3 TP - AEDs III

Coloracao de grafos

Lucas Marchisotti de Souza
Marlon Silveira Basilio

November 12, 2018

1 Introdução

Na matemática, um grafo pode ser pensado como um conjunto de objetos em que alguns pares de objetos estão conectados por links. Os objetos interconectados por links são geralmente chamados de pares de vértices conectados por arestas. Grafos podem ser usados para modelar um número surpreendentemente grande de áreas problemáticas, incluindo redes sociais, química, agendamento, entrega de encomendas, navegação por satélite, engenharia elétrica, e redes de computadores. Incluso nesta gamma de problemas está o problema da coloração de grafo.

O Problema da coloração mínima de vértices está em encontrar uma coloração válida de um grafo não-dirigido com o menor número de cores possível. Ainda não foi descoberto nenhum algoritmo rápido que resolva o problema da coloração para qualquer grafo não-direcionado. A coloração de grafos é chamada completa se todos os vertices v ∈ V são marcados com uma cor c(v) ∈ 1, . . . ,k, ou entao a coloração é considerada parcial. Um conflito é descrito como uma situação em que um par de vértices adjacentes u,v ∈ V são marcados com a mesma cor (isto é, u, v ∈ E e c(v) = c(u)). Se a coloração não tem conflitos então ela é considerada apropriada, senão é considerada imprópria. A coloração é viável se é ao mesmo tempo completa e apropriada. O número cromático de um grafo é o número mínimo de cores requeridas para colorir um grafo viável, e por sua vez, um grafo viável que utiliza exatamente o número cromático de cores é considerado ótimo (R.M.R. Lewis, 2016).

Grafos Completos

Um grafo completo com n vertices, denotado por Kn, são grafos que apresentam uma aresta entre cada par de vértices, dando um conjunto E de m = n (n − 1) 2 arestas. Isso é obvio porque todos os vértices no grafo completo são mutuamente adjacentes, todos os vértices devem a atribuição de sua própria cor individual. Daí o número cromático de um grafo completo (Kn) = n (R.M.R. Lewis, 2016).

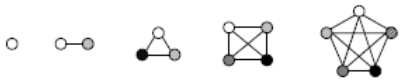

Figure 1: Grafo Completo

Grafos Bipartidos

Grafos bipartidos, são grafos cujos vértices podem ser particionados em dois conjuntos V1 e V2, de modo que as bordas só existam entre vértices em V1 e vértices em V2. Como resultado, V1 e V2 são ambos conjuntos independentes, o que significa que os grafos bipartidos podem ser colocados usando coletores justos, com vértices em V1 sendo atribuídos a uma cor, e todos os vértices em V2 sendo atribuídos a outra. É óbvio, que um grafo G apresenta um número cromático (G) = 2 se e somente se for bipartido (R.M.R. Lewis, 2016).

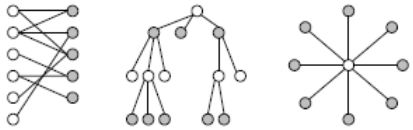

Figure 2: Grafo Bipartido

Grafos Cíclicos ou Circulares e Planares

Grafos circulares, denotados por Cn, onde n≥3, compreende um conjunto de vértices V = v1, ..., vn e um conjunto de arestas E da forma v1, v2, v2, v3, ..., vn − 1, vn, vn, v1 Os grafos ciculares mostrados na figura 3 (a). Sabe-se que apenas duas cores são necessárias para colorir Cn quando n é par (um ciclo par); Portanto, mesmo os ciclos são um tipo de gráfico bipartido. No entanto, três cores são necessárias quando n é ímpar (um ciclo ímpar)(R.M.R. Lewis, 2016). Isso é ilustrado na figura 3.

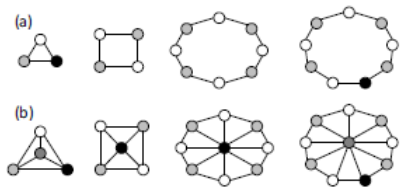

Figure 3: Grafo Circular

Grafos de Rede

Grafos de rede podem ser formados colocando todos os vértices em uma formação de treliça em um plano bidimensional. Em um gráfico de rede esparso, cada vértice é adjacente a quatro vértices: o vértice acima dele, o vértice abaixo dele, o vértice à direita e o vértice à esquerda (veja a figura 3 (a)). Para um grafo de rede denso, um padrão similar é usado, mas os vértices também são adjacentes aos vértices em suas diagonais circundantes (Figura 3 (b))(R.M.R. Lewis, 2016).

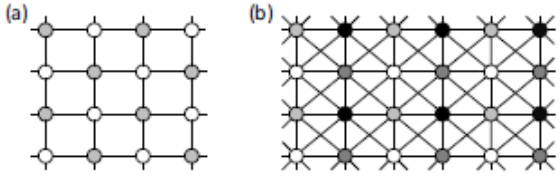

Figure 4: Grafo em malha

O problema da coloração de grafos também é conhecido como um problema de rotulagem de grafos em que os rótulos são chamados de cores. Ele busca o menor número de cores necessárias para colorir o grafo de forma tal que não haja dois vértices adjacentes de mesma cor. É um problema do tipo np-difícil, a figura 5 e 6 abaixo traz exemplos de coloração de grafo.

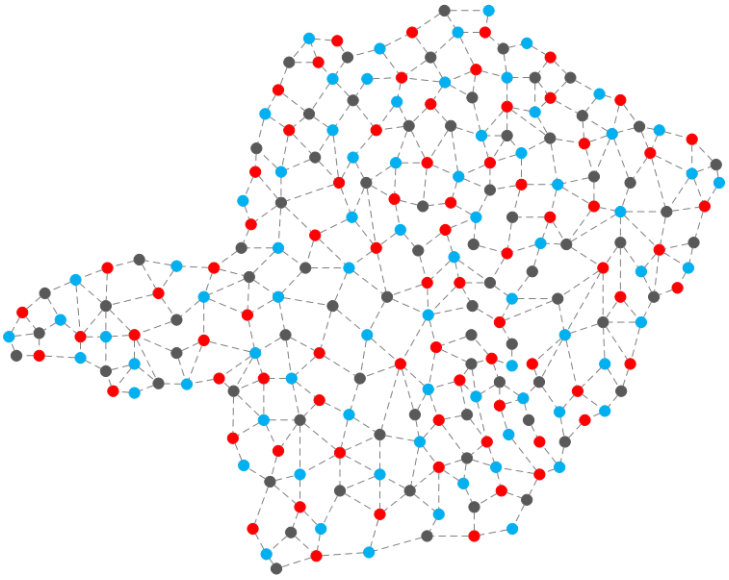

Figure 5: Exemplo de coloração de grafo

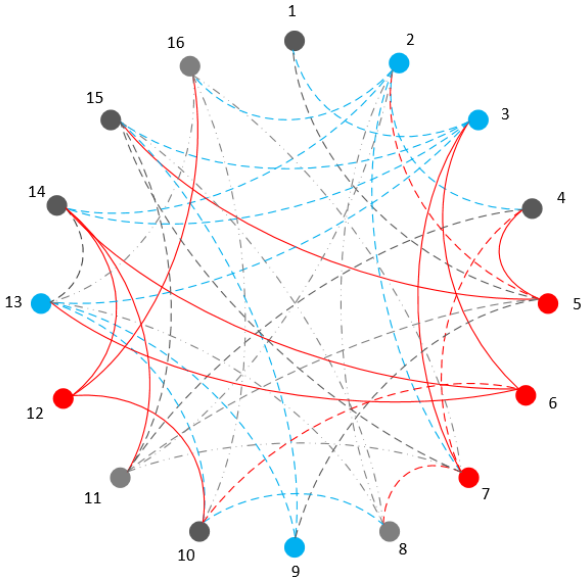

Figure 6: Exemplo de coloração de grafo

2 Desenvolvimento

A modelagem da coloração de grafos se deu utilizando um grafo G=(V,A), com representação em lista de adjacência. Nele os vértices são os nodos e as arestas as ligações entre os nodos. No passo seguinte, a construção foi dividida em 3 partes, a implementação de um algoritmo que fosse capaz de encontrar uma solução ótima para o problema e neste caso foi utilizado um algoritmo guloso com aplicação de backtracking e a implementação de duas heurísticas que pudessem resolver o problema eficientemente e que produzissem "boas" soluções sob o ponto de vista prático, a primeira sendo a heurística de coloração sequêncial e a segunda de coloração heuristica.

2.1 Algoritmo guloso com backtracking

2.1.1 Solução Proposta

Para encontrar soluções ótimas seria necessário implementrar um algoritmo guloso que verificasse todas as possibilidades, e para isso foi utilizado o que é conhecido como algoritmos de retrocesso. Backtracking é uma metodologia geral que pode ser usada para determinar uma solução ótima (ou possivelmente todas as soluções ótimas) para um problema computacional, a coloração de grafos é um deles.

Em essência, os algoritmos de backtracking funcionam construindo sistematicamente soluções parciais em soluções completas. No entanto, durante esse processo de construção assim que ele obtém evidência de que não há como completar a solução parcial atual para obter uma solução ótima, o algoritmo retrocede para tentar encontrar maneiras adequadas de ajustar a solução parcial atual.

Um algoritmo de backtracking simples para coloração de gráficos pode funcionar da seguinte maneira. Dado um grafo G = (V, E), os vértices são ordenados primeiro de alguma maneira, de modo que o vértice vi (1≤i) corresponde ao iésimo vértice nesta ordenação. Também precisamos escolher um valor k que indique o número de cores disponíveis. Inicialmente, isso pode ser simplesmente k = ∞. O algoritmo de backtracking executa uma série de etapas de avanço e retrocesso. Os passos de avanço colorem os vértices na ordem dada até que seja identificado um vértice que não possa ser colorido de forma viável com uma das k cores disponíveis.

Por outro lado, os passos para trás retrocedem pelos vértices coloridos na ordem inversa e identificam pontos em que cores diferentes, como as indicações para os vértices, podem ser feitas. As etapas de encaminhamento são então retomadas a partir desses pontos. Se uma coloração viável completa for encontrada, então k pode ser ajustado para o número de cores usadas nesta coloração menos 1, com o algoritmo então continuando. Por fim, o algoritmo termina quando um passo para trás alcança o vértice raiz v1, ou quando algum outro critério de parada, como um limite máximo de tempo, é atendido (R.M.R. Lewis, 2016).

Algoritmo de força bruta guloso que encontrará todas as possíveis soluções do grafo e retornará o menor valor cromático possível. A coloração com Backtracking utiliza a mesma estratégia da Heurística

Sequencial, onde o algoritmo vai caminhando pelo vetor e definindo a coloração mínima para cada vértice. Porém esse comportamento é replicado para todos os caminhos possíveis. Foi feito uma pequena alteração na execução do Backtracking, pois foram implementadas algumas podas. A variável global "cormax" ela é definida anteriormente como um valor alto como 9999, porém é atualizada em cada tentativa de encontrar o valor mínimo cromático. Portanto quando for encontrado um grafo em que a maior cor de seus vértices for menor que o "cormax", logo "cormax" receberá o valor desta cor. Desta forma é possível podar muitos resultados ruins e encontrar resultados ótimos sem ter que vasculhar todas as soluções possíveis.

2.1.2 Análise de complexidade

A solução por força bruta para uma k-coloração deve considerar cada uma das (k n) alocações de k cores para n vértices e verificar a viabilidade da solução. Tal abordagem é impraticável, exceto para grafos pequenos.

2.2 Heurística de coloração Sequêncial

Tem o funcionamento aproximado de um algoritmo guloso, que é um dos algoritmos heurísticos mais simples mas fundamentais para a coloração de grafos. O algoritmo opera tomando vértices um por um de acordo com alguma ordem (possivelmente arbitrária) e atribui a cada vértice a primeira cor disponível. Por causa disso é um algoritmo heurístico, as soluções que ele produz podem muito bem ser sub-ótimas, no entanto, pode também pode ser mostrado que algoritmos gulosos podem produzir soluções ideais para qualquer grafo dado a sequência correta de vértices.

Linearizamos o grafo transformando-o em um vetor de vértices e utilizamos uma lista de adjacência para determinar a vizinhança de cada vértice. Percorrendo cada vértice do grafo, usamos a função "Colorindo". A estratégia utilizada nessa função para encontrar a menor cor possível de um vértice é feita utilizando um laço em função de um contador e a incrementação da variável cor. Este contador será incrementado sempre que um vértice adjacente tiver a mesma cor que o vértice que queremos colorir. Sempre que este contador for diferente de zero, o laço será reiniciado. Quando o contador for igual a zero, significa que não temos um vértice adjacente que tem a mesma cor do vértice que queremos colorir, logo iremos sair do laço e então determinaremos a cor do vértice como a variável cor.

2.2.1 Análise de complexidade

Vamos agora estimar a complexidade computacional do algoritmo guloso com relação ao número de verificações de restrição que são executadas. Vemos que um vértice é colorido em cada iteração, significando n = c iterações do algoritmo são necessárias no total. Na iteração c ($1 \leq i \leq n$), estamos preocupados em encontrar uma cor viável para o vértice c*i. No pior caso, este vértice entrará em choque com todos os vértices que o precederam em c, significando que (i − 1) verificações de restrição serão executadas antes que uma cor adequada seja determinada. De fato, se o gráfico que estamos colorindo é o gráfico completo Kn, o pior caso ocorrerá para todos os vértices; daí um total de 0 + 1 + 2 +. . + (n − 1) verificações de restrição serão realizadas. Isto dá ao algoritmo guloso uma complexidade geral do pior caso O (n^2).

2.3 Algoritmo de coloração heurística

Este trabalho consiste em desenvolver e avaliar soluções ótimas e heurística para o problema da coloração de grafos. Uma coloração de k cores de um grafo G = (V, A), não direcionado é uma função c : V → 1, 2, · · · , k, tal que c(u) 6 = c(v) para toda aresta (u, v) ∈ A. Em outras palavras, os números 1, 2, · · · , k representam k cores e vértices adjacentes têm que ter cores diferentes. O problema da coloração de grafos consiste em determinar o número mínimo de cores necessário para colorir um dado grafo.

Linearizamos o grafo transformando-o em um vetor de vértices e utilizamos uma lista de adjacência para determinar a vizinhança de cada vértice. Obtendo o vetor de vértices, iremos ordená-lo de forma crescente, em função do grau de adjacência de cada vértice, utilizando o algoritmo de ordenação SHELL-SORT. Chamaremos a função COR-MAX. Dentro dessa função, temos um laço que fará uma leitura desse vetor de forma decrescente, para colorirmos os vértices em função dos maiores graus para os menores. Em seguida, será chamada a função "Colorindo", que irá encontrar a menor cor possível para o vértice referenciado. Após encontrar a cor do vértice, ele é definido como "visitado" e irá colorir com a sua cor todos os seus vértices adjacentes, de menos os que também já foram visitados e assim sucessivamente até visitar o ultimo vértice do grafo (quando terminar o laço).

2.3.1 Analise de complexidade

A ordenação dos vértices em ordem não crescente de graus pode ser realizada em O(n log n) com um algoritmo ótimo genérico. O cálculo do grau dos vértice é O(m + n), usando listas de adjacências.

A coloração propriamente depende da obtenção de até k cores, inserida em um laço O(n), implicando em custo O(k.n). O custo global do algoritmo é O(n2)

3 Análise dos resultados

O algoritimo guloso com backtraking apresentou os melhores resultados em todos os testes executados, ao mesmo tempo, a heurística de coloração sequêncial quando não apresentou resultados iguais ao backtraking ficou muito proximo dele. Ja a coloração heurística apresentou os piores resultados comparados com os outros dois algoritimos, tanto em tempo de execução maior quando no cálculo do mínimo de cores necessárias para obtenção do resultado como podemos observar na tabela 7, que traz alguns resultados nos testes executados.

Tempo	N de Vertices	N de Arestas	Coloração Sequêncial	Coloracao Heuristica	Backtraking	Tempo Total
N de Cores	5	20	3	3	3	0,00059 seg
Tempo de Leitura				0,00170 seg		
Tempo de Processamento			0,00002 seg	0,00002 seg	0,00008 seg	
Tempo de Sistema			0,00000 seg	0,00000 seg	0,00000 seg	
N de Cores	10	86	5	6	5	0,00080 seg
Tempo de Leitura				0,00151 seg		
Tempo de Processamento			0,00003 seg	0,00000 seg	0,00000 seg	
Tempo de Sistema			0,00000 seg	0,00003 seg	0,00003 seg	
N de Cores	15	115	7	7	6	0,00118 seg
Tempo de Leitura				0,00214 seg		
Tempo de Processamento			0,00003 seg	0,00003 seg	0,00020 seg	
Tempo de Sistema			0,00000 seg	0,00000 seg	0,00000 seg	
N de Cores	20	212	8	9	8	0,00240 seg
Tempo de Leitura				0,00282 seg		
Tempo de Processamento			0,00004 seg	0,00005 seg	0,00068 seg	
Tempo de Sistema			0,00000 seg	0,00000 seg	0,00000 seg	
N de Cores	25	387	13	11	11	0,00704 seg
Tempo de Leitura				0,00000		
Tempo de Processamento			0,00000	0,00000	0,00404	
Tempo de Sistema			0,00008	0,00008	0,00006	
N de Cores	30	542	12	11	11	0,00659 seg
Tempo de Leitura				0,00123		
Tempo de Processamento			0,00000	0,00000	0,00259	
Tempo de Sistema			0,00006	0,00010	0,00011	

Figure 7: Tabela de tempos e quantidade de cores

Já a tabela 8 mostra o cálculo da quantidade mínima de cores necessárias para colorir os variados grafos testados no programa.

N de Vertices	N de Arestas	Coloração Sequêncial	Coloracao Heuristica	Backtraking
5	3	3	3	3
5	9	3	4	3
5	20	3	3	3
5	23	3	5	3
5	23	4	4	4
10	34	3	3	3
10	26	4	4	4
10	51	5	6	5
10	70	7	7	7
10	86	5	6	5
15	98	4	6	4
15	104	5	6	5
15	115	7	7	6
15	152	7	8	7
15	180	8	8	8
20	132	3	4	3
20	125	4	5	3
20	161	8	8	8
20	212	8	9	8
20	227	8	9	8
25	111	6	7	5
25	297	11	10	9
25	335	10	10	10
25	387	13	11	11
25	590	11	13	11
30	476	5	6	4
30	420	9	13	9
30	505	11	13	11
30	542	12	11	11
30	635	14	15	14

Figure 8: Tabela de Quantidade de cores

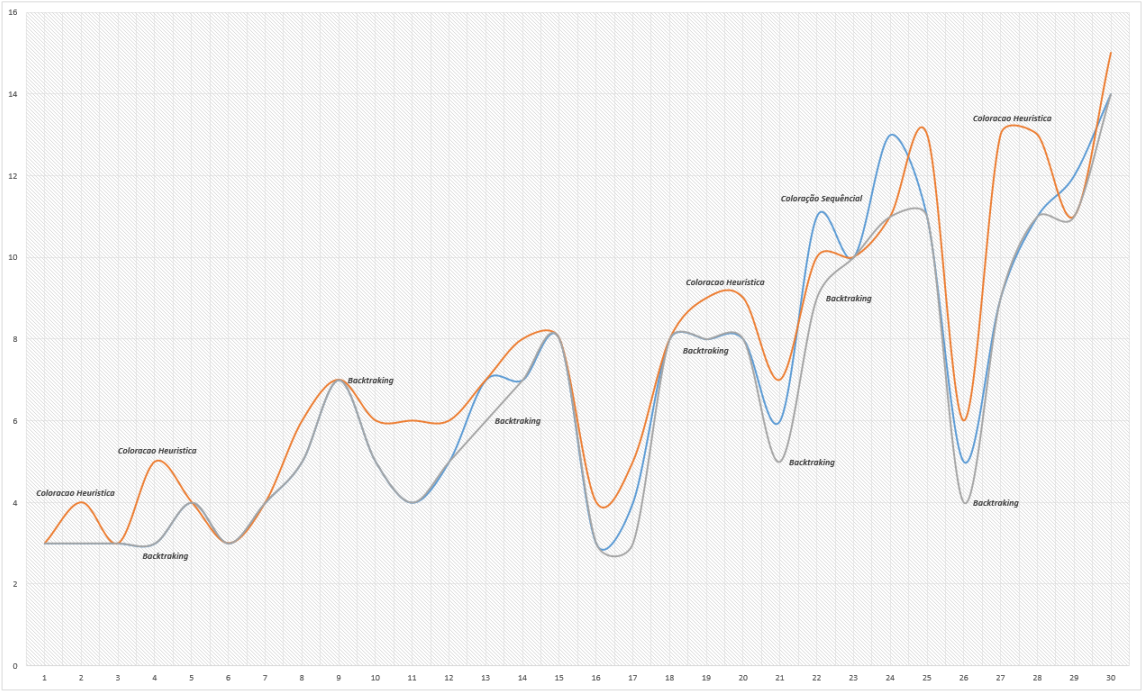

Figure 9: Grafico de cores em testes com numero crescente de arestas

4 Conclusão

O Algoritmo de Coloração com Backtracking trará sempre resultados ótimos, porém ele consome muito mais recursos computacionais e em consequência gasta mais tempo.

O Algoritmo de Coloração Sequencial muitas vezes trará resultados bons e poderá trazer até resultados ótimos, tudo depende da ordem em que os vértices são percorridos. É um algoritmo barato computacionalmente e por esta razão, pode ser utilizado em conjunto com outros algoritmos de coloração para cobrir um grafo com o menor número possível de cores.

O Algorítimo de Coloração Heurística comparada com o coloração sequêncial apresentou tempo acima na maioria dos resultados gerando piores resultados. Percebemos que os algoritimos gulosos são mais "demorados" e consomem mais recursos computacionais perém geram melhores resultados sendo que na maioria das vezes são ótimos

References

[1] Cormen T.H., Leiserson C.E., Rivest R., Stein C. 2012. *Algoritmos: Teoria e Prática*. Ed. Mit Press. Estados Unidos.

[2] Ziviani, N. *Projeto de Algoritmos Com Implementações em Pascal e C*: http://www.dcc.ufmg.br/algoritmos, Pioneira Thomson Learning, Segunda Edição, 2004.

[3] Lewis, R.M.R. *A Guide to Graph Colouring*, 2016. Ed. Springer International Publishing. Switzerland