Trabalho Final
Otimização Combinatória - INF05010

Eduardo Dalmás Faé[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

edfae@inf.ufrgs.br

1. Descrição do Problema

O problema escolhido foi a coloração mais balanceada. Ele consiste em, dado um Grafo, onde cada vértice possui um peso w, e um valor de k, encontrar uma coloração que siga as restrições para decidir se um grafo é k-colorável [1]. Outrossim minimizando, ao mesmo tempo, o peso máximo assumido por uma cor i.

2. Meta-heurística utilizada

A meta-heurística que será utilizada é a Simulated Annealing [2], que consiste em, partindo de uma solução, realizar modificações a ponto de chegar em uma nova solução, que será, ou não, aceita, levando em consideração seu novo valor e uma temperatura que é resfriada conforme o tempo passa.

3. Formulação matemática do problema

A formulação matemática segue:

Variáveis: $C_{ki} \in \{0, 1\} \quad \forall k \in [K], i \in [n]$

Função Objetiva:

min. $\quad \sum_{i \in [n]} C_{1i} * W_i$

Restrições:

$$
\begin{array}{lll}
C_{ki} + C_{kj} \leq 2 - E_{ij} & \forall k \in [K], i \in [n], j \in [n], & (1) \\
\sum_{k \in [K]} C_{ki} = 1 & \forall i \in [n], & (2) \\
\sum_{i \in [n]} C_{1i} * W_i \geq & & \\
\quad \sum_{i \in [n]} C_{ki} * W_i & \forall k \in [K] & (3) \\
C_{ki} \in \{0, 1\} & \forall k \in [K], i \in [n] & (4)
\end{array}
$$

A restrição (1) cuida para que vértices conectados por uma aresta não compartilhem a mesma cor.

A restrição (2) garante que todo vértice seja preenchido por apenas 1 cor.

A restrição (3) garante que a cor 1 sempre será a cor menos balanceada, dessa forma podemos minimizar a cor 1 e teremos o resultado esperado.

A restrição (1) expande para K*n*n restrições, enquanto a restrição (2) expande para n restrições, já a restrição (3) expande para K restrições. Por fim, a restrição (4) expande para K*n restrições.

4. Parâmetros

Os parâmetros utilizados no Simulated Annealing são:

Temperatura Máxima: É a temperatura inicial utilizada no algoritmo.

Temperatura Mínima: É a temperatura final que será utilizada como critério de terminação, finalizando o algoritmo quando a temperatura é resfriada abaixo desse valor.

Taxa de Resfriamento: É a taxa r $\in$ [0,1] em que a temperatura é resfriada.

Número de Iterações: É o número de iterações que serão realizadas antes de resfriar a temperatura.

5. Algoritmo

Algorithm 1 Simulated Annealing

```
S_cur ← SolucaoInicial()
S_best ← S_cur
temp ← temp_max
while not CriterioDeParada() do
    while not PassaramIteracoes() do
        S_new ← NovaSolucao()
        Δ ← S_new − S_cur
        if Δ ≤ 0 then
            S_cur ← S_new
            if S_new < S_best then
                S_best ← S_new
            end if
        else
            if CriterioDeAceitacao() then
                S_cur ← S_new
            end if
        end if
    end while
    temp ← TemperaturaResfriada()
end while
```

5.1. Escolha da Solução Inicial

A solução inicial é gerada através da Heurística **Recursive largest first algorithm [3]**. Contudo, esse método não funciona para todas as instâncias, visto que algoritmos gulosos para coloração de grafos, como esse adicionam novas cores caso seja necessário, o que vai de encontro com nossa limitação de k.

Para resolver esse problema foi criada uma função que checa se os vizinhos de cada vértice sem cor podem receber uma nova cor, e se essa troca de cores permite que o vértice descolorido receba uma cor própria.

5.2. Critério de Parada

O critério de parada é a temperatura mínima. O algoritmo para quando $temp < temp_{min}$.

5.3. Passagem de Iterações

Antes de resfriar a temperatura se passa I interações.

5.4. Escolha da Nova Solução

A nova solução é escolhida de 2 formas:

Primeiramente, tentamos modificar a cor de um vértice que possui a cor de maior valor. Isso é, escolhemos aleatoriamente um vértice que possui a cor mais desbalanceada e tentamos atribuir a ele uma nova cor. Caso não seja possível, tentamos outro vértice que possui essa cor.

Caso nenhum vértice da cor mais comum possa ser modificado, escolhemos um vértice aleatório e tentamos trocar a cor dele. Caso não seja possível, escolhemos outro vértice aleatório até que um vértice receba uma nova cor.

5.5. Critério de Aceitação

No caso de $S_{new} > S_{cur}$ o valor ainda pode ser aceito, com uma chance de $e^{-\Delta/temp}$.

5.6. Resfriamento da Temperatura

A temperatura é resfriada de acordo de uma taxa de resfriamento $\in [0, 1]$ de forma que:

$temp \leftarrow taxa * temp.$

6. Implementação

6.1. Plataforma de Implementação

O trabalho foi implementado e testado em sistema operacional Windows 10 Home (64-bit), com um processador Intel(R) Core(TM) i5-9400F CPU, com 6 núcleos de 2.9GHz e 16GB de memória. A linguagem de programação utilizada foi C++.

6.2. Estruturas de Dados Utilizadas

As estruturas de dados são como seguem:

Para representar as arestas foi utilizada uma matriz booleana n por n cujo valor é falso se não houver conexão e verdadeiro se houver. Para representar os pesos foi utilizada um array de floats que armazenam o peso de cada vértice. O valor de k é armazenado em um int.

Para representar uma solução temos um float que armazena o valor da solução, um int que armazena a cor mais desbalanceada, um array de inteiros que armazena a cor de cada vértice, e uma matriz de inteiros que armazena quais vértices pertences a uma dada cor.

A estrutura da solução possui ambiguidade de dados, mas isso é intencional a fim de agilizar os cálculos.

7. Teste de Parâmetros

Para os seguintes testes, variou-se cada parâmetro de entrada separadamente, para tentar determinar qual a configuração mais adequada. A configuração padrão utilizada foi:

Temperatura Máxima: 1000

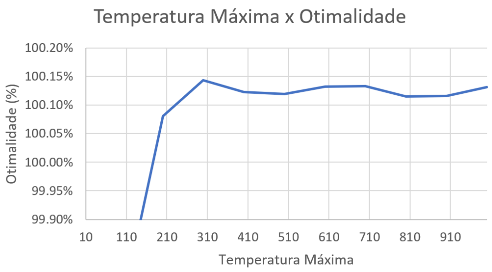
Temperatura Mínima: 10

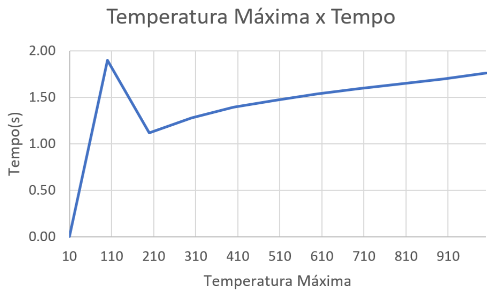
Taxa de Resfriamento: 0.99

N° Iterações: 1000

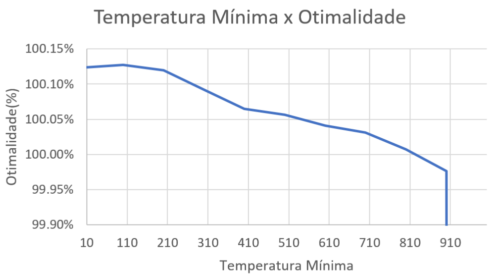
OBS: Para o cálculo de tempo foi ignorado o tempo de geração da solução inical.

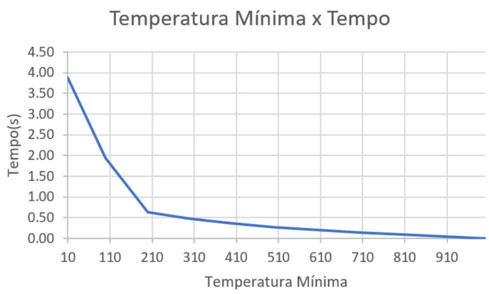
7.1. Temperatura Máxima



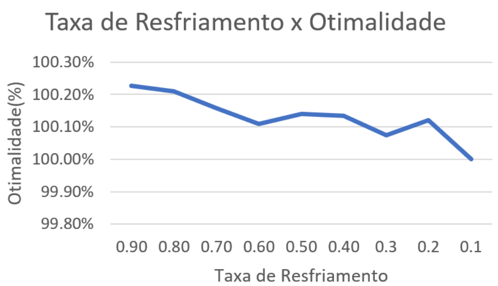


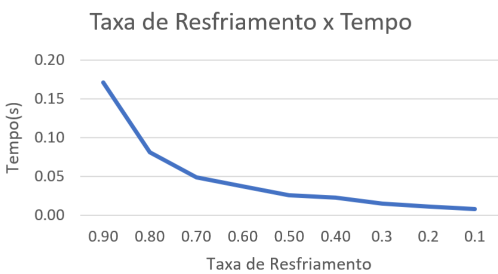
7.2. Temperatura Mínima



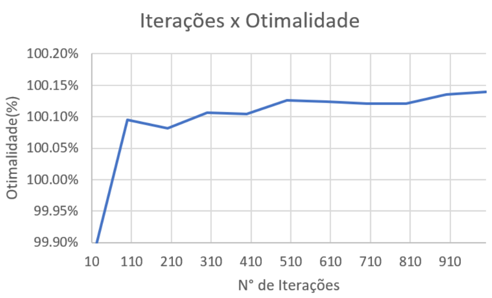


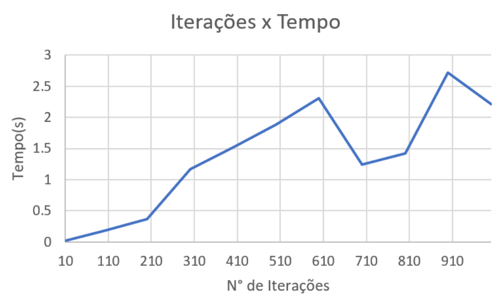
7.3. Taxa de Resfriamento





7.4. Número de Iterações





8. Teste das Instâncias

8.1. Execução com a Heurística

Instância	Valor Ref.	Valor Obtido	Desvio para Ref.	Sol. Inicial	Desvio para Sol. Inicial
cmb01	101405	101306.87	-0.10%	169862.85	-67.67%
cmb02	250083.96	250030.11	-0.02%	582537.7	-132.99%
cmb03	140129.42	139968.67	-0.11%	267787.64	-91.32%
cmb04	78146.84	77756.36	-0.50%	102695.23	-32.07%
cmb05	786315.23	712233.89	-10.40%	1965978.8	-176.03%
cmb06	330082.16	329577.09	-0.15%	559779.49	-69.85%
cmb07	156773.4	156199.81	-0.37%	250497.2	-60.37%
cmb08	636637.88	624612.44	-1.93%	1202907.67	-92.58%
cmb09	221418.46	220528.92	-0.40%	278677.71	-26.97%
cmb10	123671.76	123245.17	-0.35%	170984.68	-38.74%

8.2. Execução com o solver

O solver foi desenvolvido em Julia [4] com os pacotes GLPK e JuMP [5]. Contudo, apesar de estar funcionando, ele é extremamente lento e teve problemas com a maioria das instâncias. Infelizmente não foi possível implementar a adaptação sugerida pelo professor em aula, que possibilitaria que o solver fosse mais eficiente.

Instância	Valor Ref.	Valor Obtido	Desvio para Ref.
test.txt	7	7	0.0%

9. Conclusão

Tendo em vista os pontos supracitados, considera-se o trabalho como satisfatório e acredita-se que os objetivos atingidos foram atingidos com êxito. Contudo, pelo fato de a execução do solver para algumas instâncias não ter sido possível graças ao tamanho das entradas, conclui-se que o trabalho não ficou 100% perfeito como seria o desejado.

Referências

[1] Wikipedia. *Graph coloring*.

[2] Marcus Ritt. *Notas de aula*. 2022.

[3] Frank Thomson Leighton. *A Graph Coloring Algorithm for Large Scheduling Problems*. 1979.

[4] *Julia 1.9 Documentation*.

[5] Jaafar Ballout. *Linear programming in Julia with GLPK and JuMP*.