Problema da Fake News com Algoritmo Genético

Relatório Trabalho Final GCC118 - Programação Matemática

Rhuan Campideli Borges - 10A - 202210075

LAVRAS - MG

2025

Sumário

1 Introdução

O objetivo deste trabalho proposto e implementado é o estudo mais aprofundado de “metaheurísticas” dentro do contexto da disciplina de Programação Matemática (GCC118). Trata-se de uma jornada de aprendizagem, perpassando pelos conceitos estudados na disciplina no que tange à modelagem matemática de problemas reais de otimização, bem como sua resolução via programação.

Desse modo, o presente relatório apresenta uma proposta de solução para o Problema da Fake News, que será enunciado na próxima seção. Nesse cenário, o problema foi formulado matematicamente como um problema de programação inteira, resolvido através do solver Gurobi e, por fim, resolvido através do Algoritmo Genético, uma metaheurística evolutiva.

Na seção 2, é apresentado o enunciado do Problema da Fake News, bem como o modelo matemático proposto para representá-lo. Na seção 3, mostra-se a solução em Algoritmo Genético proposta para resolver o problema, apresentando detalhes e justificativas para a escolhas realizadas (parâmetros, geração de população inicial e critério de parada). Na seção 4, são apresentados os resultados das execuções em formato de tabela e gráficos, a fim de se realizar uma análise visual do desempenho dos algoritmos. Na seção 5, sumariza-se o trabalho, apontando aprendizados e dificuldades apresentadas durante o processo. Por fim, na seção 6, encerra-se o presente relatório com referências bibliográficas, utilizadas para justificar algumas escolhas feitas durante a implementação da metaheurística.

2 Formulação Matemática

Conforme supracitado, a proposta deste trabalho é o estudo profundo das metaheurísticas, neste caso, do Algoritmo Genético. E, para exemplificar/ilustrar esse estudo, foi dado o chamado Problema da Fake News, enunciado a seguir:

“[...] Suponha que uma mensagem classificada como suspeita por meio de uma rede neural seja enviada através de um vértice $s \in V$ de uma rede modelada por um grafo direcionado $G = (V, E)$. O conjunto de vértices V corresponde aos servidores que processam as informações, enquanto os arcos $(u, v) \in E$ são os enlaces de comunicação entre os servidores u, v. Deseja-se reduzir o número de servidores que a notícia falsa pode se encontrar após um período de tempo igual a T.

A propagação de uma informação no enlace $(u, v) \in E$ possui tempo estimado igual a t_{uv}. O Setor de Tecnologia da Informação (STI) do governo federal possui α recursos (softwares auxiliares, por exemplo) tecnológicos que podem ser aplicados aos servidores para retardar o tempo de broadcast da fake news. Assim, caso o servidor u possua algum dos α recursos instalado, o tempo de transmissão da fake news no enlace (u, v) seria $t_{uv} + \delta$, em que $\delta > 0$ representa um atraso de propagação. Cada um dos $i = 1, \ldots, \alpha$ recursos fica disponível no instante β_i.

A solução do problema consiste em determinar a designação dos recursos aos servidores. Cada servidor receberá, no máximo, um recurso. Considera-se que um recurso i é alocado no menor tempo β_i. O objetivo definido é a minimização da quantidade de servidores que a fake news poderia chegar em um tempo menor ou igual a T.

Com essa descrição do problema, formulou-se o seguinte modelo matemático:

2.1 Variáveis

- $x_{iu} \in \{0, 1\}$: vale 1 se o recurso i foi alocado ao servidor u; 0 caso contrário;
- $y_{uf} \in \{0, 1\}$: vale 1 se o servidor u foi atingido pela fake news em tempo menor ou igual a T, no cenário em que a fake news partiu do servidor f; 0 caso contrário;
- $t_{vf} \in \mathbb{Z}$: tempo mínimo para o servidor v ser atingido pela fake news, no cenário em que essa fake news partiu do servidor f;
- S: a maior quantidade de servidores atingidos pela fake news olhando todo o cenário geral.

Vale notar que o enunciado não especificou qual o vértice $s \in V$ de onde parte a fake news, portanto, é preciso analisar todos os vértices $s \in V$ possíveis.

2.2 Função Objetivo

A fake news pode partir de qualquer vértice pertencente ao conjunto de vértices V da instância. Desse modo, ao analisar o cenário, podemos obter as alcançabilidades de servidores diferentes a depender da fonte s a partir da qual partiu a fake news.

Para podermos distribuir os recursos, temos que analisar **o pior caso**, ou seja, o vértice s (fonte da fake news) que provoca maior alcançabilidade de vértices e, com isso, **nosso objetivo é minimizar essa alcançabilidade do pior caso**.

Portanto, a função objetivo é **minimizar a maior quantidade de servidores atingidos pela fake news**, considerando o pior caso de s

$$Min\ S$$

2.3 Restrições

- Determinar o valor de S para cada fonte s possível como a quantidade de servidores atingidos pela fake news no contexto em que ela partiu de s. Essa restrição determina o pior caso:

$$S \geq \sum_{u \in V} y_{us}\ ,\ \forall s\ \in V$$

- O tempo em que um vértice v é alcançado pela fake news é menor ou igual a T. Essas duas restrições forçam $y_{vs} = 1$ se o tempo $t_{vs} \leq T$ e $y_{vs} = 0$ se o tempo $t_{vs} \geq T + 1$.

$$(1 - y_{vs})(T + 1) \leq t_{vs}$$
$$t_{vs} \leq T y_{vs} + M(1 - y_{vs})$$
$$\forall v,\ s\ \in V$$

- O tempo em que um vértice v é alcançado pela fake news é calculado pelo tempo em que seu predecessor u é alcançado somado ao tempo gasto no enlace, bem como a um valor de δ se algum dos i recursos foi alocado no vértice u:

$$t_{vs} \leq t_{us} + t_{uv} + \delta \sum_{i=1}^{\alpha} x_{iu}\ ,\ \forall (u,v)\ \in E$$
$$\forall s\ \in V$$

- Para cada vértice, só pode ter no máximo um dos recursos alocados nele:

$$\sum_{i=1}^{\alpha} x_{iu} \leq 1,\ \forall u\ \in V$$

- Para cada recurso, só pode ter no máximo um dos vértices com ele alocado:

$$\sum_{u \in V} x_{iu} \leq 1, \forall i = 1, \ldots, \alpha$$

- Um recurso i só pode ser alocado a um servidor u se o tempo em que esse servidor u foi alcançado for superior ao instante β_i em que o recurso i fica disponível:

$$t_{us} \geq \sum_{i=1}^{\alpha} x_{iu} \beta_i, \forall u, s \in V$$

- No cenário em que a fake news partiu do servidor s, o valor de y para o próprio vértice s é igual a 1 e o tempo em que esse vértice s é alcançado é 0:

$$y_{ss} = 1$$
$$t_{ss} = 0$$
$$\forall s \in V$$

3 Descrição da Solução

A partir disso, a modelagem realizada foi implementada na linguagem Python utilizando o solver Gurobi, cuja sintaxe se assemelha ao que foi descrito matematicamente.

Junto ao enunciado do trabalho, consta uma pasta com 10 instâncias do problema - para fins de teste - e seus respectivos valores de solução ótima (valores referência). No entanto, encontrou-se um problema durante a implementação e os testes da solução exata do Problema da Fake News: o formato dessas instâncias. Observou-se que nos arquivos de instâncias existe um dado valor n que representa as dimensões de uma grade $n \times n$. Porém, na listagem dos pontos da grade, existem valores que ultrapassam os limites da própria grade, a qual supostamente deveria ser regular.

Como o tempo para realização deste trabalho não permitiu, decidiu-se ignorar uma tentativa de implementação de regularização dessa grade, mapeando os pontos corretamente aos vértices. Com isso, considerou-se cada ponto descrito como um vértice nomeado por suas próprias coordenadas, obtidas do arquivo. Isso trouxe, contudo, como consequência, uma disparidade entre os valores de solução

ótima obtidos com o solver e os valores passados como referência junto com as instâncias. Supõe-se que tal diferença se deve a esse problema do formato das instâncias.

Dessa maneira, na seção 4, ao apresentar os resultados da metaheurística, realizar-se-á a comparação tanto com os valores de solução ótima encontrados pelo solver quanto os valores de solução ótima de referência.

3.1 Algoritmo Genético

O Algoritmo Genético foi a metaheurística escolhida para ser estudada neste trabalho e consiste em um algoritmo evolutivo, inspirado na seleção natural que acontece na Biologia. Ou seja, dada uma determinada população de indivíduos, os mais aptos são selecionados, são submetidos a cruzamentos, havendo *crossover* dos genes a uma dada probabilidade, e sofrem mutação também a uma dada probabilidade. Com isso, a população de indivíduos vai evoluindo.

Como se trata de uma metaheurística, é um algoritmo geral, o qual pode ser adaptado para problemas específicos. Desse modo, a tarefa deste trabalho foi formular o Algoritmo Genético de forma adaptada para solucionar o Problema da Fake News.

3.1.1 Indivíduo

O primeiro passo é modelar a solução do Problema da Fake News como um indivíduo no Algoritmo Genético. Não se trata de uma tarefa difícil, visto que o próprio enunciado do problema diz que *“a solução do problema consiste em determinar a designação dos recursos aos servidores”*. Sendo assim, como a variável que representa a designação dos recursos aos servidores é de natureza binária, ela pode representar o indivíduo.

3.1.2 Mecanismos de Crossover, mutação e seleção

Encontrou-se o crossover de 2-pontos (Kora e Yadlapalli, 2017) como um mecanismo razoável de realizar cruzamentos em Algoritmo Genético, de modo a não perder performance, como no caso de mais de 2 pontos, e não ser tão simples, como no caso de um ponto. Desse modo, adotou-se esse mecanismo.

```python
# FUNÇÃO DO ALGORITMO GENÉTICO QUE APLICA O CRUZAMENTO/CROSSOVER EM DOIS INDIVIDUOS A UMA TAXA taxaCrossover
# É realizado o Crossover de 2 pontos
def cruzamento(individuo1, individuo2, taxaCrossOver):
    if random.random() < taxaCrossOver:
        # Converte o dicionário para uma lista de chaves
        chaves = list(individuo1.keys())

        # Sorteia dois pontos para o crossover
        ponto1, ponto2 = sorted(random.sample(range(len(chaves)), 2

        # Cria os filhos trocando os valores entre os dois pontos
        filho1 = {}
        filho2 = {}

        for i, chave in enumerate(chaves):
            # Troca os genes compreendidos entre os dois pontos sorteados e mantém iguais os fora do limite
            if ponto1 <= i < ponto2:
                filho1[chave] = individuo2[chave]
                filho2[chave] = individuo1[chave]
            else:
                filho1[chave] = individuo1[chave]
                filho2[chave] = individuo2[chave]

        return filho1, filho2
    return individuo1, individuo2
```

Pela própria natureza do problema de otimização em minimizar uma função objetivo, optou-se inicialmente por utilizar a seleção por elitismo na implementação do Algoritmo Genético. No entanto, durante a fase de testes do algoritmo, verificou-se que esse tipo de seleção calculava a aptidão para todos os indivíduos de uma população (a fim de ordená-los e escolher os mais aptos), o que custava uma quantidade significativa de tempo de processamento. Por uma questão de performance, então, optou-se finalmente pela seleção por torneio, usualmente utilizada.

```python
# FUNÇÃO DO ALGORITMO GENÉTICO QUE REALIZA A SELEÇÃO DOS INDIVÍDUOS MAIS APTOS DA POPULACAO
# É feita a seleção por torneio, escolhendo dois aleatórios e comparando-os quanto à aptidão, retornando o de menor aptidão (ou seja, menor S)
def selecao(populacao,dados):
  indices = random.sample(range(len(populacao)), 2)
  individuosTorneio = [populacao[indices[0]], populacao[indices[1]]]
  aptidoes = [aptidao(individuo, dados) for individuo in individuosTorneio]
  return individuosTorneio[aptidoes.index(min(aptidoes))]
```

Para evitar que indivíduos factíveis se transformem em não factíveis (por exemplo, designar recursos acima da possibilidade permitida), optou-se por um mecanismo swap para mutação, que apenas realiza a troca de posição de dois valores dentro do indivíduo.

```python
# FUNÇÃO DO ALGORITMO GENÉTICO QUE APLICA A MUTAÇÃO NOS INDIVÍDUOS A UMA TAXA taxaMutacao
# É realizada a Mutação SWAP, a fim de manter a quantidade de recursos alocados inalterada
def mutacao(individuo, taxaMutacao):
    if random.random() < taxaMutacao:
        indicesTroca = random.sample(list(individuo.keys()), 2)
        aux = individuo[indicesTroca[0]]
        individuo[indicesTroca[0]] = individuo[indicesTroca[1]]
        individuo[indicesTroca[1]] = aux
    return individuo
```

3.1.3 População Inicial, Critério de Parada e Parâmetros

De fato, foi constatado que realizar uma inicialização aleatória da população em um Algoritmo Genético gera piores resultados em relação a outras estratégias de inicialização (Vlašić, Đurasević, e Jakobović, 2019), no entanto, pelo fato de o escopo do trabalho se tratar do aprendizado de uma metaheurística, bem como pelo prazo curto, optou-se por fazer a inicialização aleatória dos indivíduos da população inicial.

```python
# FUNCAO DO ALGORITMO GENÉTICO QUE INICIALIZA ALEATORIAMENTE UMA POPULAÇÃO DE k INDIVÍDUOS
# Ela gera um número aleatório no intervalo [0,alfa] para alocar os recursos randomicamente e em quantidade factível
def inicializarPopulacao(k, dados):
    populacao = []

    for _ in range(k):

        # Número aleatório de bits 1 entre 0 e alfa (número total de recursos)
        numBits1 = random.randint(0, dados["alfa"])

        # Cria um indivíduo com "numBits1" bits iguais a 1, e o restante 0
        individuo = {f"{chave}": 0 for chave in dados["vertices"]}
        indices1 = random.sample(dados["vertices"], numBits1)  # Seleciona aleatoriamente as posições para 1

        for i in indices1:
            individuo[i] = 1

        populacao.append(individuo)

    return populacao
```

Da mesma forma, foi encontrado na literatura (Safe et al., 2004) um cálculo de limite superior para o número de iterações de um Algoritmo Genético de modo a ser adotado como critério de parada do tipo “número máximo de iterações do algoritmo”. Inicialmente, adotou-se essa técnica, porém, durante a implementação, encontraram-se alguns problemas, além do fato de que os próprios pesquisadores constatam no artigo que “o critério é teoricamente correto, mas com pouco interesse prático” (Safe et al., 2004). Com isso, fixou-se arbitrariamente o valor 10 como o número máximo de iterações/gerações de indivíduos para a execução do algoritmo.

```python
# FUNÇÃO PRINCIPAL DO ALGORITMO GENÉTICO, QUE CHAMA AS OUTRAS FUNÇÕES
# Critério de parada: número de iterações/gerações = 10
# Seleciona 4 indivíduos mais aptos e realiza 8 cruzamentos, a fim de manter um padrão de tamanho de população igual a 16
def otimizacaoAG(dados, tamanhoPopulacao, taxaCrossover, taxaMutacao):
    it = 1
    maxIt = 10
    populacao = inicializarPopulacao(tamanhoPopulacao, dados)
    SI = [aptidao(individuo, dados) for individuo in populacao] # Solução inicial (indivíduos da população inicial)
    while it <= maxIt:
        pais = {}
        for i in range(4):
            pais[i] = selecao(populacao, dados)
        novaPopulacao = []
        novaPopulacao += cruzamento(pais[0], pais[1], taxaCrossover)
        novaPopulacao += cruzamento(pais[0], pais[2], taxaCrossover)
        novaPopulacao += cruzamento(pais[0], pais[3], taxaCrossover)
        novaPopulacao += cruzamento(pais[1], pais[2], taxaCrossover)
        novaPopulacao += cruzamento(pais[1], pais[3], taxaCrossover)
        novaPopulacao += cruzamento(pais[2], pais[3], taxaCrossover)
        novaPopulacao += cruzamento(pais[3], pais[2], taxaCrossover)
        novaPopulacao += cruzamento(pais[3], pais[1], taxaCrossover)
        novaPopulacao = [mutacao(individuo, taxaMutacao) for individuo in novaPopulacao]
        populacao = novaPopulacao
        it+=1
    SF = [aptidao(individuo, dados) for individuo in populacao] # Solução final (indivíduos após a 10ª geração)
    return min(SI), min(SF)
```

O código acima mostra a implementação do Algoritmo Genético em seus passos principais (População Inicial, Seleção, Cruzamento e Mutação). Vale notar que, por torneio, foram selecionados 4 indivíduos e realizados 8 cruzamentos diferentes para a geração da nova população. Isso está estritamente relacionado ao tamanho da população.

Os três parâmetros passados para essa função: “tamanhoPopulacao”, “taxaCrossover” e “taxaMutacao” foram escolhidos conforme resultados experimentais encontrados na literatura (Hassanat et al., 2019) e, assim, determinados como:

- tamanhoPopulacao = 16
- taxaCrossover = 0.7
- taxaMutacao = 0.05

Na literatura, o valor experimental para o tamanho da população inicial encontrado é 15, no entanto, acrescentou-se uma unidade a fim de que houvesse um número par de indivíduos.

Adiante neste relatório, relatar-se-á a implementação de capturas de argumentos via linha de comando. Foram colocados esses três parâmetros supracitados como argumentos a serem passados ao chamar a execução via comando. No entanto, vale ressaltar que a seleção e os cruzamentos foram projetados para manter populações intermediárias de tamanho 16.

3.1.4 Função aptidão

A função que calcula a aptidão de um indivíduo consiste na determinação da alcançabilidade dos vértices do grafo a partir de uma fonte da fake news, seguindo os critérios das designações dos recursos. A varredura do grafo a partir de uma

dada fonte é realizada através de um algoritmo de Busca em Largura (BFS), escolhido arbitrariamente.

```python
# FUNÇÃO QUE CALCULA A ALCANCABILIDADE DE VÉRTICES A PARTIR DE UMA FONTE
# Utiliza BFS para varrer o grafo
def calcularAlcance(fonte, individuo, dados):
    T = dados["T"]
    delta = dados["delta"]
    grafo = dados["grafo"]
    alcancados = []
    fila = deque([(0, fonte)])  # (tempo, vertice)

    quantAlcancados = 1

    while fila:
        tempo_u, u = fila.popleft()
        for v, tuv in grafo[u]:
            tempo_v = tempo_u + tuv + (delta if individuo[u] == 1 else 0) # Calcula o tempo do vizinho

            # O vértice v só é considerado alcancado se seu tempo de alcance não ultrapassar T
            if tempo_v <= T and v not in alcancados:
                quantAlcancados += 1
                alcancados.append(v)
                fila.append((tempo_v, v))

    return quantAlcancados
```

Em uma primeira implementação, essa função calcularAlcance() era chamada para todos os vértices, a fim de considerar todos os vértices como fontes da fake news (um por vez) e, dentre as alcançabilidades retornadas, encontrar a maior, ou seja, o pior caso. No entanto, essa abordagem, apesar de sua precisão, performou pessimamente no que diz respeito ao tempo computacional gasto, de modo que não foi tão vantajoso em relação à solução exata via solver.

Desse modo, adotou-se um ajuste na função aptidao(), a qual passou a selecionar o vértice do grafo que apresenta maior grau - e, consequentemente, maior probabilidade de alcançar mais vértices - como a fonte da fake news. Por se tratar de um algoritmo heurístico, não é garantia que o vértice de maior grau como fonte funcione para gerar o pior caso de alcançabilidade, mas essa abordagem é mais performática em relação a tempo computacional. Além disso, é possível notar na próxima seção deste relatório que, com essa abordagem, os resultados obtidos pelo Algoritmo Genético se aproximam dos valores de solução ótima passados como referência pelo enunciado do trabalho.

```python
# FUNÇÃO DO ALGORITMO GENÉTICO PARA CALCULAR APTIDÃO DO INDIVÍDUO
# Dado um indivíduo (alocação de recursos), essa função pega como fonte o vértice com mais adjacências e
# chama a função calcularAlcance(), retornando o valor de alcancabilidade obtido
# Foi realizada a escolha de vértice de maior grau, pois tem maior probabilidade de alcançar mais vértices,
# porém, por se tratar de uma heurística, essa alcancabilidade máxima não é garantida
def aptidao(individuo, dados):
    verticeFonte, grau = dados["maiorGrau"]
    return calcularAlcance(verticeFonte, individuo, dados)
```

3.2 Protocolos de Execução

Por se tratar de um método estocástico, as soluções encontradas pelo Algoritmo Genético configuram-se como uma média de 5 execuções realizadas por "sementes randômicas" diferentes, conforme consta no código abaixo.

```python
# FUNÇÃO PRINCIPAL DO PROGRAMA, QUE REÚNE A LEITURA DE ARQUIVO, A EXECUÇÃO VIA SOLVER E EXECUÇÃO DO ALGORITMO GENÉTICO
# Essa função recebe alguns parâmetros via linha de comando:
# - nomeArquivoSolucao: o nome do arquivo para gravar as melhores soluções
# - tamanhoPopulacao: tamanho da população inicial a ser gerada para o Algoritmo Genético
# - taxaCrossover: valor decimal que representa a taxa de aplicação do crossover
# - taxaMutacao: valor decimal que representa a taxa de aplicação da mutação
# Essa função realiza a contagem do tempo de execução das funções otimizacaoGurobi() e otimizacaoAG()
# e também gera 5 sementes randômicas para executar o Algoritmo Genético 5 vezes, cada uma numa semente diferente
# Assim, os resultados retornados pelo Algoritmo Genético são médias de 5 execuções (e o tempo contado é a soma dos tempos de cada uma das 5 execuções)
# Além disso, essa função gera saídas na saída padrão e no arquivo passado como parâmetro
def main(nomeArquivoInstancia, num, nomeArquivoSolucao, tamanhoPopulacao, taxaCrossover, taxaMutacao):
    with open(nomeArquivoSolucao, "a") as f:
        f.write(f"INSTANCIA {num}.\n")

    dados = lerArquivoDat(nomeArquivoInstancia)
    inicioSolver = time.time()
    solver = otimizacaoGurobi(dados)
    fimSolver = time.time()
    if solver != -1:
        print(f"Solução via solver: {solver} em {fimSolver - inicioSolver} segundos")
        with open(nomeArquivoSolucao, "a") as f:
            f.write(f"Solução via solver: {solver} em {fimSolver - inicioSolver} segundos\n")
    else:
        print("Não há solução ótima via solver")
```

```python
  seeds = [random.randint(0, 999999) for _ in range(5)]
  solucoesI = []
  solucoesF = []

  inicioAG = time.time()
  for i in range(5):
      random.seed(seeds[i])
      SI, SF = otimizacaoAG(dados, tamanhoPopulacao, taxaCrossover, taxaMutacao)
      solucoesI.append(SI)
      solucoesF.append(SF)
  fimAG = time.time()
  mediaSI = sum(solucoesI)/5
  mediaSF = sum(solucoesF)/5
  print(f"Solução inicial via AG: {mediaSI}\nSolução final via AG: {mediaSF} em {fimAG-inicioAG} segundos (5 execuções)")
  with open(nomeArquivoSolucao, "a") as f:
      f.write(f"Solução inicial via AG: {mediaSI}\nSolução final via AG: {mediaSF} em {fimAG-inicioAG} segundos (5 execuções)\n\n")
```

Essa é a função principal do código, que realiza tanto a execução do método exato com o solver Gurobi, quanto da metaheurística.

É possível notar a marcação do tempo de execução de cada uma das funções (otimizacaoGurobi() e otimizacaoAG()), a fim de registrar o tempo computacional gasto. Além disso, utilizou a gravação dos resultados de saída em um arquivo texto cujo nome é passado via parâmetro/argumento.

Durante toda a fase de desenvolvimento da implementação, o algoritmo foi desenvolvido através de um notebook Jupyter, sendo executado através da própria interface da plataforma. No entanto, conforme as regras do enunciado do trabalho, determinou-se a implementação de ajustes, de modo que tanto os parâmetros do algoritmo quanto o nome do arquivo para registro das saídas/resultados fossem passados via linha de comando. Assim, no repositório há apenas um arquivo .py contendo todo o código, herdado do notebook Jupyter, para fins de execução via linha de comando com passagem de parâmetros.

```python
# FUNÇÃO QUE IMPLEMENTA UM PARSER PARA CAPTURAR PARÂMETROS VIA LINHA DE COMANDO
# Essa função configura 4 argumentos a serem passados ao executar o código via linha de comando
def capturarParametros():
    parser = argparse.ArgumentParser()

    # Definindo os parâmetros
    parser.add_argument('--arq', type=str, help="Nome do arquivo para registrar melhor solução", required=True)
    parser.add_argument('--popInicial', type=int, help="Tamanho da população inicial", required=True)
    parser.add_argument('--taxaCross', type=float, help="Valor decimal para a taxa de crossover", required=True)
    parser.add_argument('--taxaMut', type=float, help="Valor decimal para a taxa de mutacao", required=True)

    return parser.parse_args()

# BLOCO DE CÓDIGO RAIZ, EXECUTADO AO EXECUTAR O PROGRAMA
# Primeiramente ele captura os argumentos via linha de comando e chama a função main() para cada instância do problema
if __name__=="__main__":
    args = capturarParametros()
    for num in range(1,11):
       main(f"fake_news_problem/instances/fn{num}.dat", num, args.arq, args.popInicial, args.taxaCross, args.taxaMut)
```

4 Resultados e Análise

Durante as implementações, criaram-se dois arquivos de instância que constam no repositório (“teste.dat” e “teste2.dat”), que se tratam de instâncias menores para serem utilizadas nos testes das implementações.

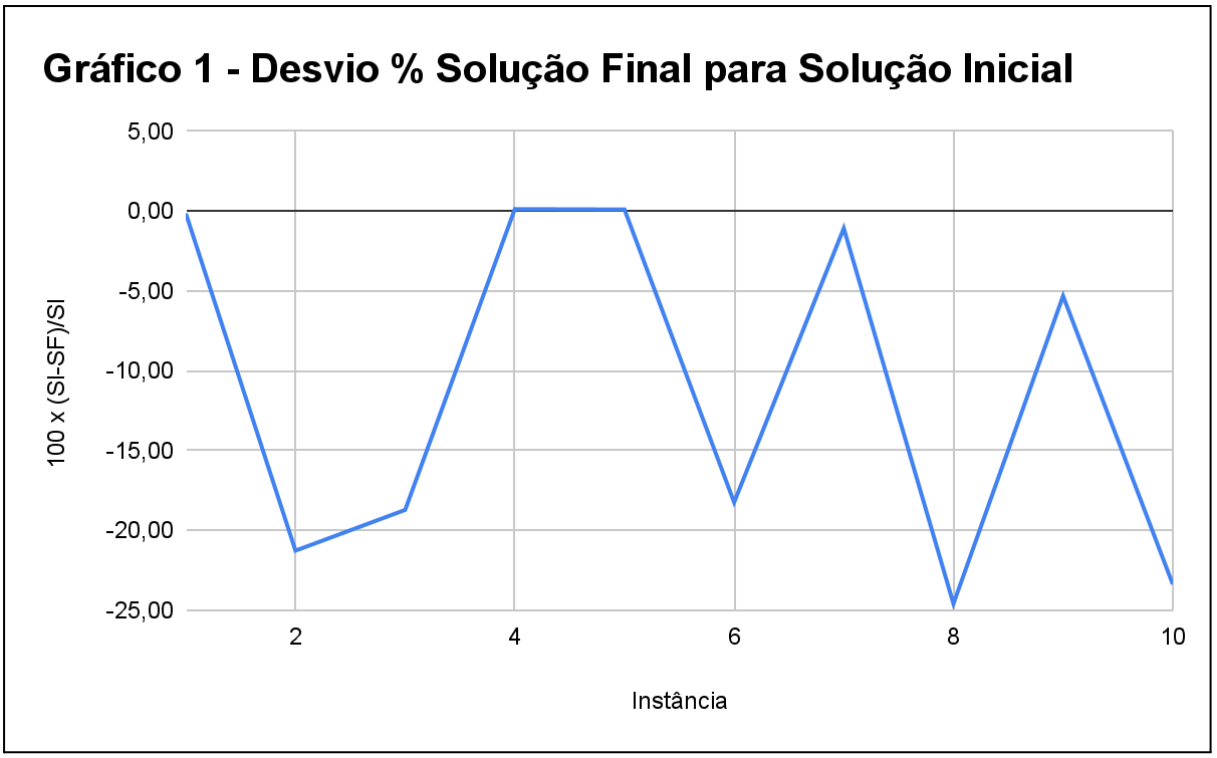
Quando o programa chegou num nível considerado aceitável pelo desenvolvedor, executou-se o código para cada uma das instâncias, a fim de obter os resultados para análises.

Os valores de solução encontrados, tanto do solver quanto da heurística, bem como os registros dos seus respectivos tempos computacionais e os cálculos dos desvios percentuais das soluções para o valor ótimo constam na Tabela 1. Vale ressaltar que, conforme relatado no início da seção anterior, realiza-se a comparação do valor de solução “ótima” tanto com a encontrada pelo solver quanto com o valor referência passado no enunciado do problema.

Tabela 1 - Resultados Computacionais

Instância	Solução Solver	Valor Referência	Tempo Solver	Solução Inicial (AG)	Solução Final (AG)	Tempo AG (Total 5 execuções)	Desvio % SF para SI	Desvio % SF para ótima (solver)	Desvio % SF para ótima (referência)
1	294	189	103,2846594	138,80	139,00	0,7638339996	-0,14	52,72	26,46
2	282	190	68,91210389	215,40	261,20	1,657651663	-21,26	7,38	-37,47
3	294	207	97,73470473	226,60	269,00	2,626433849	-18,71	8,50	-29,95
4	317	216	133,8073485	156,20	156,00	1,07813549	0,13	50,79	27,78
5	312	226	121,0452895	188,20	188,00	1,203125477	0,11	39,74	16,81
6	294	196	141,4032853	133,80	158,20	1,171869278	-18,24	46,19	19,29
7	282	196	145,2816062	204,60	206,80	1,999993324	-1,08	26,67	-5,51
8	294	213	123,6950731	194,40	242,20	2,031248093	-24,59	17,62	-13,71
9	317	226	141,7491698	173,20	182,40	1,142900705	-5,31	42,46	19,29
10	312	235	137,0226758	149,80	184,80	1,406193972	-23,36	40,77	21,36

A partir dos dados dispostos na Tabela 1, podem-se montar alguns gráficos para melhor compreensão visual dos dados para análise.



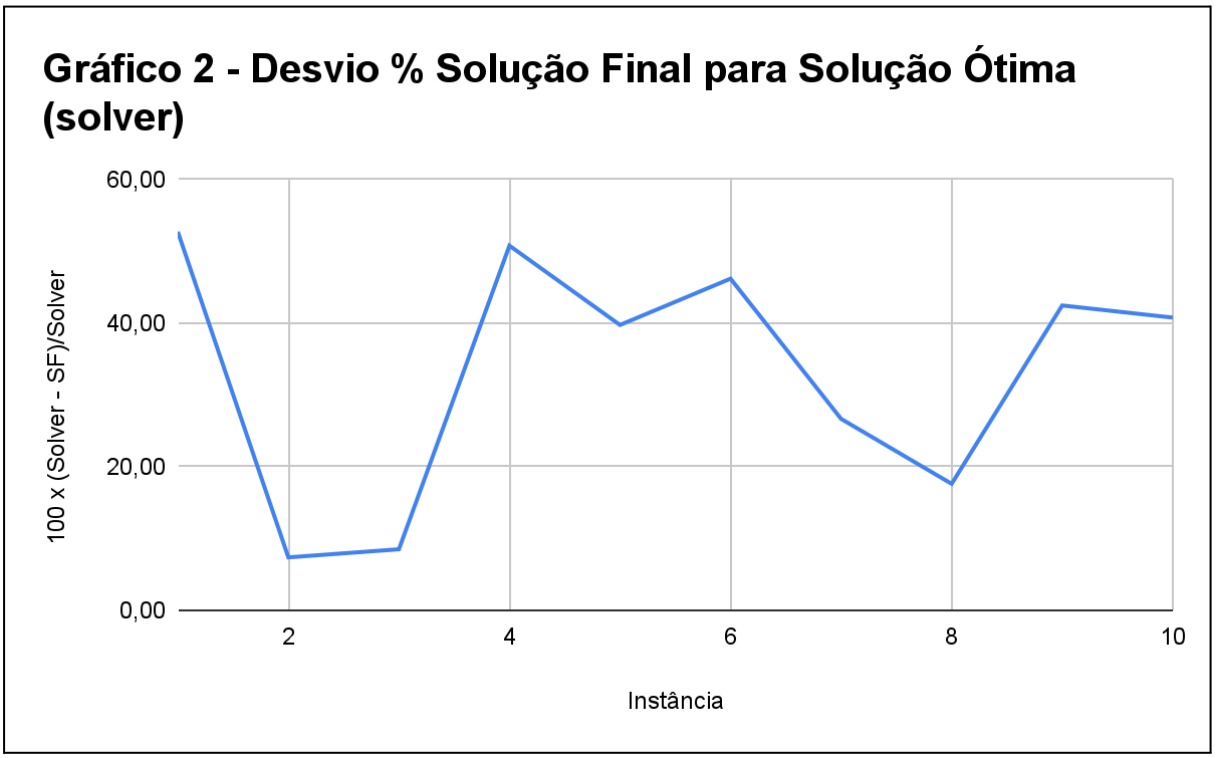


Gráfico 3 - Desvio % Solução Final para Solução ótima (Referência)

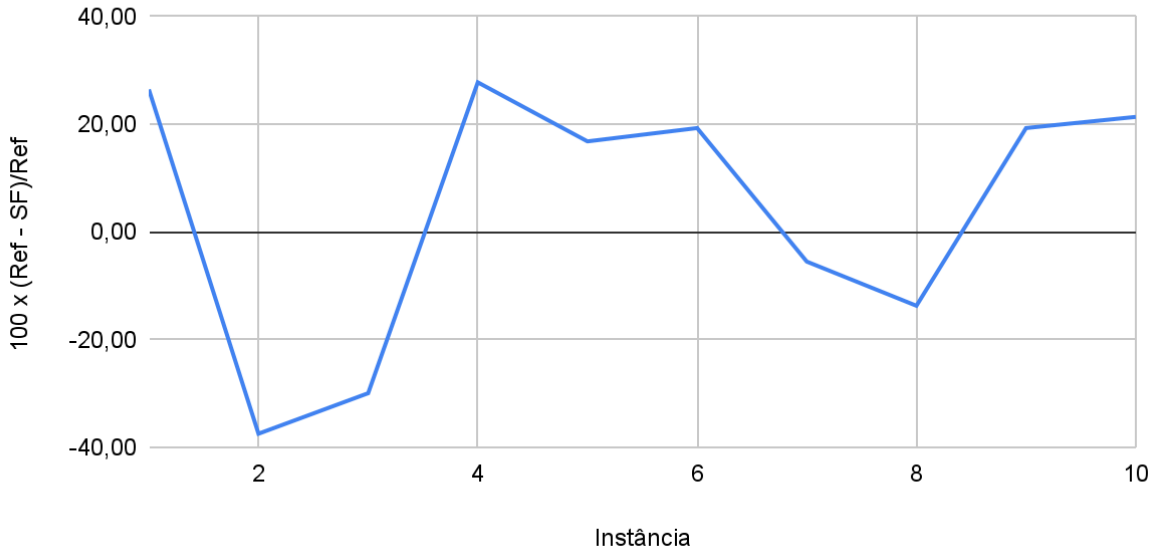
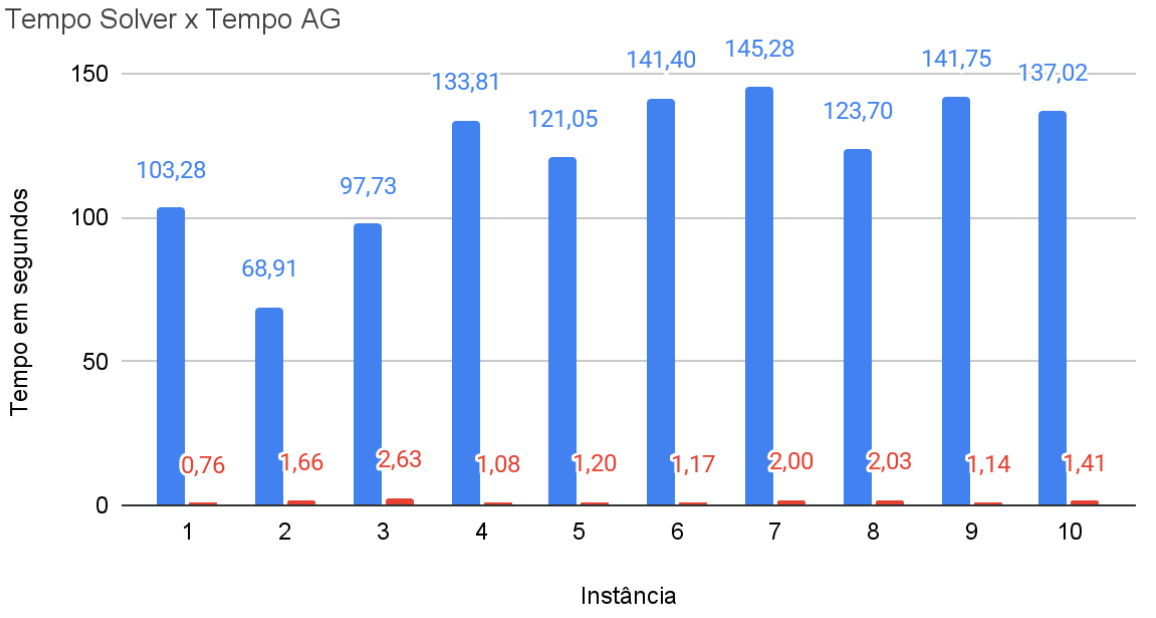


Gráfico 4 - Comparação Tempos Computacionais



É possível notar, no Gráfico 1, que os desvios percentuais das soluções finais do Algoritmo Genético para suas soluções iniciais são todos negativos (exceto das instâncias 4 e 5). Isso traz como significado o fato de que a população inicial (que gerou a solução inicial) foi significativamente alterada na grande parte das instâncias, de modo que essa solução aumentasse de valor, o que é refletido pela solução final, retirada da população final dos indivíduos.

A partir do Gráfico 2, que reflete os desvios percentuais das soluções finais para as soluções ótimas obtidas pelo solver, nota-se desvios positivos altos em relação às soluções ótimas. Isso pode estar relacionado com o que foi alertado no início da seção anterior, a respeito do problema com o formato das instâncias, o que configura como possível causa da disparidade entre os valores encontrados pelo solver e as soluções ótimas passadas como referência.

Plotou-se o Gráfico 3 a fim de comparar as soluções finais do Algoritmo Genético com as soluções ótimas passadas como referência. Apesar das flutuações nos desvios presentes no gráfico, pode-se perceber uma certa proximidade das soluções obtidas pela metaheurística em relação às soluções ótimas conhecidas. Com isso, entende-se que o Algoritmo Genético possibilitou a geração de uma população de indivíduos cujas aptidões estão próximas da ótima.

Por fim, realizando uma análise do Gráfico 4, conclui-se que a implementação da metaheurística gastou tempo computacional significativamente menor que a execução do solver, atingindo, conforme as conclusões retiradas do Gráfico 3, valores próximos do ótimo. Sendo assim, é notória a vantagem performática da metaheurística.

5 Conclusão

Em suma, com o desenvolvimento deste trabalho, foi possível conhecer profundamente o funcionamento do Algoritmo Genético, uma metaheurística evolutiva baseada em população, ao mesmo tempo em que se exercitou o raciocínio matemático para modelagem do Problema da Fake News.

Aprender a implementar uma heurística ofereceu grandes desafios para equilibrar eficiência computacional e eficácia nos resultados. Durante o processo, percebeu-se uma série de dificuldades na adequação do modelo ao problema real, bem como no real entendimento das especificações do problema e da estrutura dos arquivos de instância.

O desenvolvimento foi realizado através de um processo de divisão de tarefas, de modo que os esforços fossem focados e direcionados. De fato, gastou-se um tempo desnecessariamente grande na implementação do modelo e adequação da solução pelo solver, situação que, se fosse solucionada com maior antecedência, deixaria tempo significativo para estudo e implementação eficiente do Algoritmo Genético. Provavelmente, a pressa em relação ao prazo pode ter prejudicado o desenvolvimento correto do Algoritmo Genético.

De forma geral, apesar das complicações durante o desenvolvimento do projeto, foi possível construir um aprendizado robusto sobre metaheurísticas, mais especificamente sobre o Algoritmo Genético e seus pormenores, bem como consolidar lições em relação à modelagem de problemas inteiros de Programação Matemática.

Portanto, este trabalho final constitui-se de um compilado prático de uma série de aprendizados obtidos ao longo da disciplina de GCC118 - Programação

Matemática, o que, certamente, funcionou como motivação para um maior aprofundamento intelectual nesse campo da Ciência da Computação, bem como impulsionou interesses para essa área do mercado de trabalho.

6 Bibliografia

HASSANAT, Ahmad; ALMOHAMMADI, Khalid; ALKAFAWEEN, Esra'a; *et al*. Choosing Mutation and Crossover Ratios for Genetic Algorithms - A Review with a New Dynamic Approach. **Inf.**, v. 10, 2019. Disponível em: <https://consensus.app/papers/choosing-mutation-and-crossover-ratios-for-genetic-hassanat-almohammadi/33920bd1a2c958e5a7e3ca1f062401ca/>. Acesso em: 2 fev. 2025.

SAFE, Martín; CARBALLIDO, Jessica; PONZONI, Ignacio; *et al*. On Stopping Criteria for Genetic Algorithms. *In*: BAZZAN, Ana L. C.; LABIDI, Sofiane (Orgs.). **Advances in Artificial Intelligence – SBIA 2004**. Berlin, Heidelberg: Springer, 2004, p. 405–413. Acesso em: 2 fev. 2025.

KORA, Padmavathi; YADLAPALLI, P. Crossover Operators in Genetic Algorithms: A Review. **International Journal of Computer Applications**, v. 162, p. 34–36, 2017. Disponível em: <https://consensus.app/papers/crossover-operators-in-genetic-algorithms-a-review-kora-yadlapalli/c7a827f1f83b575fbec3eddb5e3fd9c8/>. Acesso em: 2 fev. 2025.

VLAŠIĆ, Ivan; ĐURASEVIĆ, Marko; JAKOBOVIĆ, Domagoj. Improving genetic algorithm performance by population initialisation with dispatching rules. **Computers & Industrial Engineering**, v. 137, p. 106030, 2019. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0360835219304899>. Acesso em: 2 fev. 2025.