD, D. et al. A nonlinear programming approach for estimation of transmission parameters in childhood infectious disease using a continuous time model. **Journal of the Royal Society, Interface / the Royal Society**, v. 9, p. 1983–97, 02 2012.

# Minimização do Contágio em uma Topologia com Programação Matemática

**Wesly C. Ataide**[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

wesly.ataide@grad.ufsc.br

***Abstract.*** *This paper intends to construct solutions to minimize the spread of an infection in a network of neighborhoods by restricting movement of their population, according to results from mathematical programming tools. We developed and implemented an infection model, as well as an equivalent non-linear programming formulation to be used in optimization tools. Furthermore, we constructed an instance of the problem using real data on COVID-19, and a method to find and improve solutions. Finally, we show some results that demonstrate the difficulty to deal with the complexity of the problem and with numerical imprecision.*

***Resumo.*** *Este trabalho tem por objetivo construir soluções para a minimização de contágio em uma topologia de bairros restringindo a movimentação da população entre locais. A resolução possui como foco as ferramentas da programação matemática. Desenvolvemos e implementamos um modelo de contágio, assim como uma formulação de programação não-linear equivalente a ser resolvida com ferramentas de otimização. Desenvolvemos uma instância de um grafo baseado em dados reais da COVID-19, além de um método para encontrar e melhorar soluções. Por fim, mostramos alguns resultados, demonstrando a dificuldade de lidar com a complexidade do problema e a imprecisão da ferramenta.*

## 1. Motivação

Depois dos primeiros casos de COVID-19, doença causada pelo coronavírus SARS-CoV-2, serem registrados em dezembro de 2019 na China, a infecção se espalhou rapidamente, representando um grande perigo para sistemas de saúde e para a economia mundial. As medidas adotadas por muitos países em resposta consistiram em restrições de movimento, limitando o funcionamento de estabelecimentos e incentivando seus cidadãos a ficarem em casa, com o objetivo de reduzir o contato entre indivíduos e, desse modo, frear o contágio. Com essas ações, entretanto, vem a questão de quando é seguro reabrir, quais lugares podem ser reabertos e quanta atividade pode ser permitida [Chang et al. 2021].

Para responder essas questões, modelos epidemiológicos se tornaram cruciais. Com um volume grande de dados, eles podem identificar grupos socioeconômicos mais vulneráveis, apontar classes de estabelecimentos mais propensas a espalhar a infecção e encontrar padrões para maximizar a efetividade de medidas de restrição [Chang et al. 2021]. Medidas que diminuem o contágio são os instrumentos mais importantes na luta contra o COVID-19 numa perspectiva social, pois tanto diminuem o

número de infectados como ajudam a garantir o tratamento para pessoas que se infectarem [Chang et al. 2021]. Com isso, previsões se tornaram fundamentais para reduzir os impactos negativos da pandemia tanto numa esfera social quanto numa esfera econômica [Liu et al. 2020], ao passo que guiam as políticas públicas adotadas.

## 2. Modelo

A nossa modelagem do problema, baseada em [Franco 2021], consiste na minimização do contágio em uma rede com bairros interconectados por estradas, representada por um grafo em que os bairros são os vértices (com uma população associada) e cada estrada entre um bairro $i$ e um bairro $j$ é um par de arcos $(i, j)$ e $(j, i)$. Nas próximas seções, vamos entrar em detalhes sobre cada aspecto relevante do modelo.

### 2.1. População dos vértices

Em relação ao modelo epidemiológico, escolhemos trabalhar com o modelo SIR, que divide a população em três grupos: suscetíveis, que podem se infectar; infectados, que podem transmitir; e recuperados, que não transmitem nem podem ser infectados. Levando isso em conta, a população de cada vértice $i$ para cada instante $t$ consiste no número de pessoas suscetíveis $S_i^t$, infectadas $I_i^t$ e recuperadas $R_i^t$.

O movimento entre vértices também é modelado, porém é restrito de modo que pessoas do vértice $i$ apenas podem visitar vértices adjacentes de $i$. Ainda definimos para cada arco $(i, j)$ os valores $\beta_{i\to j}$ que representam a porcentagem de pessoas que residem no vértice $i$ e visitam o vértice $j$, ou seja, que "utilizam a estrada $(i, j)$". Também é associado a cada arco $(i, j)$ uma variável binária $x_{i\to j}$ (que restringimos para ter o mesmo valor de $x_{j\to i}$), que indica que o tráfego está bloqueado se $x_{i\to j} = 0$ ou que está livre se $x_{i\to j} = 1$.

É relevante notar que essa construção faz com que o grafo seja mais facilmente analisado como um grafo dirigido da perspectiva dos valores $\beta$ e como um grafo não-dirigido da perspectiva das variáveis $x$. Desse modo, quando nos referimos a uma árvore ou a componentes conexas nesse grafo, estamos o analisando do modo não-dirigido.

As ruas bloqueadas e liberadas criam uma variação no número de pessoas em cada um dos vértices, afetando o contágio, como é explicado a seguir.

### 2.2. Dinâmica do contágio

Para analisar a dinâmica do contágio, observa-se que, em cada instante $t$, as pessoas circulando em cada vértice $i$ se dividem em dois grupos: as que vêm de um vértice vizinho a $i$ e as que moram e ficaram em $i$.

Para o primeiro grupo, para cada vértice vizinho, o número de pessoas vindo obedece às seguintes fórmulas:

$$S_{j\to i}^t = \beta_{j\to i} x_{j\to i}^t S_j^t;$$
$$I_{j\to i}^t = \beta_{j\to i} x_{j\to i}^t I_j^t;$$
$$R_{j\to i}^t = \beta_{j\to i} x_{j\to i}^t R_j^t.$$

Já o segundo grupo consiste das pessoas que não saíram do vértice $i$ e, desse modo, basta subtrair a porcentagem de pessoas que saiu:

$$S^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j}x^t_{i\to j}))S^t_i;$$
$$I^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j}x^t_{i\to j}))I^t_i;$$
$$R^t_{i\to i} = (1 - \sum_{j\in\chi(i)}(\beta_{i\to j}x^t_{i\to j}))R^t_i.$$

É relevante observar que se uma estrada é bloqueada, a porcentagem das pessoas que tinham intenção de usá-la é acrescentada no vértice onde as pessoas moram.

Com isso, o número de pessoas circulando de cada categoria é dado por:

$$\dot{S}^t_i = S^t_{i\to i} + \sum_{j\in\chi(i)} S^t_{j\to i}$$
$$\dot{I}^t_i = I^t_{i\to i} + \sum_{j\in\chi(i)} I^t_{j\to i}$$
$$\dot{R}^t_i = R^t_{i\to i} + \sum_{j\in\chi(i)} R^t_{j\to i}$$

O objetivo é minimizar o número de infectados ao longo do tempo. É necessário considerar para cada vértice e instante o número de pessoas que se recuperam e o número de novos infectados, que é uma razão do número de encontros entre pessoas suscetíveis e infectadas:

$$S^{t+1}_i = S^t_i - vE^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i);$$
$$I^{t+1}_i = I^t_i - \xi I^t_i + vE^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i);$$
$$R^{t+1}_i = R^t_i + \xi I^t_i.$$

Como a porcentagem de infectados das pessoas circulando em um vértice $i$ é $\dot{I}^t_i\dot{S}^t_i + \dot{I}^t_i + \dot{R}^t_i$, tomamos esse valor como a proporção de suscetíveis que tem um encontro com um infectado nesse vértice. O número esperado de encontros para a população do vértice $i$ é, portanto,

$$E^t_i(\dot{S}^t_i, \dot{I}^t_i, \dot{R}^t_i) = S_{i\to i}\dot{I}^t_i\dot{S}^t_i + \dot{I}^t_i + \dot{R}^t_i + \sum_{j\in\chi(i)} S_{i\to j}\dot{I}^t_j\dot{S}^t_j + \dot{I}^t_j + \dot{R}^t_j.$$

Temos uma limitação sobre o modo como escolhemos bloquear as ruas, entretanto: devemos garantir que seja possível acessar todos os bairros, ou seja, o subgrafo induzido pelas ruas liberadas deve ser conexo. Na próxima seção, discutimos como garantir essa restrição no nosso modelo de programação não-linear.

### 2.3. Método das mercadorias

Um modo de garantir a conexidade (baseado em [Carvalho et al. 2020]) é através do fluxo com uma abordagem similar a de produtores e consumidores de mercadorias, com um vértice arbitrário $s$ escolhido como fonte e as restrições a seguir.

$$\sum_{(i,j)\in E} f_{i\to j} + 1 = \sum_{(j,i)\in E} f_{j\to i} \text{ para cada vértice } i \neq s;$$
$$\sum_{(s,j)\in E} f_{s\to j} = n - 1;$$

em que $f_{i\to j}$ representa o fluxo que passa pelo arco $(i, j)$, e $n$ é o número de vértices do grafo.

Uma escolha das variáveis $f$ que satisfaça essas condições é um fluxo $s$-arborescente. Note que cada fluxo $s$-arborescente induz uma árvore geradora no grafo ao tomarmos como arestas todo par de arcos $(i, j)$ em que $f_{i\to j} > 0$ ou $f_{j\to i} > 0$. Como nossos arcos vêm em pares, a estrutura do grafo essencialmente se mantém.

Para integrar a garantia da conectividade com o contágio, adicionamos restrições para que um par de arcos não seja bloqueado quando há fluxo sobre ele: para todo arco $(i, j)$, $nx_{i\to j} \geq f_{i\to j}$.

## 3. Uma primeira solução viável com uma busca iterativa

Para iniciar o solver com um limitante superior, fornecemos uma árvore geradora inicial ao software e calculamos o número de infectados considerando tal árvore. Além disso, tentamos encontrar árvores geradoras que diminuem o número de infectados utilizando um processo de busca iterativa que vamos descrever a seguir.

Primeiramente, sendo $T$ uma árvore geradora de um grafo $G$, vamos definir uma árvore geradora adjacente de $T$ como uma árvore geradora que difere em apenas uma aresta.

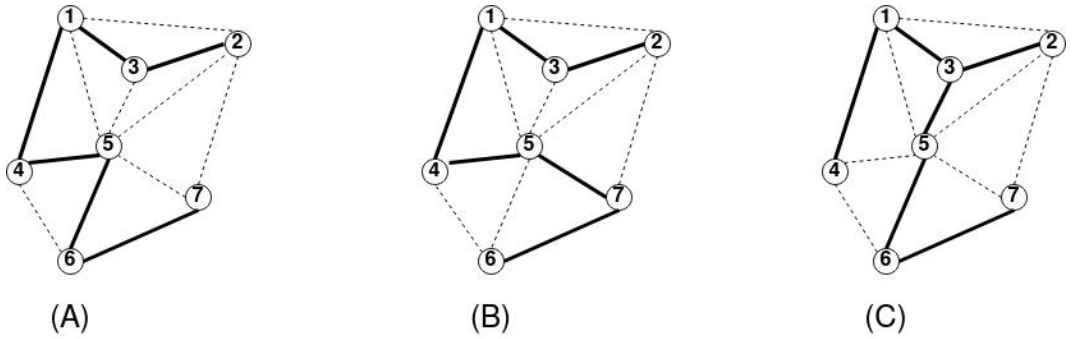


**Figura 1. Três árvores geradoras (as linhas sólidas) de um grafo (que contém também as linhas pontilhadas).**

Na Figura 1, (B) é uma árvore adjacente a (A), pois elas têm todas as arestas em comum, exceto uma ($\{5, 6\}$ em (A) e $\{5, 7\}$ em (B)). Similarmente, a árvore (C) também é uma árvore adjacente a (A) (apenas $\{4, 5\}$ em (A) e $\{3, 5\}$ em (C) não têm correspondentes). Mas (B) e (C) não são adjacentes, pois $\{4, 5\}$ e $\{5, 7\}$ estão em (B), mas não em (C) e $\{3, 5\}$ e $\{5, 6\}$ estão em (C), mas não em (B).

Procedemos do seguinte modo, então:

Iniciamos com uma árvore geradora qualquer $T$. Buscamos entre o conjunto de todas as árvore geradoras adjacentes de $T$ e escolhemos a árvore $T'$ que minimiza o número de infectados. Se o valor de infectados de $T'$ for menor que o de $T$, $T'$ toma o lugar de $T$ e repetimos o procedimento. Caso contrário, paramos.

Podemos repetir esse processo por um número dado de iterações, definindo vários mínimos locais. Naturalmente, mantemos apenas o que levar ao menor número de infectados.

O processo específico de busca entre as árvores adjacentes merece um detalhamento maior. Dada uma árvore geradora $T$, escolhemos uma aresta do grafo que não esteja em $T$ e a adicionamos em $T$. Isso forma um ciclo ($T$ mais a aresta inserida). Para cada aresta nesse ciclo, a retiramos da árvore e recalculamos o número de infectados, salvando a árvore que o minimize. Repetimos o processo, escolhendo outra aresta até que todas as arestas que não estavam na árvore original tenham sido escolhidas.

## 4. Construção de um grafo baseado em Florianópolis

Para obtermos resultados mais concretos, propomos construir um grafo baseado em dados reais de uma cidade. Construímos então um grafo exemplo de Florianópolis, baseado nos dados do covidômetro, um site disponibilizado pela prefeitura, que contém dados sobre a COVID-19 de acordo com uma divisão de bairros de Florianópolis. O mapa nos fornece números de casos confirmados, casos ativos, recuperados e óbitos. Este mapa não apresenta a população total de cada bairro.

Para obter a população de cada bairro, recorremos ao censo de 2010, mantido no banco de dados do IBGE. Apesar de também ser organizado em bairros, eles não correspondem exatamente aos bairros do mapa anterior. A solução proposta foi agrupá-los de acordo com a posição geográfica aproximada correspondente. Para os bairros de que não tiveram nenhum correspondente claro, estimamos a população como proporcional ao número de casos. Essa população se torna o número de pessoas suscetíveis em cada um dos bairros, enquanto o número de casos ativos se torna o número de infectados. Com essas informações, criamos dois grafos; um com 12 regiões (mais simples, para tentarmos encontrar uma solução ótima) e um com 16 regiões (mais complexo, com o objetivo de simplesmente melhorar a solução inicial). Ambos são ilustrados na figura 2.

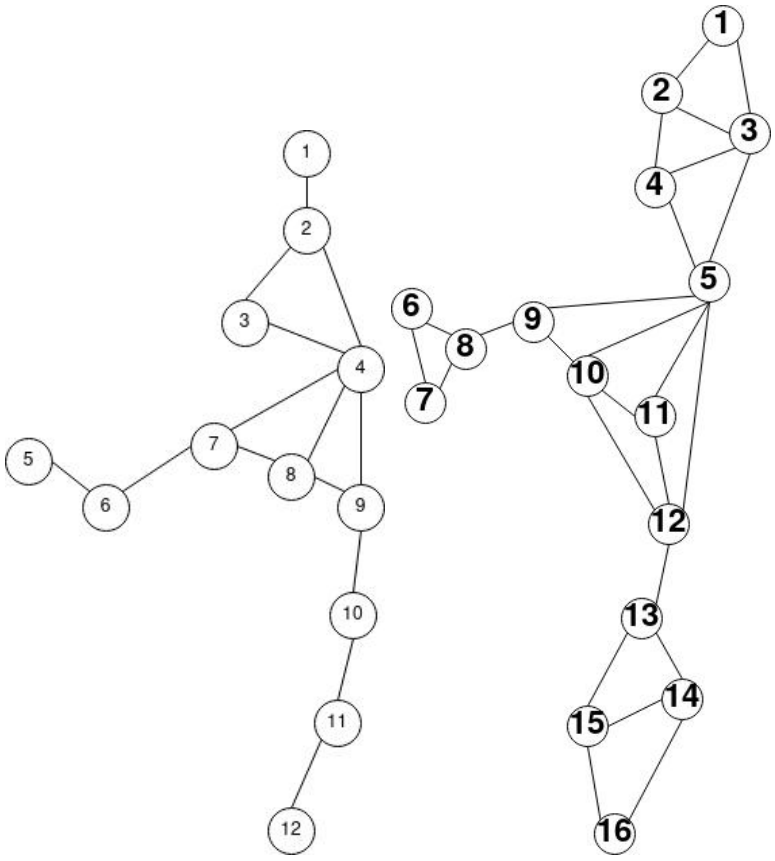


**Figura 2. Esquemático dos grafos construídos.**

## 5. Implementação

O modelo apresentado foi implementado em Python 3, com uma classe representando o grafo do modelo e seu comportamento. A busca iterada foi implementada utilizando

vários métodos auxiliares. Por exemplo, métodos para adicionar arcos no grafo, encontrar ciclos, e testar diferentes árvores em número de infectados. Para resolver nosso problema de otimização utilizamos o solver SCIP, através do JuMP, uma linguagem de modelagem para programação matemática embutida em Julia [Dunning et al. 2017]. O SCIP foi escolhido por ser um software com licença acadêmica, além de ser compatível com o JuMP e permitir restrições não-lineares.

A princípio, nossa implementação se preocupa em minimizar o número de infectados apenas de um tempo inicial até o próximo tempo. Como começamos de um tempo inicial sem recuperados, eles não interferem no resultado em apenas uma iteração, e portanto na implementação mantivemos apenas os suscetíveis e infectados.

Como um ponto de partida, fornecemos ao *solver* uma solução: uma árvore geradora que tenha o menor número de infectados, obtida no processo de busca iterativa (passamos tanto o número de infectados quanto a estrutura). A ideia por trás disso é que, se o otimizador já tem uma solução, ele potencialmente pode evitar investigar soluções dos subproblemas que tiverem um limitante inferior maior do que a solução atual. Além disso, o *solver* consegue obter o dual, permitindo estimar a distância máxima de um solução ótima. Tendo definido os cálculos necessários para encontrar os novos infectados, adicionamos a função objetivo, que corresponde à essa expressão.

Ainda precisamos adicionar as restrições para que as soluções sejam conexas. Usamos a abordagem das mercadorias, criando as variáveis associada com o fluxo e as relacionando do modo já explicado.

A implementação descrita pode ser encontrada no repositório GitHub do projeto (https://github.com/weslyca/TCC).

## 6. Resultados

Os testes foram executados em um Notebook Aspire ES1-572 Acer, com CPU Intel(R) Core(TM) i3-6006u com frequência de 2.00GHz e 4GB de memória RAM. Em relação ao software, usamos Python versão 3.9.9 e Julia versão 1.7.0 com JuMP versão 0.22.1. Primeiramente, passamos os grafos que construímos pelo processo de busca iterada, obtendo as árvores representadas na Figura 3.

Ao passarmos o grafo de 12 regiões pelo programa em Julia, observamos uma amplificação dos erros numéricos devido a precisão de 64 bits. Para o grafo de 12 regiões, o otimizador aparenta encontrar uma solução com número esperado de infectados igual a 622.236065, menor que o valor da árvore da busca iterada. Todavia, ao validarmos a árvore retornada, ela não é melhor do que a da busca iterada, retornando o valor de 622.281867.

Já para o grafo de 16 regiões, o valor ótimo retornado foi 622.238778; entretanto, ao calcularmos o número de infectados da árvore da figura 3, obtemos 622.287136, novamente um valor diferente e mais alto do que o da melhor árvore.

Em geral, problemas de precisão e a grandeza do espaço de soluções dificultam a aplicação da programação não-linear tal como implementamos. Em geral, heurísticas como a da busca iterativa se mostraram muito mais promissores em de fato encontrar uma solução relativamente boa.

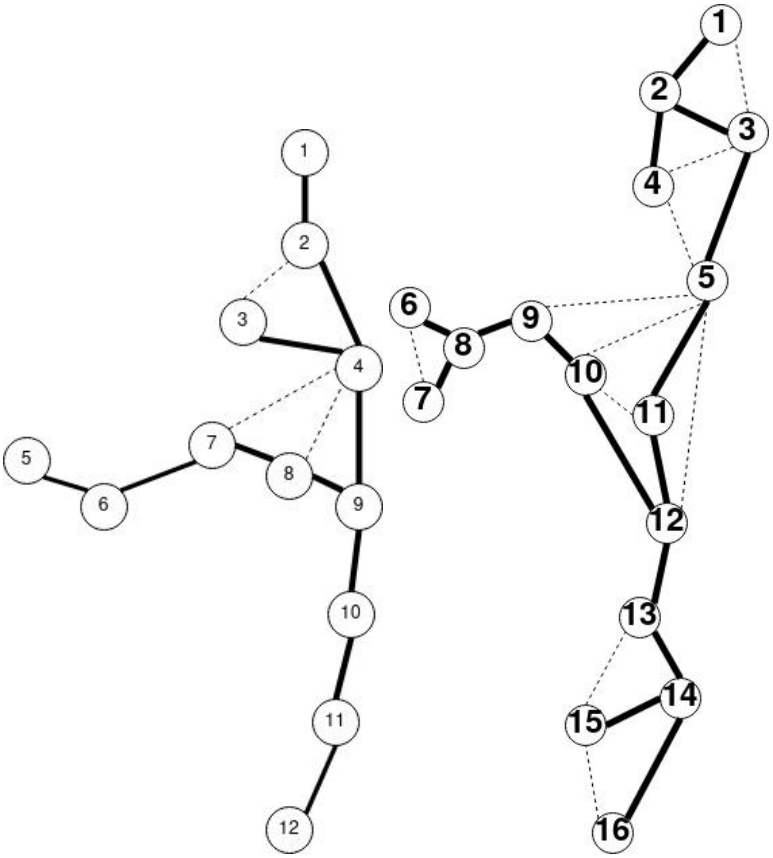


**Figura 3. As árvores (linhas sólidas) obtidas pela busca iterativa após 2000 iterações. Essas árvores serão dadas ao solver como solução inicial.**

## Referências


Carvalho, C., Costa, J., Sales, C. L., Lopes, R., Maia de Oliveira, A. K., and Nisse, N. (2020). On the characterization of networks with multiple arc-disjoint branching flows. Research report, UFC ; INRIA ; CNRS ; Université Côte d'Azur ; I3S ; LIRMM ; Université de Montpellier.

Chang, S., Pierson, E., Koh, P. W., Gerardin, J., Redbird, B., and Grusky, D. Leskovec, J. (2021). Mobility network models of covid-19 explain inequities and inform reopening. *Nature*, 589:82–87.

Dunning, I., Huchette, J., and Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320.

Franco, Á. J. P. (2021). Epidemic model with restricted circulation and social distancing on some network topologies. In *Cellular Automata*, pages 261–264, Cham. Springer International Publishing.

Liu, C., Wu, Niu, R., Wu, X., and Fan, R. (2020). A new sair model on complex networks for analysing the 2019 novel coronavirus (covid-19). *Nonlinear Dyn*, 101:1777–1787.