Approximate Graph Coloring

Jorge Siqueira Serrão, Ramses Messias de Oliveira Carvalho

Universidade Federal de Roraima

2023

# 1. Análise e descrição do artigo Karger, Motwani, Sudan, 1998, Approximate Graph Coloring by 1998, Approximate Graph Coloring by Semidefinite Pro- gramming

No artigo é considerado o problema de colorir k – grafos colorizáveis com o menor número de cores possível, para isso é apresentado um algoritmo de tempo polinomial aleatório que colore um grafo de 3 cores em n vértices com um mínimo de cores {O(^(1/3) log^(1/2) log n),O(n^(1/4) log^(1/2) n)} onde é o grau máximo de qualquer vértice.

No artigo é trabalhado o vetor de relaxamento de coloração cuja solução é, por sua vez, usada para aproximar a solução para o problema da coloração. Em vez de atribuir cores aos vértices de um gráfico, é atribuido vetores unitários (n-dimensionais) aos vértices. Dado um grafo G=(V, E) em n vértices, e um número real k >= 1, um vetor k-colorizável de G é uma atribuição de valores unitários vi do espaço R^n para cada vértice i ∈ V, de tal modo que para quaisquer dois vértices adjacentes e j, o produto escalar de seus vetores satisfaz a desigualdade.

(vi,vj) <= -1/(k-1)

Resolvendo o problema de coloração vetorial: Para resolver o problema é preciso seguir a seguinte definição. Dado um grafo G = (V, E) com n vértices, uma matriz k-colorizável do grafo é uma matriz n x n simétrica semidefinida positiva M, com mii = 1 e mij <= -1/(k-1) se{i,j} ∈ *E.* Considerando um grafo que tenho um vetor ou matriz k-colorizável. Seguinifica que há solução para o programa semidefinido com α = -1/(k-1).

Semicoloração: Um k-semicolorizável de um grafo G é uma atribuição de k cores para pelo menos metade de seus vértices de forma que não haja dois vértices adjacentes com a mesma cor.

Se um algoritmo A pode ki-semicolorir qualquer subgrafo i-vértice do grafo G em tempo randomizado polinomial, onde ki aumenta com i, então A pode ser usado para O(kn log n)-cor G. Além disso, se existir e > 0 tal que para todo i, ki = Ω(i^e), então A pode ser usado para colorir G com O(kn) cores.

Arredondamento por partições de hiperplano: Considera-se um hiperplano H. É dito no artigo que para separar dois vetores se eles não estão no mesmo lado do hiperplano. Para qualquer beira {i, j} ∈ E, nós dizemos que o hiperplano H corta a beira se ele separa o vetor vi e vj. Usando o algoritmo de Wigderson o algoritmo pode ser melhorado passando de O(n^0,613) cores para O(n^0,387) cores.

Teoria da dualidade por definição dado um grafo G = (V, E) em n vértices, um vetor estrito de coloração k de G é uma atribuição de vetores unitários ui do spaço R^n para cada vértice i ∈ V, de modo que para quaisquer dois vértices adjacentes i e j o produto escalar de seus vetores satisfaz a igualdade (ui, uj) = -1/(k1). Como dito no capitulo 8 do artigo um grafo é estritamente vetorial para colorir k se ele tiver uma estrita coloração vetorial para k.

A lacuna entre cores vetoriais e números cromáticos: No capitulo 9 é debatido sobre o fato de o algoritmo em análise não está em sua forma ótima onde é apresentado o teorema de Milner cuja a definição é: Seja S1,… Sα uma anticadeia de conjuntos de um universo de tamanho m tal que, para todos os i e j, |Si ∩ Sj | ≥ t. Então, deve ser o caso de a <= (m(m+t+1)/2).

O segundo teorema do capitulo 9 estabelece que os grafos têm uma grande lacuna entre seu vetor de número cromático e os números cromáticos.

Seja n=(n,r) denotam o número de vértices do grafoK(m,r,t). Para r = m/2 e t=m/8, o grafo K(m,r,t) é vetor de 3 cores, mas tem um número cromático de pelo menos n^0,0113.

X >= (1,007864)^lg n = n^lg1,007864 ≈ n^0,0113

O terceiro teorema fala que existe um grafo kneser K(m,r,t) que é um vetor de 3 cores mas tem um número cromático excedendo n^0,016101, onde n = (m,n) denota o número de vértices no grafo. Além disso para grandes k, existe um grafo de Kneser K(m, r, t) que ser colorido com o vetor k, mas tem número cromático excedendo n^0,0717845. Usando o teorema de Milner é possível provar que o expoente do número cromático é pelo menos.

1- - (m – t)log 2m/(m – t) + (m + t)log 2m/(m +t) 2((m – r)log m/(m – r) + r log m/r)

Isso mostra que exite um conjunto de valores com vetor número cromático 3 e número cromático pelo de menos n^0,016101. Para grandes números cromáticos de vetor constante, o valor limite do expoente do número cromático é aproximadamente 0,0717845.

# 2.Implementação do algoritmos de coloração de grafos

Para colorir os grafo foram utilizados os algoritmos backtracking e guloso.

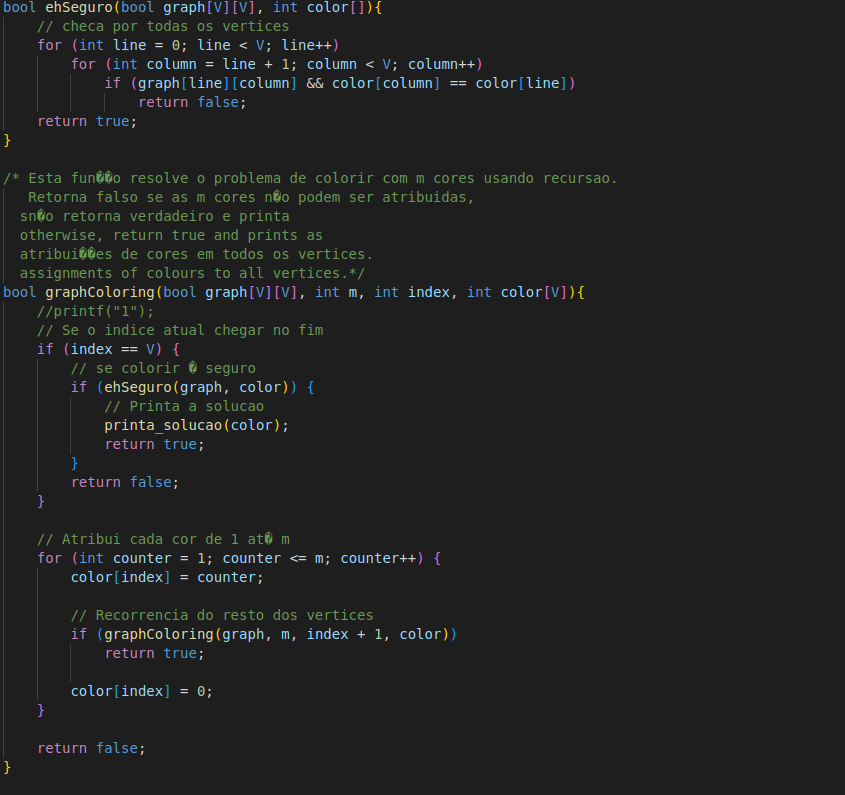


Imagem 1: parte do algoritmo de backtrakcing criação do grafo.

(Somatorio(Σ) de i=1 até v) \* (somatorio(Σ) de j=i+1 até v) + T(i) = { 1 i=v | T(i+1) +1 i < v } (somatorio(∑ ) de i=1 até v) \* (v-i)

(v^2) – (v^2)/2 + v/2 (v^2)/2 + v/2

T(i) = T(i+1) +1

T(i) = [T(i+2) +1] +1 T(i) = T(i+2) +2

T(i) = [T(i+3)+1] +2 T(i) = T(i+3) +3

T(i) = T(i+k) + k

Assume: i+k = v logo k = v – i T(i) = T(v) + v - i

T(i) = 1 + v - i

(v^2)/2 + v/2 + 1 + v - i = (v^2)/2 + 3v/2 - i + 1 O(v^2) ou O(n^2)

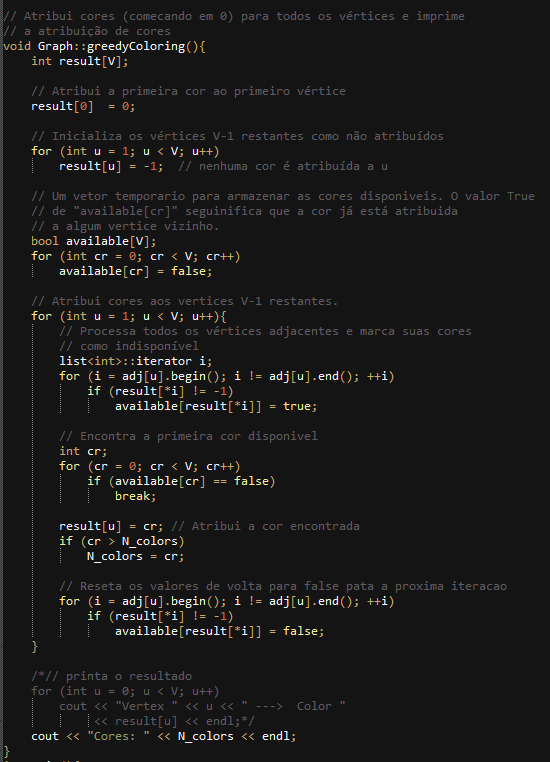


Imagem 3: Função do algoritmo guloso para colorir os vértices do grafo.

(Somatorio(∑) de cr=1 até v)\*((Somatorio(∑) de i=adj[u].begin() até adj[u].end()) + (Somatorio(∑) de cr=1 até v) + (Somatorio(∑) de i=adj[u].begin() até adj[u].end()))

(Somatorio(∑) de cr=1 até v)\*(adj[u].end() - adj[u].begin() +1) + (Somatorio(∑) de cr=1 até v)\*v + (Somatorio(∑) de cr=1 até v)\*(adj[u].end() - adj[u].begin() +1)

vadj[u].end() - vadj[u].begin() + v + v^2 + vadj[u].end() - vadj[u].begin() + v v^2 + 2vadj[u].end() - 2vadj[u].begin() + 2v

O(v^2) ou O(n^2)

# 3. Análise dos resultados

Para realizar o codificação dos algoritmos foi utilizada a IDE code block com compilador GCC no windows 10, para gerenciamento dos artefatos foi utilizado o Trello. Foram feito três caso de testes para cada entrada de arestas:

Melhor caso: quando nenhum vértice é conectado a alguma aresta, assim só haverá uma cor.

Pior caso: quando cada vértice do grafo é conectado com cada um dos outros vértice, assim haverá uma cor diferente para cada vértice(Grafo Completo).

Caso médio: foi gerado um grafo que conecta os vértices aleatoriamente, e com a metade da quantidade de arestas de cada do pior caso.

A análise dos algoritmos consistiu de testes de 4 à 10000 vértices, onde em cada teste foram feitas três análises com quantidade de cores diferentes para que fosse possível adquirir os valores referentes ao melhor, pior e caso médio de cada uma das implementações analisadas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Teste N° | Vértice s | Arestas (pior/ médio) | Algoritmo Guloso: Tempo - N° Cores (melhor/ pior/ médio) | Algoritmo BackTracking: Tempo - N°  Cores  (melhor/ pior/ médio) |
| 1 | 4 | 16/8 | 0.071 s - 1 cor/ 0.078 s - 4 cores/  0.085 s - 3 cores | 0.079 s - 1 cor/ 0.076 s - 4 cores/ 0.075 s - 2 cores |
| 2 | 10 | 100/50 | 0.082 s-1 cor/0.072 s - 10 cores/ 0.078 s - 5 cores | 0.078 s - 1 cor/2.913 s - 10 cores/  2.592 s - 5 cores |
| 3 | 12 | 144/72 | 0.071 s-1 cor/0.081 s - 12 cores/ 0.067 s - 5 cores | 0.071 s - 1 cor/ timelimitout/ timelimitout |
| 4 | 100 | 10^4/5x10  ^3 | 0.063 s-1cor/0.074s - 100 cores/  0.080 s - 27 cores | 0.077 s - 1 cor/ timelimitout/ timelimitout |
| 5 | 10000 | 10^8/10^7 | 0.081 s-1 cor/ alloc error/ 18.677s - 294 cores | execution error/ execution error/ execution error |

Imagem 4: Tabela com resultado dos casos de uso de cada algoritmo.

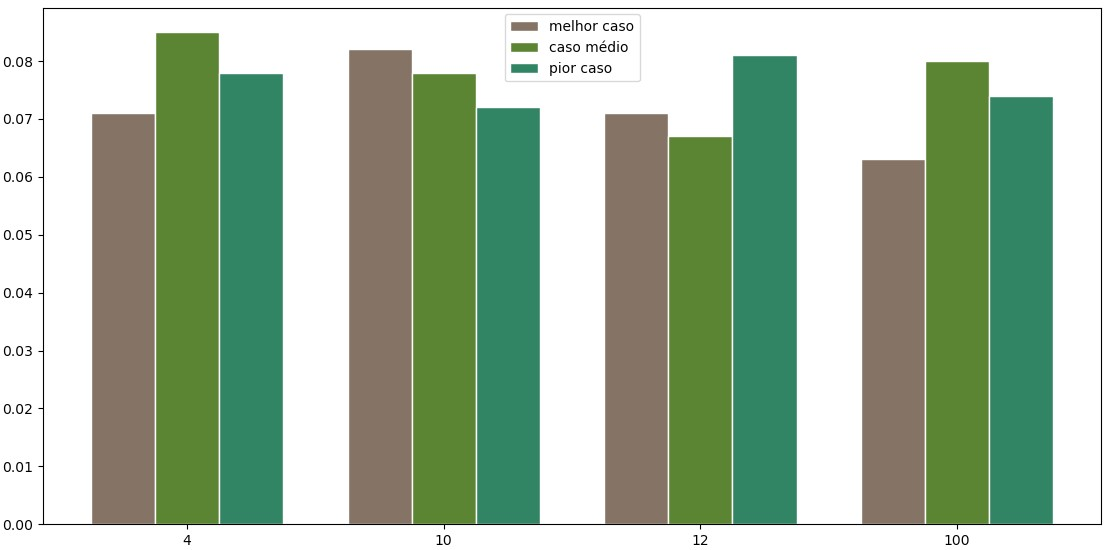


Imagem 5: Gráfico de desempenho do algoritmo guloso quantidade/tempo.

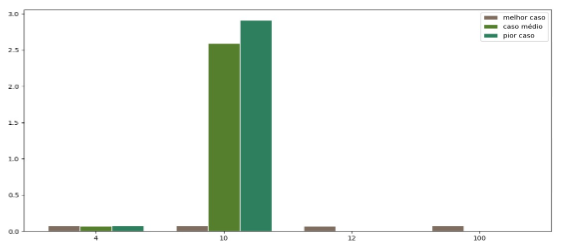


Imagem 6: Gráfico de desempenho do algoritmo backtracking quantidade/tempo.

Foi observado que o BackTracking, aumenta muito o seu tempo de execução, conforme maior o número de vértices do grafo, não sendo possível identificar o tempo no seu tempo limite determinado para um grande número de arestas, esse comportamento pode ser devido ao fato de o mesmo utilizar recursividade, o que aumenta o número de recursos proporcionalmente aos números nas entradas, e não funciona para mil dezenas de entrada.

O algoritmo Guloso possui um tempo de execução estável desde pequenas, entradas, até às demais grandes entradas. No caso do teste 5, ele gera alloc error também para o caso de 5x10^7 arestas, por isso foi adaptado para um caso que ainda funciona(10^7), para mostrar que é capaz de gerar ainda um grande número de cores.

# 4. Referências

Karger, Motwani, Sudan, 1998,

Approximate Graph Coloring by Semidefinite Pro-gramming

https://[www.geeksforgeeks.org/test-case-generation-set-4-random-directed-undirected-weighted-](http://www.geeksforgeeks.org/test-case-generation-set-4-random-directed-undirected-weighted-) and-unweighted-graphs/