TRABALHO PRÁTICO 3:
Coloração de grafos

Thompson Moreira Filgueiras

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

thom@dcc.ufmg.br

1. Resumo

Resumo. *Este relatório descreve o problema da coloração de grafos. Assim como apresenta três maneiras distintas para resolver o problema e por fim faz uma comparação entre estes algoritmos, considerando tempo e distância da solução ótima para uma dada estratégia utilizada. É também feita uma avaliação com o custo benefício de utilizar determinada estratégia.*

2. INTRODUÇÃO

O problema da k-coloração de grafos consiste em dado um grafo, encontrar o número mínimo de cores que é possível colorirem seus vértices de tal forma que vértices adjacentes não tenham a mesma cor. Neste trabalho prático foram implementados três algoritmos para resolver o problema da k-coloração de grafos, a saber: tentativa e erro, branch and bound e heurística gulosa. O problema é classificado como Np - difícil.

O número cromático de um grafo é o número mínimo de cores nas quais é possível preenche-lo. Uma estratégia para encontrar este número cromático é testar todas as configurações possíveis. Obviamente este algoritmo é fatorial e uma entrada com 14 vértices demora praticamente um dia inteiro para ser processada. O que prova que esta estratégia é proibitiva para resolver problemas reais.

Uma estratégia gulosa para o problema, consiste em colorir um vértice com uma cor que já tenha sido usada, caso este vértice não esteja ligado a nenhum outro que utilize esta cor. Se não for possível, colore com outra cor. Repete estes passos para os outros vértices.

A estratégia Branch and bound para o problema consiste em utilizar a estratégia tentativa e erro, mas, sempre que uma nova solução for encontrada a memorize caso seja melhor do que as outras. Ao calcular a próxima combinação, pare caso ultrapasse uma solução melhor já encontrada.

O restante deste relatório é organizado da seguinte forma. A Seção 3 discute alguns temas relacionados ao trabalho. A Seção 4 descreve a solução proposta. A Seção 5 trata de detalhes específicos da implementação do trabalho. A Seção 6 contém a avaliação experimental dos três algorítmos. A Seção 7 conclui o trabalho comparando os algorítmos.

3. REFERÊNCIAS RELACIONADAS

Podemos dividir as referências associadas ao problema estudado e à solução proposta dentre os seguintes grupos:

- **Solução da clique máxima de um grafo** Solucionar a clique máxima consiste exatamente em encontrar a k-coloração ótima de um grafo, pois se existe um componente conexo no grafo, então cada vértice deste componente tem que ter uma cor diferente dos outros. Se for encontrado o maior componente conexo do grafo, então esta é a menor coloração possível.

4. SOLUÇÃO PROPOSTA

A solução proposta consiste em primeiramente criar um grafo G, com V vértices e A arestas. Uma vez criado o grafo, foram usadas três estratégias: tentativa e erro, guloso e branch and bound.

A solução tentativa e erro foi refinada para evitar que fossem testadas configurações de cores inconsistentes, ou seja, não colore um vértice com uma cor que desobedece a especificação do problema (vértices adjacentes não podem ter a mesma cor). Teorema: seja G um grafo qualquer. Existe um algoritmo guloso que encontra a k-coloração mínima deste grafo específico, mas que não encontra a melhor coloração para um outro grafo G' qualquer. Isto se deve ao fato do algoritmo guloso escolher uma ordem para colorir um grafo. Prova do teorema: Se não existe uma sequência na qual um grafo pode ser colorido com cores consistentes, então ou este grafo não pode ser colorido por cores consistentes ou não existe solução para o problema. Mas sabemos que existe uma solução para o problema que colore o grafo com cores consistentes $k = V$ é uma solução possível. Então existe uma sequência na qual o grafo pode ser colorido com cores consistentes, logo o teorema está provado. Uma prova mais formal pode ser encontrada em [2]Para o algoritmo de permutação foi utilizado o algoritmo do factoradic, que consiste em gerar um vetor no qual o elemento índice i contém a posição na qual o vetor original deve ficar. Para mais informações sobre factoradic acesse: [**?**]

A solução Branch and bound consiste em partir do pressuposto que o número máximo de cores é igual ao número de vértices. A partir daí ele utiliza o algoritmo guloso para calcular uma solução, salva o número calculado pelo algoritmo guloso e faz permutações idem ao algoritmo tentativa e erro. No entanto se ao calcular o número de cores de uma permutação este número utrapassa a um número menor já calculado ele abandona esta tentativa e parte para a próxima, até efetuar todas as permutações.

A solução gulosa não gera o mínimo de cores sempre, embora dependendo da sequência na qual os vértices são coloridos ela pode gerar a solução ótima. Esta eurística consiste em colorir um vértice com a menor cor possível, levando em consideração as cores dos vértices adjacentes a ele, se já foram coloridos ou não.

A seguir, serão descritas as estruturas de dados e algoritmos propostos, assim como a análise de complexidade dos algoritmos:

4.1. Estruturas de dados

4.1.1. Grafo

O grafo possui o número de vértices, o número de arestas e um vetor de vértices. Cada vértice possui um conjunto de arestas ligadas a ele, estas arestas contém a posição do vetor de vértices na qual este vértice está ligado por esta aresta. O grafo utilizado foi o grafo lista encadeada, na qual cada elemento do vetor de vértices está ligado a uma lista

contendo suas arestas. A principal vantagem desta estrutura é o fato de que para grafos esparços, a quantidade de memória alocada é pequena. Em contrapartida, para grafos conexos é utilizada muito mais memória do que na versão com matizes. Mas mesmo assim a primeira opção foi escolhida, pois no geral um grafo não é conexo.

4.2. Algoritmos

Definições:

- G = grafo qualquer
- A = número de arestas
- V = número de vértices
- B = um vértice qualquer do grafo
- C = um vértice qualquer do grafo

4.2.1. Grafo

Armazena os vértices e as arestas do arquivo de entrada. O algoritmo de criação de um grafo é $O(V * A)$, onde´para cada vértice são inseridas todas as suas arestas. Note que para um grafo não direcionado, são inseridas as arestas que ligam C com B e B com C. Os algoritmos relacionados a grafos foram os mesmos sugeridos pelo Ziviani, e podem ser encontrados na página: [1]

4.2.2. Guloso

O algoritmo guloso consiste em varrer cada vértice, analisando suas arestas. Se a aresta que liga a um vértice usa uma cor, então ele a memoriza em um vetor auxiliar com o tamanho V as cores na qual ele está ligado. Depois varre este vetor procurando pela menor cor que pode ser usada, quando ele encontra esta cor ele colore o vértice com a mesma. Este algoritmo é $O(V * A)$. A cada vez que este algoritmo avalia se uma cor pode ou não ser usada é considerado que ele fez uma tentativa, pois apesar de não ter efetuamente colorido, ele avaliou que não poderia colorir com esta cor.

4.2.3. Tentativa e erro

A estratégia tentativa e erro pode ser dividida em duas partes, a primeira é o cálculo do fatoridic, que é um vetor que guarda a k-ésima permutação de um outro vetor de mesmo tamanho, estas alterações sao salvas no vetor Perm de permutações onde o i-ésimo elemento do vetor permutado é o indice guardado pelo i-ézimo elemento do vetor Perm. Veja o código a seguir onde recebe-se o tamanho do vetor a ser permutado e k, que corresponde a k-esima permutação do vetor original. A complexidade desta função é $O(n2)$

A segunda parte deste algoritmo consiste em realizar todas as permutações possíveis do vetor de vértices. Isto é feito utilizando o vetor perm de permutações. Depois, para cada permutação do vetor aplique o algoritmo guloso usado acima. A complexidade deste algoritmo é $O(V! * (V^2 + V * A)))$, pois para cada permutação é calculado o factoradic, que é $O(V^2)$ e é efetuado também o algoritmo guloso, que é $O(V * A)$

1: coloreGuloso

```
Entrada: G, numVertices
cores = 0;
repita
    repita
        k = valorAresta[j];
        cor = G-¿Vertices[k].cor;
        se cor == Preenchida então
            vAux[cor] = 1;
        fim
    até j de 1 a numArestas ;
    repita
        se(vAux[cor] != 0)  cor = j;
        break;
    até j de 1 a numArestas ;
    G-¿Vertices[i].cor = cor;
    se (cor ¿ cores) então
        cores = cor;
    fim
até i de 1 a numVertices ;
return cores;
```

2: Factoradic

```
Entrada: (
size, k)  repita
    factoradic[size - j] = k MOD(J);
    k = k/j;
até j de 1 a Size ;
repita
    Perm[j] = factoradic[j]+1;
    repita
        se Perm[i] ¿ Perm[j] então
            perm[j]++;
        fim
    até j de 1 a Size ;
até i de 1 a Size ;
return(Perm);
```

3: coloreTentativa

```
Entrada: (
G, numVertices)
cores = 0;
fat = calculaFatorial(numVertices);
repita
    Perm = Factoradic(numVertices, i);
    permutaGrafo(G, perm);
    cor = coloreGuloso(G);
    se (cor ¿ cores) então
        cores = cor;
    fim
até i de 1 a fat ;
return cores;
```

4.2.4. Algoritmo Branch and Bound

O Branch and Bound consiste em refinar a solução tentativa e erro, de tal forma a não continuar a fazer cálculos nos quais já se sabe que a solução será maior do que uma já calculada. Uma forma de realizar esta ”poda”, é alterar o algoritmo guloso para receber um parametro de parada caso ele ultrapasse um número limite de cores passadas por parâmetro. Desta forma a complexidade dos dois algoritmos tanto tentativa e erro quanto branch and bound será a mesma, com a diferença que o branch para de calcular naquela permutação caso a solução já tenha sido encontrada.

4: coloreBranch

```
Entrada: (
G, numVertices)
cores = 0;
limite = numVertices;
fat = calculaFatorial(numVertices);
repita
    Perm = Factoradic(numVertices, i);
    permutaGrafo(G, perm);
    cor = coloreGuloso(G, limite);
    se  cores = cor;
    então cor ¿ cores
      |   1
    fim
    imite = cores;
até i de 1 a fat ;
return cores;
```

5. IMPLEMENTAÇÃO

5.1. Código

5.1.1. Arquivos .c

- **main.c:** Arquivo principal, ele controla o fluxo de execussão principal.
- **interface.c:** Contém as funções modularizadas que normalmente são feitas pelo arquivo main.
- **file.c:** Define as funções relacionadas a leitura e escrita de arquivos
- **grafo.c:** contém as operaçoes relacionadas a um grafo.
- **branch.c:** Contém as funções de branch and bound.
- **guloso.c:** Contém a eurística gulosa para o problema.
- **tentativa.c:** Contém o algoritmo de tentativa e erro do problema

5.1.2. Arquivos .h

- **grafo.h:** Define a estrutura de um grafo usando lista encadeada

5.2. Compilação

O programa deve ser compilado através do compilador GCC através do makefile na pasta raíz deste trabalho ou atravéz do seguinte comando:

```
gcc -Wall main.c interface.c file.c grafo.c branch.c guloso.c tentativa.c -o tp2
```

5.3. Execução

A execução do programa tem como parâmetros:

- Um arquivo que contém os vértices e as arestas de um grafo.
- O nome do arquivo de saida.
- O algoritmo a ser executado 1 = Tentativa e erro, 2 = Branch and Bound ou 3 = Guloso
- Nome do arquivo para salvar as estatisticas de tempo(parametro nao obrigatorio)

  O comando para a execução do programa é da forma:

```
."\tp2 -i<arquivo de entrada> -o <arquivo de saida> -s <algoritmo> -t <arquivo de esta
```

5.3.1. Formato da entrada

A entrada deve conter obrigatoriamente: uma linha com a inicial p, na qual é seguida pela palavra edge e por dois números: número de vértices do grafo e número de arestas do mesmo. Várias linhas contendo a inicial 'e', onde haverão dois números em sequência: o primeiro contém o vértice que está se adicionando uma aresta, a aresta é o segundo número que representa a qual outro vértice este está sendo ligado. Os vértices devem ser números de 1 a número de vértices. Linhas com a inicial 'c' são ignoradas.

```
p edge 4 4
1 2
1 4
2 1
2 3
3 2
3 4
4 1
4 3
```

5.3.2. Formato da saída

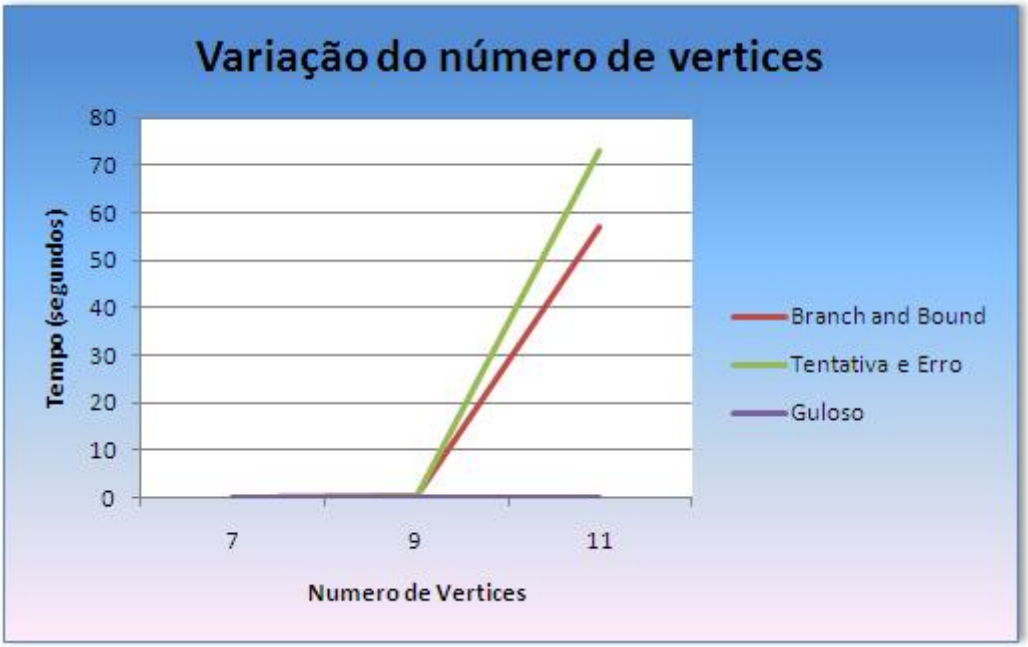
A saída do programa exibe o número de tentativas para calcular a solução e o número cromático encontrado.

```
número cromático: x
número de tentativas: y
```

6. AVALIAÇÃO EXPERIMENTAL

Foram realizadas quatro bateria de testes, cada teste foi realizado dez vezes e os valores obtidos são a média destes 10 testes. A primeira bateria de testes verifica o impacto que existe em variar o número de vértices mantendo-se o número de arestas. Pelo fato do algoritmo ser fatorial, os testes comparativos entre os três ocorreram para entradas com 7, 9 e 11 vértices. A segunda bateria de testes mediu o impacto que existe ao variar o número de arestas mantendo-se o número de vértices. Foram testados um grafo com 7 vértices desconexo, conexo e com 12 arestas. Por fim a ultima bateria de testes avalia a performance do algoritmo guloso, medindo o tempo, o número de tentativas e por fim a distância que a resposta obtida esteve da solução ótima.

O primeiro gráfico de vértices por tentativas mostra que um pequeno aumento de vértices acarreta um grande aumento do número de tentativas para os algoritmos tentativa e erro e branch and bound, o que era de se esperar, pois a complexidade desses algoritmos é fatorial. Enquanto o algoritmo guloso praticamente não aparece no gráfico, pois sua granularidade é muito menor do que os outros algoritmos, afinal, o guloso é somente polinomial. Pelo mesmo motivo, o gráfico de vértices X tentativas apresenta também um aumento absurdo com um pequeno aumento da entrada.



Os dois gráficos a seguir apresentam o impacto existente ao se aumentar o número de arestas para um dada entrada. Observamos que o número de tentativas é praticamente linear com o aumento do número de arestas para os dois algoritmos fatoriais. No entanto o tempo cresceu acima do número de crecimentos, isto pode ser explicado pelo fato de que com mais arestas o algoritmo guloso perde mais tempo verificando quais arestas ele pode ou não considerar, em outras palavras este aumento se deve as constantes da ordem de complexidade.

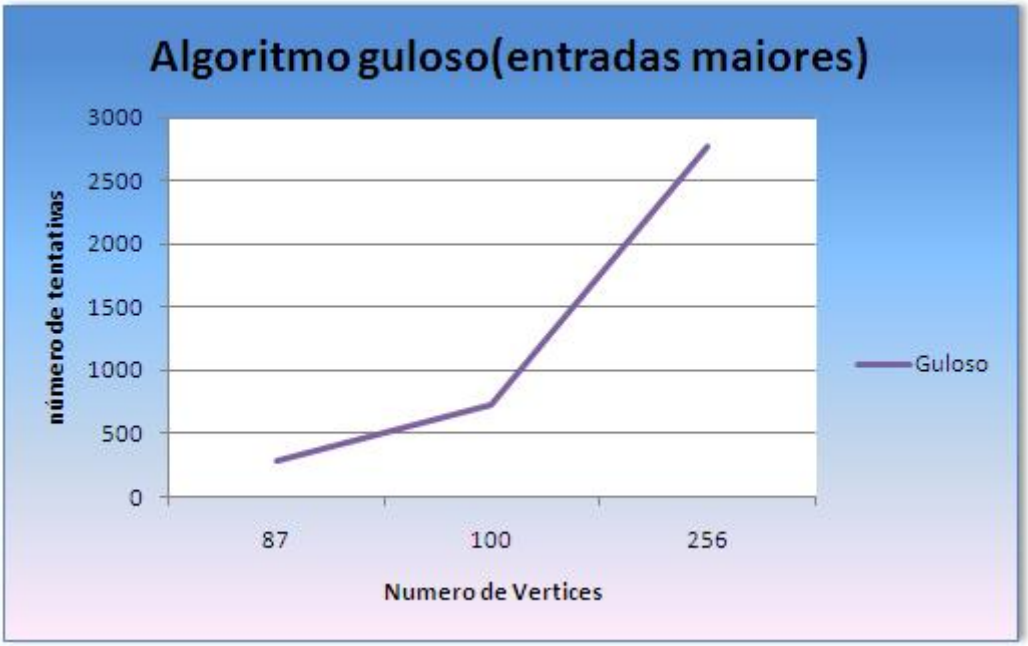
Como era impossível testar entradas grander para os algoritmos fatoriais, os testes com muitas entradas foram realizados com o algoritmo guloso somente. Percebemos pelo

gráfico a mesma curva observada no último gráfico, o que demonstra que aquela variação citada acima está ligada a parte gulosa da solução. Percebemos um aumento de tentativas e de tempo significante do segundo teste para o terceiro, mas isto se deve ao deslocamento do exixo x, que não está corretamente posicionado. Note que a distancia em unidade do primeiro teste para o segundo é bem menor do que do segundo para o terceiro, o que compromete a plotação do gráfico dando uma idéia de que ele realiza uma curva, mas como o comparativo é meramente ilustrativo, é razoável manter o gráfico desta forma, contanto que esta resalva tenha sido feita para uma melhor avaliação do leitor.

A última parte da avaliação consiste em verificar o quão distante o algoritmo gu-

loso ficou da solução exata. Para os testes com grafo desconexo e conexo ele acertou o número cromático, mas neste caso não tinha como ele errar uma vez que não existe outra possibilidade de cores para estes grafos e um eventual erro neste cálculo seria na realidade um erro no algoritmo.

Os testes mais significativos foram o teste com 7 e 11 vértices, no qual o guloso errou por 1 e por 2 o número cromático do grafo que eram respectivamente 3 e 4, mostrando que ele não é perfeito.



7. CONCLUSÃO

Dos três algoritmos o guloso não encontrou a solução ótima do problema, mas foi o único a calcular uma solução em tempo hábil. Se esta fosse uma aplicação para uma empresa, dentre os três ele seria o algoritmo adotado, pois é o único polinomial.

Dentre os outros dois algoritmos, o tentativa e erro é o pior deles tanto em performance, quanto em número de comparações, pois ele continua calculando mesmo quando já encontrou uma solução melhor.

O algoritmo branch and bound é superior ao guloso no sentido que o tempo gasto por ele nos testes foi menor e também superior ao guloso, considerando que ele consegue calcular a solução ótima.

O comparativo do branch and bound com o guloso em questões de performance é injusto, mas o guloso não é capaz de calcular a solução ótima sempre. Neste ponto há um impasse: qualidade X velocidade. Neste caso se for garantido que o número de vértices não será maior do que 10, é recomendável usar o branch and bound. Pelo simples fato dele dar a solução ótima. Para entradas maiores ele é proibitivo e o recomendado é usar o guloso.

Algumas melhorias que poderiam ser consideradas neste trabalho são:

- Ordenação do vetor de vértices colocando os vértices com o maior grau na frente, para a solução gulosa somente.

Referências

[1] Tad grafo. http://www.dcc.ufmg.br/algoritmos/cap7/codigo/c/7.4a7.5e7.8-grafolistaap.c.

[2] Teoria dos grafos. http://www.scribd.com/doc/17139801/Teoria-Dos-Grafos.