UM ALGORITMO EFICIENTE PARA COLORAÇÃO DE ARESTAS BASEADO NO TEOREMA DE VIZING

Tiago de Oliveira Januario [1], Sebastián Urrutia[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais – UFMG
Avenida Presidente Antônio Carlos 6627
Belo Horizonte – Minas Gerais – Brazil

{januario,surrutia}@dcc.ufmg.br

Resumo. *Este artigo apresenta um algoritmo eficiente para a coloração de arestas de um grafo usando $\Delta(G)+1$ cores, onde $\Delta(G)$ é o grau máximo do grafo. A sua implementação é baseada na demonstração do Teorema de Vizing, que prova que todo grafo pode ser colorido com no máximo $\Delta(G)+1$ cores. Foram propostas várias adaptações nas estruturas de dados, e na ordem de escolha das arestas e das cores com o intuito de reduzir o tempo de execução das operações realizadas com maior frequência. A implementação proposta é eficiente em termos de tempo de execução e supera outras implementações disponíveis na literatura.*
PALAVRAS CHAVE. *Coloração de Arestas, Teorema de Vizing, Algoritmos Eficientes, Teoria e Algoritmos em Grafos.*

Abstract. *This paper presents an efficient algorithm for coloring the edges of a graph using $\Delta(G)+1$ colors, where $\Delta(G)$ is the maximum degree in the graph. The implementation is based on the proof of Vizing's Theorem, wich proves that every graph can be colored with at most $\Delta(G)+1$ colors. Several adaptations in data structures and the processing order of edges and colors are proposed in order to reduce the running times of the operations used most often. The proposed implementation is efficient in terms of running time outperforming other implementations available in the literature.*
KEYWORDS. *Edge Coloring, Vizing Theorem, Efficient Algorithms, Theory and Graph Algorithms.*

1. Introdução

A Coloração de Arestas é um problema importante, estudado em Teoria do Grafos, que consiste em colorir as arestas de um grafo de tal forma que arestas incidentes em um mesmo vértice tenham cores distintas e que o número de cores seja mínimo.

Na prática, muitos problemas podem ser resolvidos quando modelados como um problema de coloração de arestas, como o escalonamento de tarefas a serem realizadas em cooperação com dois processadores [Kosowski 2009], operações de transferência de arquivos [Nakano et al. 1995], e alocação de canais em redes de sensores sem fio [Hsu et al. 2006].

Até o momento, poucos trabalhos deram ênfase a uma análise profunda das implementações dos algoritmos baseados na prova do Teorema de Vizing. Tem-se conhecimento

de três implementações disponíveis. Uma delas [Dong's 2010] não é competitiva com as outras duas. O código disponível em [UTA 2010a] é baseado no algoritmo apresentado em [Hu and Blake 1997] enquanto que [UTA 2010b] é baseado no algoritmo apresentado em [Misra and Gries 1992].

Apesar de sua característica polinomial, estes algoritmos não foram testados em grandes grafos. Neste trabalho foram propostas várias adaptações nas estruturas de dados e na ordem de escolha das arestas e das cores com o intuito de reduzir o tempo de execução das operações realizadas com maior frequência.

G denota um grafo simples, não direcionado e finito. O número mínimo de cores necessárias para produzir uma coloração para G é chamado de índice cromático, definido por $\chi'(G)$. $V(G)$ e $E(G)$ representam os conjuntos de vértices e arestas de G, respectivamente. Um subgrafo de G é um grafo G' com $V(G') \subseteq V(G)$ e $E(G') \subseteq E(G)$.

Para cada vértice $v \in G$, $Adj(v)$ denota o conjunto de vértices que são adjacentes a v. O grau de um vértice v é $d(v) = \mid Adj(v) \mid$. O grau máximo de um vértice é $\Delta(G) = max_{v \in V(G)}\{d(v)\}$. Um vértice é universal se $d(u) = \mid V(G) \mid -1$.

Uma atribuição de cores a arestas de um grafo G é uma função $\lambda : E(G) \rightarrow C$. Os elementos do conjunto C são chamados de cores. Um conflito em uma atribuição de cores é a existência de duas arestas com a mesma cor incidindo em um único vértice. Um vértice u é dito ser satisfeito quando $\lambda(uv) = \lambda(uw)$ implica em $v = w$, para todas as arestas em $Adj(u)$. Uma coloração de arestas é uma atribuição de cores de forma que todo vértice é satisfeito, ou de forma equivalente, que não existam conflitos. $H(c_1, c_2)$ é um subgrafo de G induzido pelas arestas coloridas com as cores c_1 ou c_2.

Um grafo é G é dito Classe 1 se $\chi'(G) = \Delta(G)$ e Classe 2 se $\chi(G) = \Delta(G) + 1$. O Teorema de Vizing define que não existe outra possibilidade: todo grafo pertence ou à Classe 1, ou à Classe 2 [Vizing 1964]. [Holyer 1981] prova que determinar qual o número mínimo de cores necessárias para colorir um grafo, entre estes dois valores, é NP-Completo. Demais conceitos em Teoria dos Grafos não apresentados neste texto podem ser encontrados em [West 2001].

Este trabalho está organizado da seguinte forma. Na Sessão 2 é apresentado um algoritmo baseado no Teorema de Vizing. Detalhes sobre as estruturas de dados usadas e sobre os critérios de implementação são dados na Sessão 3. Os resultados experimentais são apresentados na Sessão 4 e na última sessão são apresentadas as conclusões deste trabalho.

2. Algoritmo Implementado

No mínimo $\Delta(G)$ cores são necessárias para colorir as arestas de G, uma vez que um vértice com grau máximo precisa de $\Delta(G)$ cores para ser satisfeito. O principal resultado sobre o estudo de coloração de arestas surgiu em 1964 com o Teorema de Vizing [Vizing 1964]. Este teorema mostra que todo grafo G pode ser colorido com no máximo $\Delta(G) + 1$ cores.

A prova deste teorema, que pode ser encontrada em [Misra and Gries 1992], é baseada no aumento da coloração do grafo através da adição de uma nova aresta colorida a cada iteração. Seja λ uma coloração de arestas parcial de G com $\Delta(G) + 1$ cores, onde "parcial" significa que algumas arestas de G não possuem cores atribuídas. Uma vez que o número de cores excede $\Delta(G)$, todo vértice possui, pelo menos, uma cor não presente

em suas arestas incidentes. Sempre que uma aresta (u, v_0) é selecionada para ser colorida, surgem três casos a serem considerados. No primeiro, a interseção do conjunto de cores não incidentes em ambos os vértices adjacentes ($C(u, v_0)$) é não vazia e a aresta (u, v_0) é colorida com uma cor escolhida do conjunto $C(u, v_0)$. Se esta interseção é vazia, considere uma cor c_1 não incidente no vértice v_0, e uma cor c_2 não incidente no vértice u. A partir destas cores é definido o caminho máximo $c_1/c_2 - path \in H(c_1, c_2)$ partindo do vértice v_0. Se este caminho não termina em u, temos o segundo caso onde as cores c_1 e c_2 podem ser trocadas alternadamente no caminho, e a aresta (u, v_0) pode ser colorida com a cor c_2. Se o caminho termina no vértice u, então existe uma aresta (u, v_1) incidente no vértice u colorida com a cor c_1. Neste último caso, basta remover a cor c_1 da aresta (u, v_1) e inseri-la na aresta (u, v_0). A aresta (u, v_1) é então recolorida com uma cor diferente através da aplicação do mesmo procedimento. O restante da prova do Teorema de Vizing mostra que certamente em algum momento futuro a aresta (u, v_i), com $i \leq \Delta(G)$, será colorida de tal forma a aumentar o número total de arestas coloridas no grafo.

O Algoritmo 1, que colore G com não mais que $\Delta(G) + 1$ cores, é baseado na prova do Teorema de Vizing.

Na prática, existe mais de uma forma de colorir um grafo com não mais que $\Delta(G) + 1$ cores. O Algoritmo 1 inicia com um grafo não colorido. A cada iteração uma aresta de G é escolhida, dentre aquelas que ainda não foram coloridas, através do procedimento $returnUncoloredEdge(G)$.

O procedimento $findUnusedCommonColor(e_1, G)$ tenta encontrar uma cor c_1 não incidente nos vértices v_0 e u para colorir a aresta $e_1(u, v_0)$. O *Caso 1* ocorre quando o procedimento $findUnusedCommonColor(e_1, G)$ consegue determinar uma cor para a aresta e_1, i.e., quando $C(u, v_0)$ não é vazio.

Os procedimentos $findUnusedColor(v_0, G)$ e $findUnusedColor(u, G)$ encontram uma cor c_1 não incidente no vértice v_0, e uma cor c_2 não incidente no vértice u, respectivamente. Umas vez que este procedimento é chamado, pode-se garantir que uma aresta de cor c_1 incide em u e que alguma aresta de cor c_2 incide em v_0.

Após determinar um caminho P partindo do vértice v_0, se P não termina em u, as cores c_1 e c_2 podem ser alternadas neste caminho, e a aresta $e_1(u, v_0)$ pode ser colorida com a cor c_2. Este processo é então chamado de *Caso 2*.

O *Caso 3* ocorre quando o caminho P termina no vértice u. Neste caso, a cor c_1 é removida da aresta $e_2(u, v_1)$ e então a aresta $e_1(u, v_0)$ é colorida com a cor c_1. Na próxima iteração a aresta $e_2(v_1, u)$ será colorida com uma cor diferente.

A escolha das estruturas de dados para a implementação do algoritmo foi baseada na análise das operações realizadas com maior frequência. Aumentar o desempenho da implementação está relacionado com a redução do tempo computacional das seguintes operações:

1. Determinar uma cor $c \in C(u, v_0)$, i.e., uma cor não incidente em ambos os vértices v_0 e u. Esta operação é fundamental para a execução do *Caso 1* do Algoritmo 1.
2. Determinar uma cor $c \in C$ não incidente em um dado vértice. Esta operação é essencial para ambos *Caso 2* e *Caso 3*.
3. Encontrar um vértice w adjacente a u que é conectado por uma aresta de cor c_1. Esta operação é necessária para a obtenção do caminho P.

Algorithm 1: COLORING

```
begin
    coloredEdges ← 0
    e1 ← returnUncoloredEdge(G)
    while coloredEdges <| E(G) | do
        case3 ← false
        v0 ← edge[e1].v
        u ← edge[e1].u
        c ← findUnusedCommonColor(e1, G)
        if there is a color c ∈ C unused at both ends v and u then
            insertColor(e1, c, G)                    /* Caso 1 */
        else
            c1 ← findUnusedColor(v0, G)
            c2 ← findUnusedColor(u, G)
            Determine a maximum path P in H(c1, c2) starting from v0
            if P does not end at u then
                Exchange the colors c1 and c2 in P        /* Caso 2 */
                insertColor(e1, c2, G)
            else                                          /* Caso 3 */
                v1 = color[u][c1].adjacentVertex
                e2 = color[u][c1].edge
                eraseColor(e2, c1, G)
                insertColor(e1, c1, G)
                case3 ← true
        if case3 = true then
            e1 ← e2
            edge[e2].taboo ← c1
        else
            coloredEdges ← coloredEdges + 1
            e1 ← returnUncoloredEdge(G)
```

3. Estruturas de dados e decisões de implementação

Neste artigo foram implementadas várias versões do Algoritmo 1 utilizando a linguagem C e estruturas de dados baseadas em Listas de Arestas para a representação do grafos. Estas estruturas de dados foram projetadas para otimizar as principais operações realizadas ao longo da execução do algoritmo.

3.1. Estruturas de dados

Cada vértice v do grafo G está associado a um conjunto de cores disponíveis $availableColors[v]$ de tamanho máximo $\Delta(G)+1$ que armazena os índices das cores não utilizadas para colorir as arestas incidentes em v.

O vetor de arestas, *edges*, armazena a seguinte informação:

structure *edges*{
    integer v;
    integer u;
    integer $color$;
    integer $taboo$;
}

Dada uma aresta e, os campos v e u do elemento $edges[e]$ representam os vértices conectados pela aresta e e o campo $color$ armazena o índice da cor atribuída para esta aresta. O campo $taboo$ é utilizado, no *Caso* 3 do algoritmo, para impedir que qualquer aresta seja recolorida com uma cor que acabou de ser removida.

Para cada vértice v e cor c, a matriz $colors$ armazena a seguinte estrutura:

structure *colors* {
    boolean $status$;
    integer $adjVertex$;
    integer $edge$;
    integer $colorIndex$;
}

Dado um vértice v e uma cor c, o campo $colors[v][c].status$ é definido como *true* sempre que uma aresta colorida com a cor c incide no vértice v, caso contrário é definido como *false*. O campo $colors[v][c].adjVertex$ armazena o índice do vértice u adjacente ao vértice v conectado por uma aresta colorida com c. O campo $colors[v][c].edge$ armazena o índice da aresta que conecta os vértices v e u e o campo $colors[v][c].colorIndex$ armazena o índice da cor c dentro do conjunto $availableColors[v]$.

Esta estrutura permite determinar qual o vértice adjacente a v que é conectado por uma aresta de cor c, juntamente com o índice desta aresta, em tempo $O(1)$. Caso a cor c não incida no vértice v, e o algoritmo decida colorir uma de suas arestas incidentes com a cor c, é possível remover esta cor do conjunto $availableColors$ em tempo $O(1)$.

Foram propostas duas implementações do conjunto de cores disponíveis $availableColors$, uma baseada em um vetor de índices de cores e outro baseado em *strings de bits*. Ambos são avaliados em termos de eficiência para atualização e recuperação das informações armazenadas.

As atualizações das informações sobre as cores das arestas incidentes a um dado vértice v são facilmente realizadas na implementação que utiliza o vetor de índice de cores. Caso uma cor c seja removida, o campo $colors[v][c].colorIndex$ apontará para a posição desta cor no conjunto $availableColors$. Se esta mesma cor estiver sendo adicionada, basta inseri-la no final do vetor $availableColors$ e atualizar esta informação em $colors[v][c]$. Estas operações são executadas em tempo $O(1)$ em ambos os casos. Determinar uma cor disponível do conjunto $availableColors$ também é fácil, basta acessar um elemento válido dentro do vetor. No entanto, determinar uma cor disponível em dois vértices distintos não é tão fácil, uma vez que é preciso percorrer a lista de cores disponíveis de um dos vértices, verificando se esta cor está disponível no outro vértice utilizando o campo $status$ da estrutura $colors$. Toda esta operação é executada em tempo $O(\Delta)$.

Na implementação com *string de bits* foi utilizado um vetor de inteiros de 32-*bits* de tamanho $\lceil \frac{\Delta(G)+1}{32} \rceil$, chamado $availableBitColor$. Cada inteiro neste vetor representa 32 cores consecutivas. O j-ésimo bit do i-ésimo inteiro do vetor representa a disponibilidade

da cor $32 \times i + j$. A atualização da cor c no conjunto de cores disponíveis no vértice v também é realizada em tempo $O(1)$ uma vez que basta calcular a sua posição no vetor de *string de bits* e definir o seu novo valor. Observe que ao usar esta nova estratégia os campos $status$ e $colorIndex$ na estrutura $color$ tornam-se desnecessários.

Determinar uma cor disponível a partir do *string de bits* é mais difícil, pois é preciso encontrar um elemento não-zero dentro vetor de números inteiros e após encontra-lo, determinar um bit 1. Para determinar um bit 1 em um *string de bits* rapidamente, as posições do bit mais significativo dos números decimais de 1 a 2^{16} foram pré-calculadas e armazenadas na tabela $bitTable$. Se o inteiro não-zero i não é superior a 2^{16}, a posição do seu correspondente bit mais significativo é $bitTable[i]$, caso contrário, é de $bitTable[i >> 16] + 16$.

A operação para determinar uma cor disponível em dois vértices distintos justifica esta implementação. Para encontrar esta cor basta determinar um bit 1 nos dois *string de bits*. Para executar esta operação podemos escolher um índice nos vetores de inteiros e utilizar o operador AND para verificar se existe uma interseção entre os conjuntos de cores de cada vértice representado pelo inteiro. A seguir a $bitTable$ é usada para determinar o bit mais significativo definido como 1. A cor representada por este bit está disponível em vértices. Caso o resultado do operador AND não retorne um número não-zero, basta escolher outro índice e repetir o procedimento. Mesmo que esta operação tenha tempo $O(\Delta)$, na prática é mais rápido do que a mesma operação para a implementação usando vetor de índices de cores.

3.2. Decisões de implementação

Foram analisadas a influência da ordem de seleção das arestas na função $returnUncoloredEdge$ e a seleção de uma cor disponível em um determinado vértice ou em um dado par de vértices. Estas decisões de implementação influenciam o desempenho do algoritmo alterando o número de vezes que o algoritmo entra nos casos 1, 2 e 3. Uma vez que o *Caso* 1 envolve um menor número de operações computacionais, é desejável que o algoritmo encontre uma cor para a aresta a ser colorida neste caso.

Foram avaliadas duas possibilidades para a ordem de seleção das arestas: a ordem lexicográfica (sequencial) e ordem aleatória. Considera-se a seleção em ordem lexicográfica quando todas as arestas incidentes ao vértice v_i são selecionadas para serem coloridas antes de qualquer aresta incidente ao vértice v_{i+1}, com exceção da aresta (v_i, v_{i+1}). Também foram avaliados dois tipos de seleção de cores disponíveis em $availableColors[v]$: primeira disponível e aleatória. Como é possível observar, quatro algoritmos são avaliados.

4. Resultados

O algoritmo descrito na Sessão 2 foi codificado na linguagem C e compilado com o compilador *gcc* na versão 4.4.1 utilizando o parâmetro de otimização -O3. Os códigos disponíveis em [UTA 2010a] e [UTA 2010b] foram compilados com o mesmo compilador e mesmos parâmetros.

Foram gerados aleatoriamente 50 grafos densos com o número de vértices variando em $n = 100 \times i$, $i = 1, ..., 50$ onde a probabilidade de ligação entre as arestas é de 99% sem a presença de nós universais.

Todos os algoritmos foram executados na mesma plataforma Pentium Core 2 Duo, com 5 Gbytes de memória RAM e 3.0 GHz de processamento.

4.1. Resultados Computacionais

O primeiro experimento foi realizado através da implementação que utiliza vetor de índice de cores, e quatro algoritmos foram testados levando em conta as seguintes decisões de implementação:

1. CSAS: As arestas são selecionadas em ordem lexicográfica e a primeira cor disponível é sempre escolhida;
2. CSAA: As arestas são selecionadas aleatoriamente e a primeira cor disponível é sempre escolhida;
3. CAAS: As arestas são selecionadas em ordem lexicográfica a cor é escolhida aleatoriamente;
4. CAAA: Tanto as arestas quanto as cores são selecionadas aleatoriamente.

As Figuras 1 e 2 mostram o número de vezes que o algoritmo troca o fluxo de execução para os *Caso 2* e *Caso 3*, respectivamente. A partir destas figuras é possível observar que uma certa aleatoriedade é essencial para a eficiência do algoritmo. Na versão do algoritmo sem qualquer escolha aleatória, nota-se que o número de vezes que o algoritmo troca o fluxo de execução para o *Caso 2* ou *Caso 3* é muito maior que em qualquer uma das outras versões.

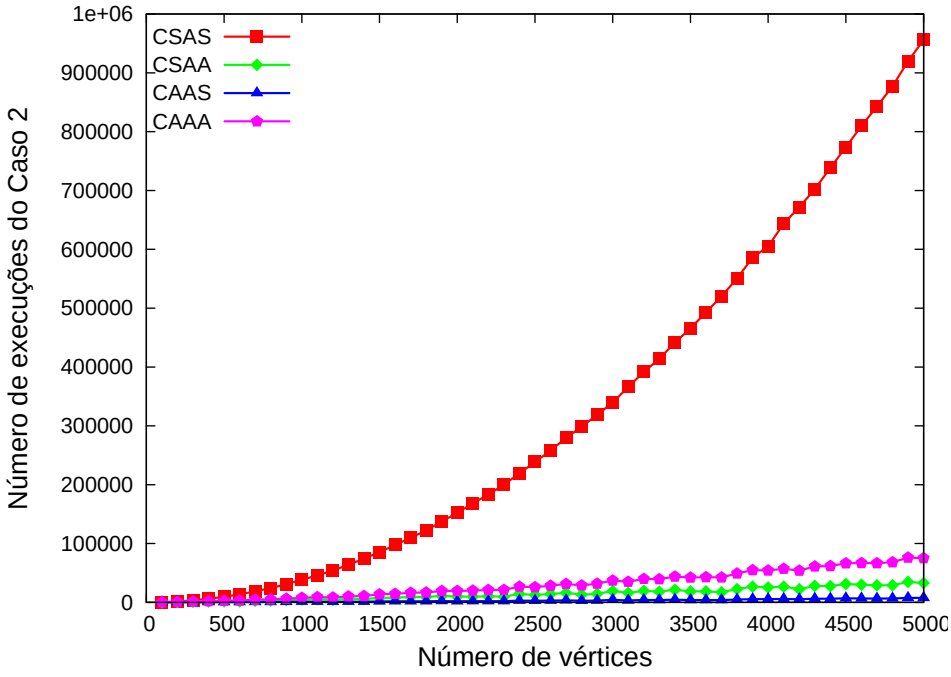

Figura 1. Número de execuções do *Caso 2* para diferentes versões do algoritmo.

O segundo experimento analisou a distribuição das cores no grafo para cada uma das versões do algoritmo. Foram determinados os valores de desvio padrão para o total de ocorrências de cada cor o ao longo do processo de coloração em uma instância com 5000 vértices. A Figura 3 mostra os valores de desvio padrão encontrados. As estratégias CSAA e CSAS apresentam os maiores valores de desvio padrão uma vez que as primeiras cores

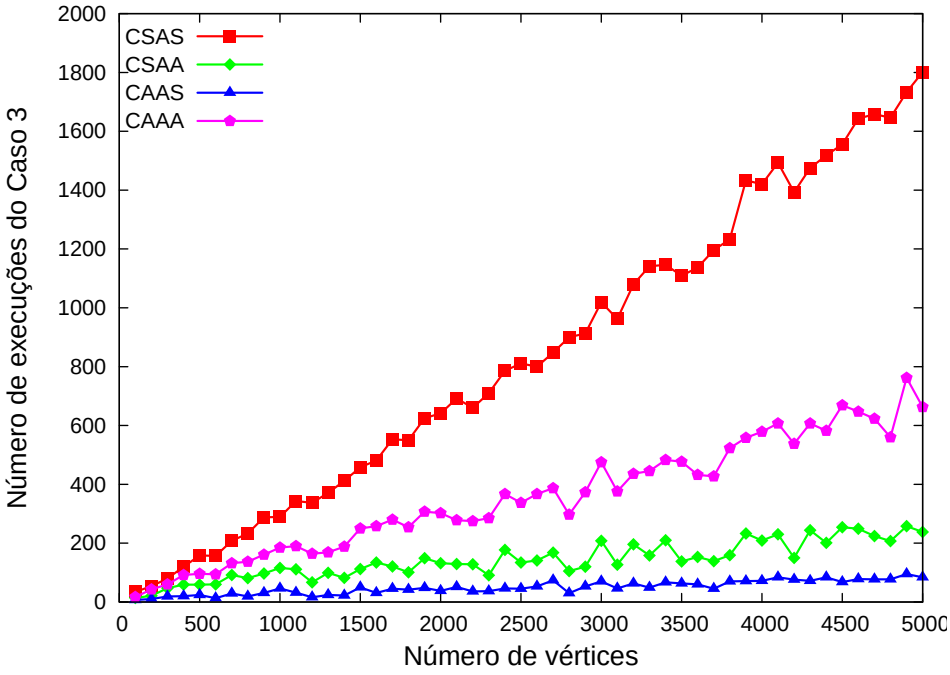

Figura 2. Número de execuções do *Caso 3* para diferentes versões do algoritmo.

disponíveis são as inicialmente escolhidas para a coloração, promovendo um desequilíbrio na sua distribuição. Nas implementações que utilizam as estratégias CAAS e CAAA a seleção das cores é feita de forma aleatória, e os resultados mostram que estas implementações possibilitam uma distribuição balanceada das cores no grafo ao longa do processo de coloração.

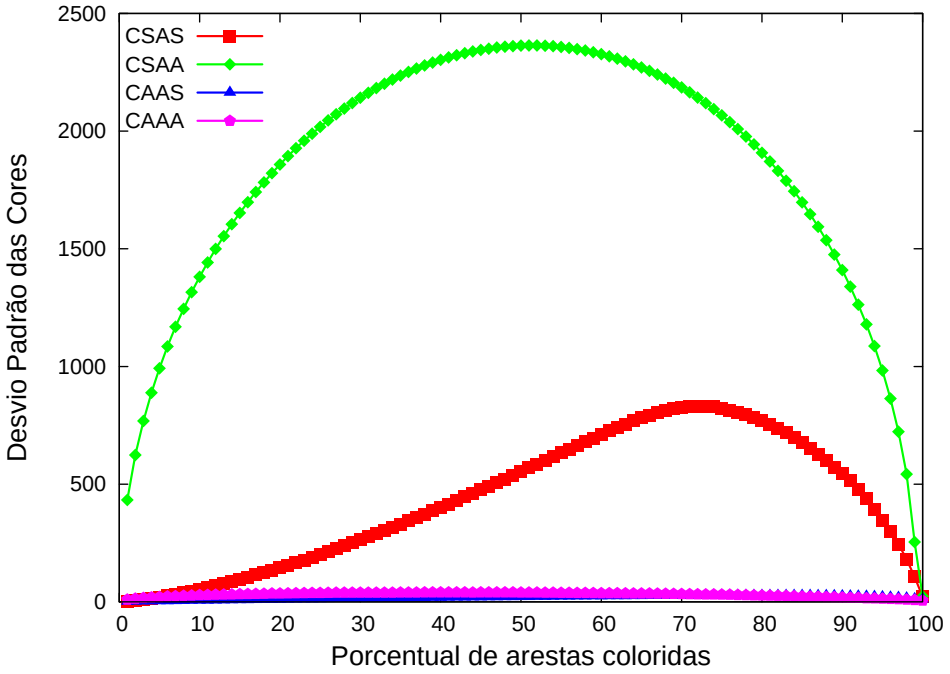

Figura 3. Análise da coloração para um grafo com 5000 vértices.

A Figura 4 mostra que CAAS é melhor versão do algoritmo, em termos de tempo de execução. Este resultado é esperado, uma vez que esta é a versão que troca o fluxo de execução para o *Caso* 2 e no *Caso* 3 o menor número de vezes e por contribuir para a obtenção homogênea da distribuição de cores.

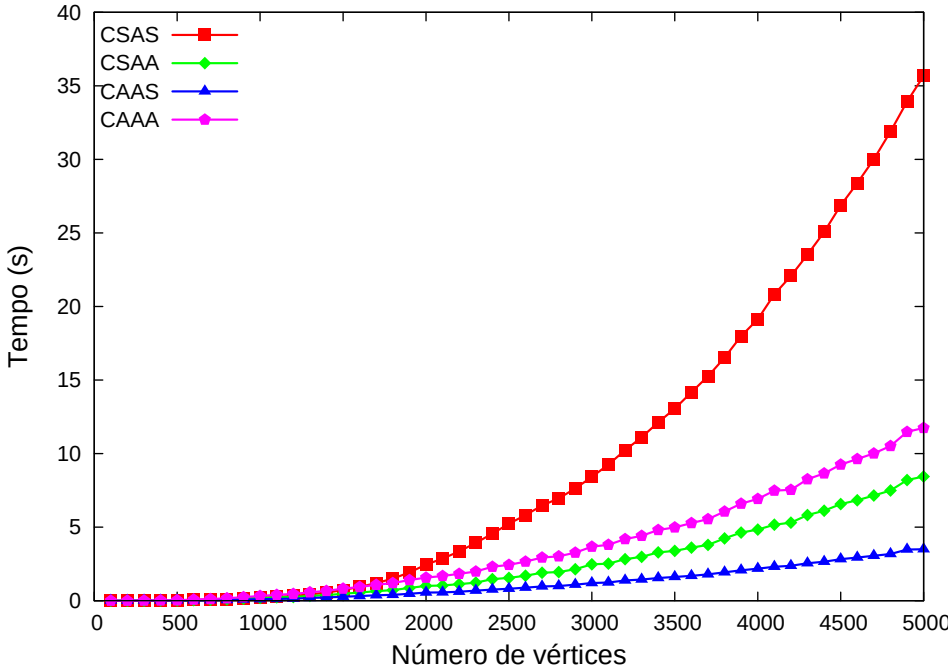

Figura 4. Tempos de execução de diferentes versões do algoritmo.

No terceiro experimento são comparadas as duas estruturas de dados usadas para representar o conjunto $availableColors[v]$. Com base nos dois primeiros experimentos, todos os algoritmos consideram a estratégia CAAS em suas implementações.

A Figura 5 compara os tempos computacionais para as três implementações: vetor de índice de cores, vetor de inteiros de 32-*bits* e vetor de inteiros de 64-*bits* e os resultados mostram que a implementação que utiliza *string de bits* é mais eficiente que a implementação de vetor de índices de cores.

Mesmo com a aleatorização da escolha de cores dentro do *string de bits* com o uso de SHIFT rotacional, através da Figura 6 é possível observar que a implementação que utiliza inteiros de 32-*bits* promove uma distribuição mais homogênea das cores ao longo da coloração que a de 64-*bits*, mas ambas apresentam desvio padrão superior ao algoritmo que considera a estratégia CAAS com o uso de vetor de índice de cores.

No último experimento foram comparados os algoritmos propostos (32-*bits*, CAAS) com as duas melhores implementações encontradas na literatura [UTA 2010b, UTA 2010a]. A Figura 7 mostra os tempos de execução dos algoritmos e é possível ver claramente que a implementação proposta supera as outras duas implementações conhecidas.

5. Conclusões

Neste trabalho foi apresentada uma implementação eficiente de um algoritmo baseado no Teorema de Vizing para a coloração de arestas de um grafo utilizando não mais que $\Delta(G)+$

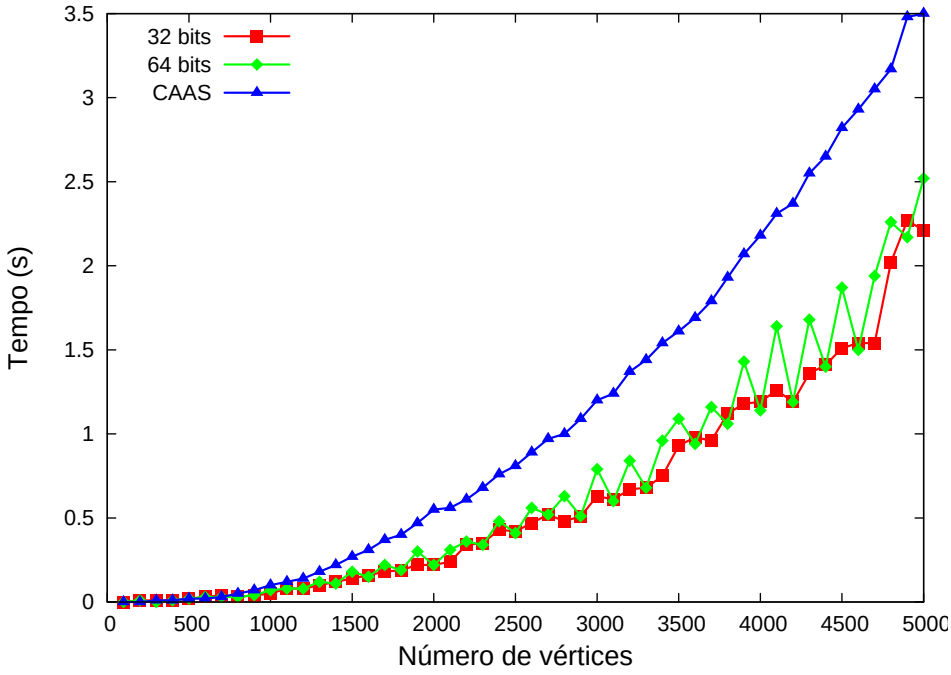

Figura 5. Comparação entre os melhores algoritmos implementados.

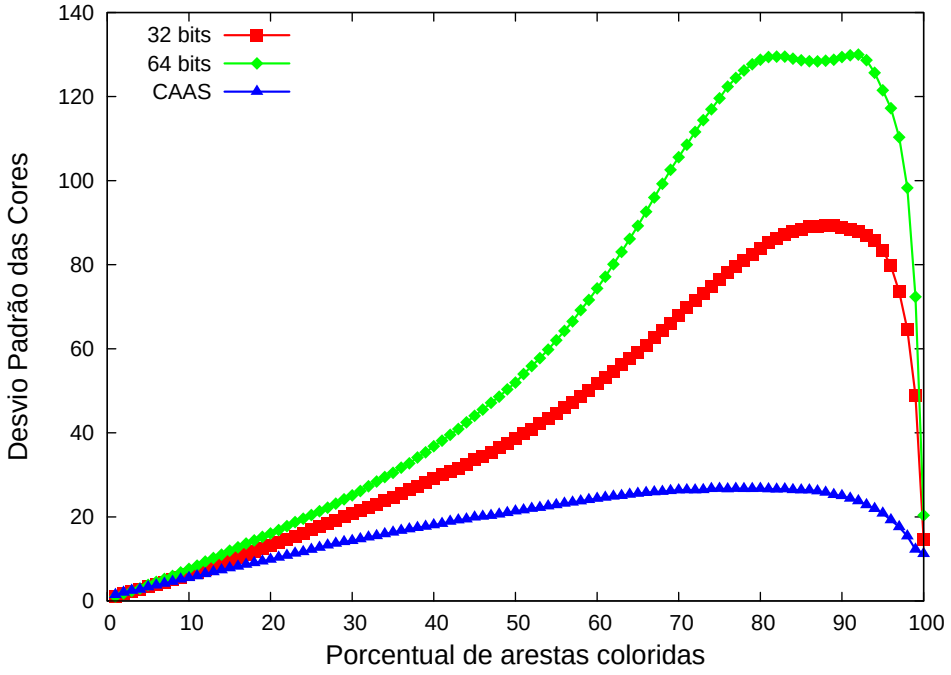

Figura 6. Análise da coloração para um grafo com 5000 vértices.

1 cores. Foram propostas duas estruturas de dados para a representação do conjunto de cores disponíveis em um dado vértice. Várias estratégias de ordem de processamento das arestas e das cores foram experimentalmente testadas.

Como mostrado nos experimentos, a implementação proposta apresentou alto desempenho em termos de tempo de execução quando comparada com as implementações

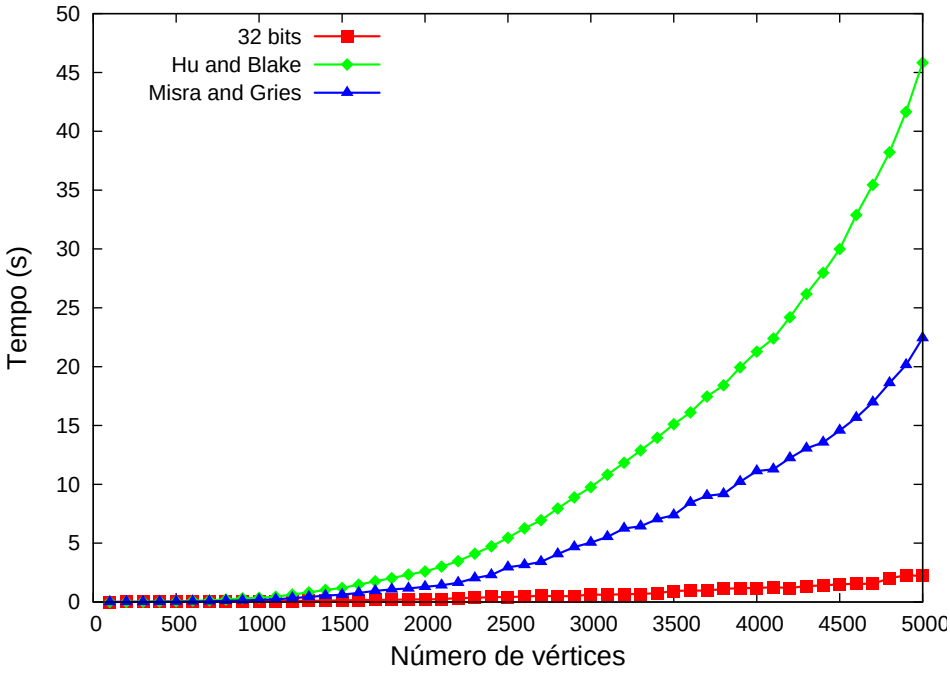

Figura 7. Comparação com as implementações da literatura.

disponíveis na literatura. A representação das cores disponíveis para a coloração de uma dada aresta com o uso de inteiros de 32-*bits* permitiu a redução do tempo computacional gasto para encontrar uma cor comum não incidente em dois vértices adjacentes. A seleção das arestas em ordem lexicográfica e das cores aleatoriamente mostrou-se adequada para o problema de coloração de arestas uma vez que esta abordagem contribuiu para a obtenção de uma distribuição homogênea das cores ao longo do grafo, que em termos, aumenta a probabilidade do algoritmo trocar o fluxo de execução para o *Caso 1* do algoritmo.

Referências

Dong's, Y. (2010). Stony brook project implementations. `http://www8.cs.umu.se/kurser/TDBA77/VT06/algorithms/WEBSITE/IMPLEMEN/STONY/DISTRIB/VIZING/`. [Online; accessed 10-April-2010].

Holyer, I. (1981). The np-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718 – 720.

Hsu, C.-C., Liu, P., wei Wang, D., and Wu, J.-J. (2006). Generalized edge coloring for channel assignment in wireless networks. *International Conference on Parallel Processing*, pages 82–92.

Hu, Y. F. and Blake, R. J. (1997). Algorithms for scheduling with applications to parallel computing. *Adv. Eng. Softw.*, 28(10):563–572.

Kosowski, A. (2009). Approximating the maximum 2- and 3-edge-colorable subgraph problems. *Discrete Appl. Math.*, 157(17):3593–3600.

Misra, J. and Gries, D. (1992). A constructive proof of vizing's theorem. *Inf. Process. Lett.*, 41(3):131–133.

Nakano, S., Zhou, X., and Nishizeki, T. (1995). *Computer Science Today*, chapter Edge-coloring algorithms, pages 172 – 183.

UTA (2010a). Prototype vizing (degree+1) edge colorer. `http://reptar.uta.edu/NOTES4351/edgeColor4.c`. [Online; accessed 10-April-2010].

UTA (2010b). Prototype vizing (degree+1) edge colorer. `http://reptar.uta.edu/NOTES5311/misraGriesNew.c`. [Online; accessed 10-April-2010].

Vizing, V. G. (1964). On an estimated of the chromatic class of a p-graph. *Diskrete Anal*, 3:25 – 30.

West, D. B. (2001). *Introduction to Graph Theory (2nd Edition)*. Prentice Hall.