Coloração Aproximada de Grafos: Análise e Implementação dos Algoritmos *First Fit* e *DSatur*

George Lucas Monção Zambonin e Wandressa da Silva Reis

Resumo* — *Este artigo apresenta os detalhes da implementação de dois algoritmos de coloração aproximada de grafos. Uma análise comparativa dos algoritmos é realizada. A implementação busca avaliar a eficiência computacional de cada algoritmo em diferentes tipos de grafos, oferecendo uma visão de suas capacidades.

Abstract* — *This paper presents the implementation details of two approximate graph coloring algorithms. A comparative analysis of the algorithms is conducted, along with a practical application of graph coloring, which plays a crucial role in areas such as scheduling, frequency allocation, and map coloring. The implementation allows for the evaluation of the computational efficiency, scalability, and adaptability of each algorithm across different types of graphs, providing a clear understanding of their capabilities and limitations.

I. INTRODUÇÃO

A coloração de grafos é um problema clássico na teoria dos grafos, que consiste em atribuir cores aos vértices de um grafo de tal forma que dois vértices adjacentes não recebam a mesma cor. Este problema possui diversas aplicações práticas, como na alocação de frequências em redes sem fio, escalonamento de tarefas, design de circuitos eletrônicos, entre outros. Devido à sua complexidade computacional, especialmente em grafos grandes e densos, encontrar soluções exatas para problemas de coloração de grafos pode ser inviável em termos de tempo de execução. Por isso, surgem métodos aproximados que oferecem soluções eficientes, embora não necessariamente ótimas, com tempos de execução reduzidos.

Este relatório documenta a implementação dos algoritmos de coloração aproximada de grafos *First Fit* e *DSatur*. Ambos os algoritmos têm como objetivo minimizar o número de cores usadas, ao mesmo tempo em que garantem que a regra de coloração seja respeitada.

II. O Problema da Coloração Aproximada de Grafos

O problema da coloração aproximada de grafos consiste em atribuir cores a um grafo de maneira que nenhum par de vértices adjacentes compartilhe a mesma cor, minimizando o número total de cores utilizadas. Em termos formais, dado um grafo $G = (V, E)$, onde V representa o conjunto de vértices e E o conjunto de arestas, o objetivo é encontrar uma função de coloração $c: V \rightarrow C$, onde C é o conjunto de cores, de modo que para cada aresta $(u, v) \in E$, $c(u) \neq c(v)$.

A coloração de grafos é um problema NP-difícil, o que significa que não existe um algoritmo eficiente conhecido que resolva todos os casos em tempo polinomial. Assim, a coloração aproximada visa encontrar soluções que são "suficientemente boas" em um tempo razoável, mesmo que não sejam ótimas. Os algoritmos aproximados buscam garantir que o número de cores utilizadas esteja dentro de um fator constante do número mínimo de cores necessário, conhecido como número cromático.

A. Aplicação Prática da Coloração de Grafos

Uma das aplicações mais conhecidas da coloração de grafos é a coloração de mapas, que pode ser exemplificada pelo teorema das quatro cores. Este teorema afirma que, dado um mapa de países (ou regiões), é sempre possível colorir os países de tal forma que nenhum país adjacente tenha a mesma cor, utilizando no máximo quatro cores, como é exemplificado na figura 1.

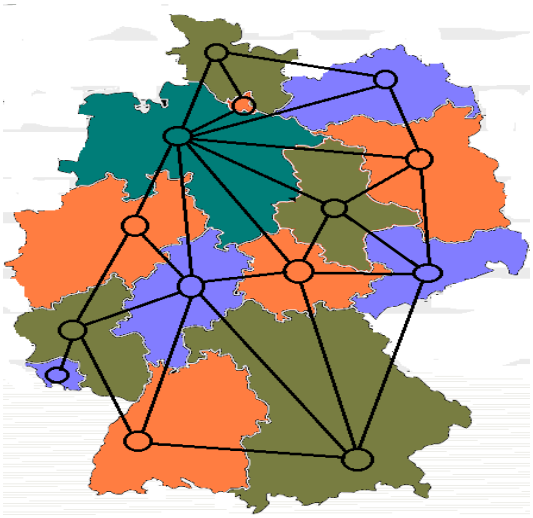

Fig. 1. Exemplo de coloração de mapas utilizando o conceito de coloração de grafos

A importância dessa aplicação reside em sua utilidade em situações reais, como:

1. **Planejamento Territorial**: Ao dividir uma região em várias áreas administrativas ou políticas, a coloração de mapas ajuda a visualizar e organizar as divisões, assegurando que vizinhos não tenham a mesma designação.

2. **Atribuição de Frequências**: Em redes de comunicação, onde diferentes torres de transmissão podem interferir entre si, a coloração de grafos é utilizada para designar diferentes frequências de operação, evitando a sobreposição de sinais.

3. **Escalonamento de Tarefas**: Em contextos em que várias tarefas precisam ser realizadas em diferentes regiões, a coloração pode ser aplicada para assegurar que tarefas que utilizam os mesmos recursos não sejam agendadas simultaneamente.

Essas aplicações demonstram como o problema da coloração de grafos se estende além do âmbito teórico, proporcionando soluções práticas para desafios do mundo real.

III. Coloração de grafos é um NP-completo

A colorização de grafos é um problema clássico em ciência da computação e combinatória, que envolve atribuir cores aos vértices de um grafo de modo que nenhum par de vértices adjacentes compartilhe a mesma cor. O objetivo é minimizar o número de cores utilizadas, o que é conhecido como o número cromático do grafo. Esse problema, chamado de Problema de Colorização de Grafos, pertence à classe dos problemas NP-completos, um subconjunto dos problemas NP.

1. Problemas NP e NP-completo

Antes de demonstrar por que a coloração de grafos é um problema NP, é essencial entender o conceito de NP (tempo polinomial não determinístico). NP refere-se a uma classe de problemas de decisão onde, dada uma solução candidata, é possível verificar sua correção em tempo polinomial. Em outras palavras, se uma solução é fornecida (como uma colorização válida de um grafo), pode-se verificar rapidamente se ela satisfaz as condições requeridas.

Um problema é NP-completo quando duas condições são atendidas:

- Ele está em NP, o que significa que sua solução pode ser verificada em tempo polinomial.
- Qualquer problema em NP pode ser transformado em um problema NP-completo em tempo polinomial (essa propriedade é conhecida como NP-dureza).
- O Problema de Coloração de Grafos é NP-Completo

A coloração de grafos é NP-completa, o que significa que não só pode ser verificada em tempo polinomial, mas é tão difícil quanto qualquer outro problema em NP. A versão de decisão do Problema de Coloração de Grafos pergunta se um grafo pode ser colorido usando no máximo k cores, onde k é um inteiro dado. Esse problema de decisão é conhecido como k-Colorabilidade.

Para mostrar que a Coloração de Grafos é NP-completa, podemos delinear os dois passos necessários:

- Pertinência a NP: Dada uma colorização válida do grafo, verificar se nenhum par de vértices adjacentes têm a mesma cor e se o número de cores utilizadas não excede k pode ser feito em tempo polinomial. Portanto, a Coloração de Grafos pertence a NP.
- NP-dureza: A NP-dureza da Coloração de Grafos pode ser provada através de uma redução de outro problema NP-completo conhecido. A redução mais comum é a partir do problema 3-SAT, um problema NP-completo conhecido. Essa redução mostra que qualquer instância de 3-SAT pode ser transformada em uma instância do Problema de Coloração de Grafos em tempo polinomial, demonstrando assim que a Coloração de Grafos é pelo menos tão difícil quanto 3-SAT.

Devido a esses fatores, a colorização de grafos é classificada como NP-completa, o que significa que, embora as soluções possam ser verificadas rapidamente, encontrar a solução em si (ou seja, o número cromático mínimo) é computacionalmente intratável para grafos grandes.

3. Implicações da NP-Completude

A NP-completude do Problema de Colorização de Grafos tem implicações significativas tanto para aplicações teóricas quanto práticas. Na prática, resolver a colorização de grafos de forma ótima para grafos grandes é computacionalmente dispendioso, e à medida que o tamanho do grafo aumenta, o problema se torna exponencialmente mais difícil. Algoritmos de aproximação e métodos heurísticos são comumente empregados para encontrar soluções quase ótimas dentro de limites de tempo razoáveis, especialmente em aplicações do mundo real, como agendamento, alocação de registradores em compiladores e gerenciamento de recursos de rede.

IV. First Fit

O algoritmo First-Fit é um heurístico amplamente utilizado para resolver o problema de coloração de grafos, onde o objetivo é atribuir cores aos vértices de forma que dois vértices adjacentes não compartilhem a mesma cor. O algoritmo processa cada vértice em uma ordem determinada e atribui a menor cor possível que não conflita com seus vizinhos.

A principal vantagem do *First Fit* é sua simplicidade e rapidez, já que ele não requer muita computação para determinar qual cor atribuir a cada vértice. No entanto, ele não garante uma solução ótima (mínimo número de cores), pois a ordem de processamento dos vértices pode influenciar significativamente o resultado.

Esse algoritmo é frequentemente utilizado em cenários onde a eficiência computacional é mais importante que a qualidade da solução, como em sistemas de alocação de recursos e escalonamento de tarefas.

A. Visão Geral do Algoritmo

O algoritmo First-Fit percorre o grafo vértice por vértice. Para cada vértice, o algoritmo examina seus vértices adjacentes para determinar quais cores já estão em uso. Em seguida, atribui a menor cor disponível que não está sendo usada pelos vizinhos. O processo pode ser descrito nos seguintes passos:

1. Inicializar o Vetor de Cores: Um vetor color[] é inicializado com -1 para todos os vértices, indicando que nenhum vértice foi colorido ainda.
2. Inspeção dos Vizinhos: Para cada vértice u, o algoritmo verifica todos os seus vértices adjacentes v para marcar as cores que estão em uso como indisponíveis.
3. Atribuir a Cor: O algoritmo então atribui a menor cor disponível a u, garantindo que nenhum vértice adjacente compartilhe a mesma cor.
4. Resetar Cores Usadas: Após colorir cada vértice, o vetor que rastreia as cores usadas é resetado para o próximo vértice.

B. Análise da Complexidade de Tempo

A complexidade de tempo do algoritmo First-Fit é determinada pelo número de vértices V e arestas E no grafo. As operações principais incluem verificar os vizinhos de cada vértice e atribuir uma cor:

1. Processamento de Vértices: O algoritmo processa cada vértice exatamente uma vez, resultando em tempo O(V) para esta etapa.
2. Verificação dos Vizinhos: Para cada vértice u, o algoritmo verifica todos os seus vizinhos v. Como o número total de verificações de vizinhos em todos os vértices é proporcional ao número de arestas, esta etapa tem uma complexidade de O(E).
3. Atribuição de Cores: Após verificar os vizinhos, atribuir a menor cor disponível requer iterar pelas cores possíveis, o que leva tempo O(1) em implementações práticas (O(Δ), onde Δ é o grau máximo do grafo).
4. Resetar Cores Usadas: O reset do vetor de cores usadas leva O(V) para cada vértice, resultando em $O(V^2)$ no total.

No entanto, na maioria dos cenários reais, especialmente para grafos esparsos, onde $E \ll V^2$, o tempo gasto na verificação dos vizinhos domina a complexidade geral. Como resultado, a complexidade de tempo é:

$$O(V+E)$$

C. Considerações Práticas

Em grafos densos, onde o número de arestas E se aproxima de $O(V^2)$, a complexidade do algoritmo pode chegar a $O(V^2)$. No entanto, em grafos esparsos, que são comuns em aplicações do mundo real, E é tipicamente proporcional a V, resultando em uma complexidade linear. Portanto, o algoritmo First-Fit é altamente eficiente para grafos esparsos, tornando-se uma escolha popular em problemas de escalonamento e alocação de recursos.

V. DSatur

O algoritmo *DSatur* (Saturation Degree) é um algoritmo sofisticado para coloração de grafos, projetado para minimizar o número de cores utilizadas. A principal característica do *DSatur* é a forma como ele prioriza a escolha dos vértices a serem coloridos, baseada no grau de saturação, que é o número de cores distintas já atribuídas aos seus vizinhos.

O funcionamento do algoritmo segue os seguintes passos:

1. Inicialmente, escolhe-se o vértice de maior grau (ou seja, com mais vizinhos) e atribui-se a ele a primeira cor.

2. Para os próximos vértices, o algoritmo seleciona aquele com o maior grau de saturação, ou seja, o vértice cujo conjunto de vizinhos já contém o maior número de cores diferentes. Se houver empate, o vértice com o maior grau (número total de vizinhos) é escolhido.

3. O vértice selecionado é colorido com a cor de menor valor que não tenha sido usada por seus vizinhos.

4. Repete-se o processo até que todos os vértices sejam coloridos.

O *DSatur* geralmente produz melhores resultados do que algoritmos mais simples, como o *First Fit*, em termos do número de cores usadas, sendo particularmente eficiente em grafos densos. No entanto, é um algoritmo mais complexo e exige mais tempo de execução em comparação com heurísticas mais básicas. Ele é frequentemente usado em aplicações que requerem soluções de coloração otimizadas, como alocação de recursos e escalonamento de tarefas.

Algoritmo 1 DSatur (Saturation Degree)

```
Entrada: G, cores, k
início
  ordena vértices V(G) em ordem decrescente de graus
  v0 ← vértices com maior grau
  cor[v0] ←0
  para vi ∈ {V – {v0}} faça
    vi ←vértice com maior grau de saturação
    cor[vi] ←menor cor disponível
  fim
fim
```

A complexidade do DSatur depende da estrutura específica do grafo e do número de vértices. No pior caso, o algoritmo tem uma complexidade de tempo de $O(n^2)$, onde n é o número de vértices no grafo. Essa complexidade surge das seguintes operações:

- **Cálculo do Grau de Saturação:** Para cada vértice, o algoritmo precisa manter e atualizar um registro do seu grau de saturação. Isso requer verificar os vizinhos de um vértice para determinar quantas cores distintas estão sendo usadas.
- **Seleção de Vértice:** Após cada vértice ser colorido, o DSatur deve computar o grau de saturação para os vértices restantes não coloridos. Como isso requer percorrer os vizinhos de um vértice, o processo de seleção do próximo vértice a ser colorido pode levar até O(n) no pior caso.
- **Atribuição de Cor:** O algoritmo precisa verificar as cores dos vértices vizinhos e encontrar a menor cor disponível. Essa etapa é realizada em tempo constante para cada vértice, uma vez que as cores dos vizinhos foram verificadas.

No geral, a complexidade de tempo é dominada pela necessidade repetida de recalcular os graus de saturação e selecionar o vértice mais restrito, levando a uma complexidade de $O(n^2)$. Embora isso ainda seja exponencial para grafos grandes, o DSatur apresenta um desempenho significativamente melhor na prática em comparação com algoritmos de busca exaustiva.

VI. Resultados

A. Número Cromático

Podemos analisar os resultados obtidos através da implementação dos algoritmos. No gráfico 1, observamos a comparação em termos da métrica de números cromáticos (*chromaticNumber*). O algoritmo que apresentou o melhor desempenho foi o *Dsatur*, principalmente devido à sua estratégia de seleção de vértices baseada no grau de saturação.

Esse enfoque permite que o *Dsatur* se aproxime mais do número cromático mínimo, ou seja, o menor número de cores necessário para colorir o grafo de forma adequada. Isso ocorre porque o algoritmo prioriza vértices com vizinhos já coloridos com diferentes cores, otimizando a coloração e reduzindo a possibilidade de usar cores em vértices.

Em contrapartida, o algoritmo *First Fit*, que segue uma abordagem mais simples e sequencial, tende a utilizar mais cores, uma vez que não possui o mesmo nível de sofisticação na escolha dos vértices e cores. Essa diferença se torna particularmente evidente em grafos de maior complexidade e densidade, onde o *Dsatur* demonstra maior eficiência ao alcançar resultados mais próximos do número cromático real do grafo.

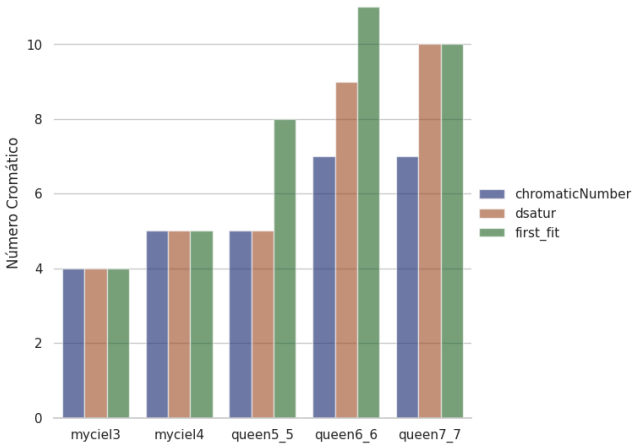

Gráfico 1. Número cromático

B. Tempo de Execução

Quanto ao tempo de execução, podemos ver no gráfico 2 que o *DSatur* apresenta um desempenho inferior em termos de velocidade, em comparação com o *First Fit*. Isso ocorre devido à sua complexidade computacional, que é mais elevada, já que o *DSatur* precisa recalcular constantemente o grau de saturação de cada vértice a cada iteração. Esse processo de priorização e seleção de vértices torna o grafo mais lento, especialmente em grafos grandes e densos, onde o número de vértices e arestas aumenta significativamente.

Em contraste, o *First Fit*, por seguir uma abordagem sequencial e direta, apresenta tempos de execução menores, sendo uma opção mais eficiente em termos de velocidade. No entanto, essa eficiência no tempo de execução vem ao custo de uma solução menos otimizada, resultando em um maior número de cores utilizadas.

Portanto, há uma clara troca entre a qualidade da solução e o tempo de execução: o *DSatur* é mais eficaz para minimizar o número cromático, mas com um tempo de execução maior, enquanto o *First Fit* é mais rápido, porém menos eficiente em termos de otimização de cores. A escolha entre os dois algoritmos dependerá do contexto da aplicação e das necessidades específicas, seja priorizando a otimização do número de cores ou a eficiência computacional.

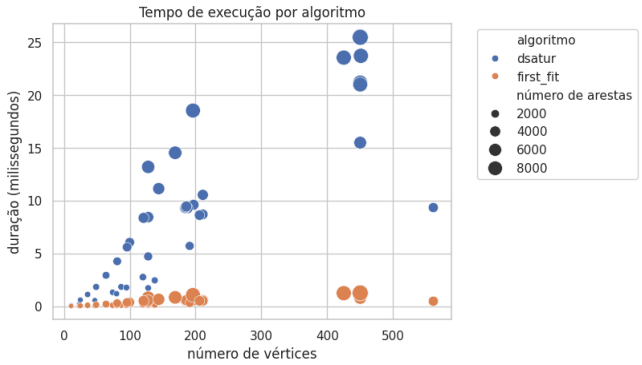

Gráfico 2. Tempo de execução

C. Benchmark

Para alguns dos testes de benchmark realizados com grafos utilizados em estudos de algoritmos, apresentamos os resultados na tabela abaixo. Nela, comparamos o desempenho do algoritmo exato, *DSatur* e *First Fit*, tanto em relação ao número de cores usadas (*cores*), quanto ao tempo de execução em segundos (*T(seg)*). A tabela permite observar como cada algoritmo se comporta em diferentes tipos de grafos, tanto em termos de otimização de cores quanto de eficiência temporal.

Tabela I
Benchmark

Grafo	Exato X		DSatur		FirstFit	
	cores	T(seg)	cores	T(seg)	cores	T(seg)
myciel3	4	0.000051	4	0.000066	4	0.000006
myciel4	5	0.666137	5	0.000178	5	0.000013
queen5_5	5	0.000060	5	0.000590	8	0.000043
queen6_6	7	0.324335	9	0.001102	11	0.000078
queen7_7	7	0.008013	10	0.001816	10	0.000124
miles1500			73	0.021802	76	0.001292
mulsol			49	0.009598	49	0.000547
latinsquare10			130	0.824502	213	0.036444

Número de cores (cores): O algoritmo *Exato* é capaz de encontrar a solução ótima, usando o menor número de cores possível para colorir os vértices. O *DSatur* apresenta um bom desempenho, muitas vezes se aproximando da solução ótima,

enquanto o *First Fit* tende a usar mais cores, especialmente em grafos mais complexos como o *latinsquare10*.

Tempo de execução (T(seg)): O *First Fit* se destaca por ser o mais rápido, com tempos de execução muito menores em comparação aos outros algoritmos, o que confirma sua simplicidade e eficiência computacional. O *DSatur*, embora mais lento que o *First Fit*, ainda apresenta tempos de execução bastante razoáveis para um algoritmo que prioriza a otimização do número de cores.

VII. Conclusão

Em conclusão, este estudo comparou dois algoritmos de coloração aproximada de grafos, *First Fit* e *DSatur*, com base em métricas como número cromático e tempo de execução. Os resultados mostraram que o *DSatur* oferece soluções mais próximas do número cromático mínimo, sendo mais eficiente em termos de otimização de cores, especialmente em grafos complexos. No entanto, essa maior precisão vem acompanhada de um custo computacional mais elevado, com tempos de execução maiores em comparação ao *First Fit*. Este último, por sua vez, destaca-se pela simplicidade e rapidez, embora tenda a usar mais cores, especialmente em grafos de maior densidade.

Portanto, a escolha do algoritmo depende do contexto e dos objetivos da aplicação. Quando a otimização do número de cores é prioritária, o *DSatur* se mostra mais eficaz. No entanto, em cenários onde a eficiência computacional é crucial, o *First Fit* se torna uma alternativa viável, oferecendo uma boa relação entre velocidade e simplicidade de implementação.

O link completo para o projeto se encontra neste link: https://github.com/wandressareis/GeorgeZamboninWandressaReis_ApproximateGraphColoring_AA_RR_2024

Referências

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.

Brélaz, D. (1979). *New Methods to Color the Vertices of a Graph*. Communications of the ACM, 22(4), 251–256. https://doi.org/10.1145/359094.359101

Welsh, D. J. A., & Powell, M. B. (1967). *An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems*. The Computer Journal, 10(1), 85–86. https://doi.org/10.1093/comjnl/10.1.85

Zuckerman, D. (1993). *On Unapproximable Versions of NP-Complete Problems*. SIAM Journal on Computing, 25(6), 1293–1304. https://doi.org/10.1137/S0097539794266407