

UNIVERSIDADE FEDERAL DE RORAIMA

Approximate Graph Coloring

Alunos: Luis Henrique e Pedro Aleph

Boa Vista
2020

Análise e descrição do artigo Karger, Motwani, Sudan, 1998.
Approximate Graph Coloring by Semidefinite Programming

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 $\{O(\Delta^{1/3} \log^{1/2} \Delta \log n), O(n^{1/4} \log^{1/2} n)\}$ onde Δ 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 atribuído vetores unitários (n -dimensionais) aos vértices. Dado um grafo $G=(V, E)$ em n vértices, e um número real $k \geq 1$, um vetor k -colorizável de G é uma atribuição de valores unitários v_i do espaço R^n para cada vértice $i \in V$, de tal modo que para quaisquer dois vértices adjacentes i e j , o produto escalar de seus vetores satisfaz a desigualdade.

$$(v_i, v_j) \leq -1/(k-1)$$

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 $n \times n$ simétrica semidefinida positiva M , com $m_{ii} = 1$ e $m_{ij} \leq -1/(k-1)$ se $\{i, j\} \in E$. Considerando um grafo que tenho um vetor ou matriz k -colorizável. Segue que há solução para o programa semidefinido com $\alpha = -1/(k-1)$.

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 k_i -semicolorir qualquer subgrafo i -vértice do grafo G em tempo randomizado polinomial, onde k_i aumenta com i , então A pode ser usado para $O(kn \log n)$ -cor G . Além disso, se existir $\epsilon > 0$ tal que para todo i , $k_i = \Omega(i^\epsilon)$, 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 $\{i, j\} \in E$, nós dizemos que o hiperplano H corta a beira se ele separa o vetor v_i e v_j . 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 u_i do espaço R^n para cada vértice $i \in V$, de modo que para quaisquer dois vértices adjacentes i e j o produto escalar de seus vetores satisfaz a igualdade $(u_i, u_j) = -1/(k-1)$. Como dito no capítulo 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 capítulo 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 S_1, \dots, S_α uma antecadeia de conjuntos de um universo de tamanho m tal que, para todos os i e j , $|S_i \cap S_j| \geq t$. Então, deve ser o caso de

$$\alpha \leq (m + t + 1)/2.$$

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

Seja $n = (m - r) \log \frac{2m}{m - r} + (m + r) \log \frac{2m}{m + r}$

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

$$X \geq (1,007864)^{\lg n} = n^{\lg 1,007864} \approx n^{0,0113}$$

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 $n = (m - r) \log \frac{2m}{m - r} + (m + r) \log \frac{2m}{m + r}$

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.

$$\frac{1 - (m - r) \log \frac{2m}{m - r} + (m + r) \log \frac{2m}{m + r}}{2((m - r) \log m / (m - r) + r \log m / r)}$$

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

Implementação dos códigos

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

Função do algoritmo backtracking que checa se é seguro colorir um vertice.

```
// checa se o grafo colorido é seguro ou não
bool ehSeguro(bool graph[V][V], int color[]){
    // checa por todas os vertices
    for (int i = 0; i < V; i++)
        for (int j = i + 1; j < V; j++)
            if (graph[i][j] && color[j] == color[i])
                return false;
    return true;
}
```

Função do algoritmo backtracking para colorir os vértices do grafo.

```
/* Esta função resolve o problema de colorir com m cores usando recursão.
   Retorna falso se as m cores não podem ser atribuídas,
   não retorna verdadeiro e printa */
bool graphColoring(bool graph[V][V], int m, int i, int color[V]){
    // Se o índice atual chegar no fim
    if (i == V) {
        // se colorir é seguro
        if (ehSeguro(graph, color)) {
            // Printa a solução
            printa_solucao(color);
            return true;
        }
        return false;
    }

    // Atribui cada cor de 1 até m
    for (int j = 1; j <= m; j++) {
        color[i] = j;

        // Recorrência do resto dos vértices
        if (graphColoring(graph, m, i + 1, color))
            return true;

        color[i] = 0;
    }

    return false;
}
```

$(\text{Somatorio}(\Sigma) \text{ de } i=1 \text{ até } v) * (\text{somatorio}(\Sigma) \text{ de } j=i+1 \text{ até } v) + T(i) = \{ 1 \mid i=v \mid T(i+1) + 1 \mid i < v \}$
 $(\text{somatorio}(\Sigma) \text{ de } i=1 \text{ até } v) * (v-i)$
 $(v^2)/2 - (v^2)/2 + v/2$
 $(v^2)/2 + v/2$

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

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 $O(n^2)$

Função do algoritmo guloso para colorir os vértices do grafo.

```
void Graph::greedyColoring() {
    int result[V];
    result[0] = 0;

    for (int u = 1; u < V; u++)
        result[u] = -1;

    bool available[V];
    for (int cr = 0; cr < V; cr++)
        available[cr] = false;

    for (int u = 1; u < V; u++) {
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = true;

        int cr;
        for (cr = 0; cr < V; cr++)
            if (available[cr] == false)
                break;

        result[u] = cr;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = false;
    }
}
```

$(\text{Somatorio}(\Sigma) \text{ de } cr=1 \text{ até } v) * ((\text{Somatorio}(\Sigma) \text{ de } i=\text{adj}[u].\text{begin}() \text{ até } \text{adj}[u].\text{end}()) + (\text{Somatorio}(\Sigma) \text{ de } cr=1 \text{ até } v) + (\text{Somatorio}(\Sigma) \text{ de } i=\text{adj}[u].\text{begin}() \text{ até } \text{adj}[u].\text{end}()))$

$(\text{Somatorio}(\Sigma) \text{ de } cr=1 \text{ até } v) * (\text{adj}[u].\text{end}() - \text{adj}[u].\text{begin}() + 1) + (\text{Somatorio}(\Sigma) \text{ de } cr=1 \text{ até } v) * v + (\text{Somatorio}(\Sigma) \text{ de } cr=1 \text{ até } v) * (\text{adj}[u].\text{end}() - \text{adj}[u].\text{begin}() + 1)$

$v \cdot \text{adj}[u].\text{end}() - v \cdot \text{adj}[u].\text{begin}() + v + v^2 + v \cdot \text{adj}[u].\text{end}() - v \cdot \text{adj}[u].\text{begin}() + v$

$v^2 + 2v \cdot \text{adj}[u].\text{end}() - 2v \cdot \text{adj}[u].\text{begin}() + 2v$

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

Benchmark

Para realizar foi utilizada a IDE code block com compilador GCC no windows 10, 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.

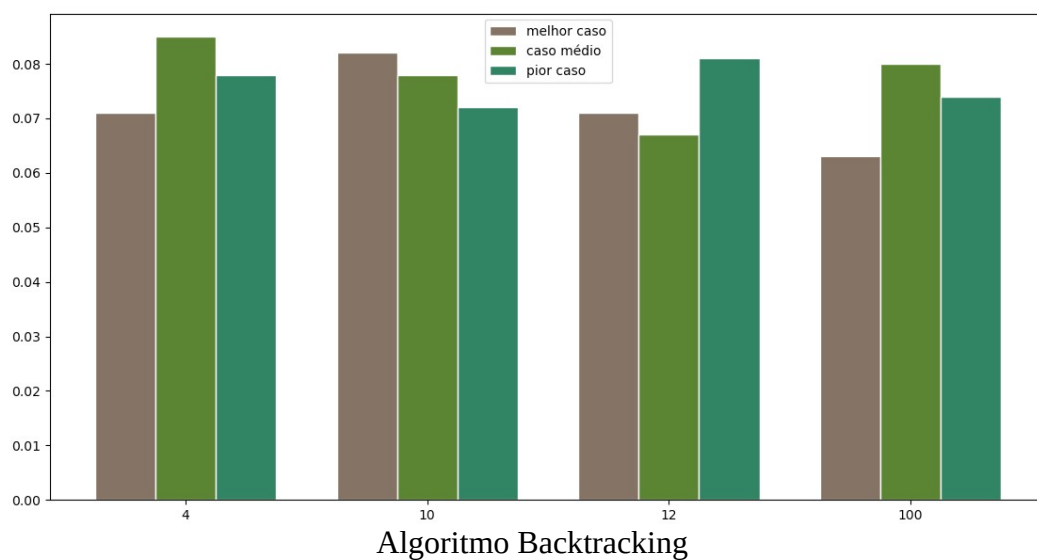
Pior caso: quando cada vértice do grafo é conectado com cada um dos outros vértice, assim haverá uma cor diferente para cada vértice.

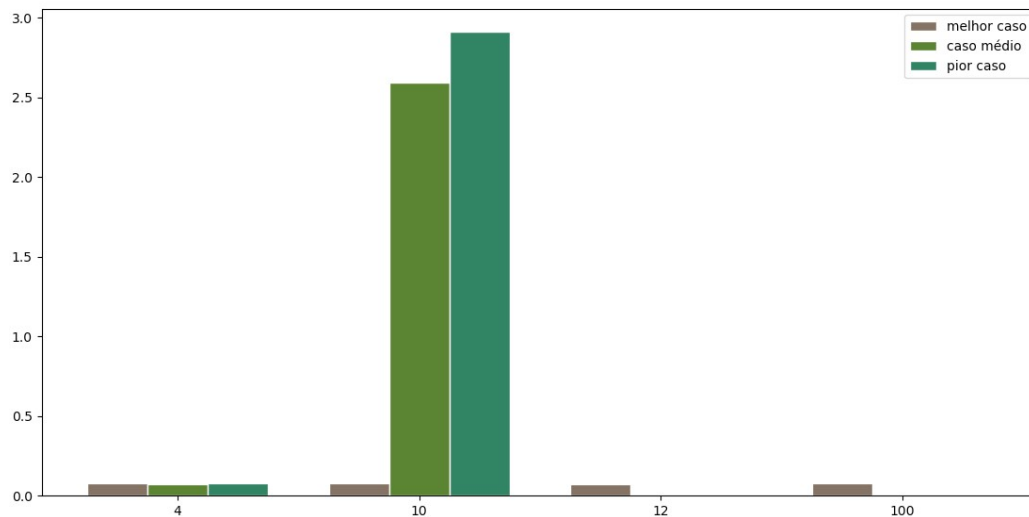
Caso médio: foi gerado um grafo que conecta os vértices aleatoriamente, e com a metade da quantidade de arestas de cada do pior caso.

Análise dos resultados

Teste N°	Vértices	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	$10^4/5 \times 10^3$	0.063 s - 1 cor/ 0.074 s - 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

Algoritmo Guloso





Observamos que o BackTracking, aumenta drasticamente 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 grande número de arestas, tal comportamento pode ser devido a sua natureza recursivo, que aumenta o número de recursos conforme maiores os números nas entradas, e não funciona para mil dezenas de entrada.

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 5×10^7 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.

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-and-unweighted-graphs/>