PROBLEMA E ALGORITMOS DE COLORAÇÃO EM GRAFOS - EXATOS E HEURÍSTICOS

COLORING ALGORITHMS AND PROBLEM IN GRAPHS - EXACT AND HEURISTIC

Alfredo Silveira Araújo Neto
Universidade Estadual do Ceará/Brasil
alfredosilveira@yahoo.com.br

Marcos José Negreiros Gomes
Universidade Estadual do Ceará/Brasil
negreiro@graphvs.com.br

Abstract: **Context: Given a graph G and an integer $k > 0$ the graph coloring problem consists to use a set of $n \leq k$ colors to assign to each node of G a certain color, so that adjacent vertices have distinct color. Despite the existence of accurate methods capable of determining the staining solution to the problem, it is observed that these techniques have an exponential complexity in some cases. The limitations presented by exact methods when applied to instances of greater complexity motivated the development of several heuristics, able to indicate satisfactory solutions for graphs in general. This paper describes the problem of coloring graphs, presents some of the techniques that can be used in their determination, besides presenting some real problems that can be modeled as a graph G and solved based on the result achieved by applying a coloring on G.**

***Keywords*: Coloring of graphs; Agorithms; Heuristics**

Resumo: **Contexto: Dados um grafo G e um inteiro $k > 0$, o problema da coloração em grafos consiste em utilizar um conjunto de $n \leq k$ cores para atribuir a cada vértice de G uma determinada cor, de modo que vértices adjacentes tenham cores distintas. A despeito da existência de métodos de exatos, capazes de determinar a solução do problema de coloração, observa-se que estas técnicas possuem em alguns casos complexidade exponencial. A limitação apresentada pelo métodos exatos quando aplicados a instâncias de maior complexidade motivou o desenvolvimento de diversas heurísticas, aptas a indicar soluções de maneira satisfatória para grafos em geral. Este trabalho descreve o problema da coloração em grafos, apresenta algumas das técnicas que podem ser utilizadas na sua determinação, além de enumerar alguns problemas reais que podem ser modelados a partir de um grafo G e solucionados com base no resultado alcançado pela aplicação de uma coloração sobre G.**

Palavras-chave: **Coloração de grafos; Algoritmos; Heurísticas.**

I. INTRODUÇÃO

Suponha a tarefa de colorir os países da América Sul, representados por meio do mapa da figura 1, de tal maneira que quaisquer dois países com uma fronteira em comum tenham que ser coloridos com cores distintas. Quantas cores, no mínimo, seriam necessárias para execução desta tarefa? A despeito de saber-se que o número mínimo de cores necessário para este fim é quatro, pode-se afirmar que isto é verdadeiro para qualquer mapa [4]?

Embora não seja difícil colorir um mapa da América do Sul com o emprego de apenas quatro cores, esta tarefa torna-se impossível se tentarmos utilizar somente três cores, pois os países Argentina, Bolívia, Brasil e Paraguai, por apresentarem fronteiras em comum, demandam no mínimo quatro cores distintas para sua representação, conforme as restrições impostas pelo problema [4].

Não obstante a apresentação dos países vizinhos em cores diferentes possa ser justificada pelo argumento de permitir uma melhor visualização dos itens representados no mapa, poderia não estar claro, entretanto, um raciocínio que sugerisse que quatro seria o número mínimo de cores necessário para colorir os países de qualquer mapa. De fato, seria provavelmente possível imaginar um mapa complicado, contendo uma grande quantidade de países, alguns dos quais com numerosos países vizinhos, construído de tal forma que possivelmente somente um grande número de cores seria suficiente para colori-lo completamente [4].

Embora possa parecer somente uma curiosidade, este problema despertou por muito tempo o interesse e a atenção de muitos pesquisadores, desta forma, as tentativas e iniciativas em solucioná-lo contribuíram de modo significativo para o desenvolvimento de uma área da matemática conhecida como Teoria dos Grafos. Como conseqüência, o problema da coloração acabou por adquirir uma denominação que o

tornaria conhecido em todo o mundo matemático: o Problema das Quatro Cores [4].

Fig. 1 - Mapa da América do Sul [4].

O problema das quatro cores pode ser descrito como um problema de decisão que tem como objetivo indicar se os países representados em um mapa qualquer, podem ser coloridos com quatro cores ou menos, de modo que países com uma fronteira em comum sejam coloridos diferentemente. Muitos dos conceitos, teorias e problemas da teoria dos grafos são decorrentes do problema das quatro cores, e, a despeito do mesmo ser apresentado como um mapa real ou imaginário, com fronteiras que delimitam vizinhanças entre países, estados, províncias, ou distintas unidades geográficas, o mesmo conceito pode ser estendido para quaisquer outras divisões de caráter geral [4].

O problema da coloração foi proposto em 1852 por Francis Cuthrie, através de uma carta escrita ao seu irmão Federick, que na ocasião era aluno do matemático Augusto De Morgan. Francis presumia em sua carta que aparentemente qualquer mapa desenhado em uma folha de papel poderia ser colorido com apenas quatro cores sem que os países com fronteiras em comum tivessem a mesma cor, e, por meio da missiva, questionava ao seu irmão se haveria alguma maneira de provar este fato matematicamente. Frederick apresentou o problema a De Morgan, porém este não foi capaz de estabelecer uma prova, e, a despeito do grande esforço de pesquisa empregado ao longo dos 124 anos seguintes, ninguém teve a capacidade de apresentar um mapa em que cinco cores fossem necessárias ou de estabelecer a prova formal de que quatro cores seriam suficientes [3].

O problema das quatro cores somente viria se resolvido em 1976 por Kenneth Appel e Wolfang Haken, com o emprego de uma técnica de prova por computador, que utilizou resultados já estabelecidos da teoria dos grafos juntamente com certo raciocínio matemático adicional para demonstrar que o problema geral pode ser reduzido a um número finito de casos particulares, dotados de certas propriedades que permitem estabelecer que a coloração pode ser efetuada com somente quatro cores. Por meio do uso de programas de computador, em torno de 1.700 instâncias do problema foram cuidadosamente criadas, e, após 1.200 horas de processamento verificou-se que todos os casos necessitaram de no máximo quatro cores para solução, ficando então este resultado conhecido como o Teorema das Quatro Cores [3].

Não obstante outras demonstrações semelhantes tenham sido apresentadas posteriormente, uma importante questão ainda permanece em aberto: existirá uma prova não-enumerativa do Teorema da Quatros Cores, ou seja, que possa ser obtida sem a utilização do computador [3]?

Problemas de natureza algorítmica para os quais existem soluções com tempo de execução polinomial são considerados tratáveis, enquanto aqueles que comprovadamente não podem ser solucionados por algoritmos polinomiais são denominados intratáveis. Os problemas tratáveis constituem a classe P ao passo que os problemas que somente podem ser resolvidos com o emprego de algoritmos enumerativos constituem a classe NP [3].

A existência em NP de uma classe extensa de problemas denominada de $NP-Completa$ estimulou os estudos de complexidade e teoria dos algoritmos, além de promover o desenvolvimento de heurísticas, que podem ser definidas como quaisquer métodos ou técnicas criados para solucionar determinados tipos de problemas [3] [10].

O fato de um dado problema de programação ser classificado como $NP-Completo$ constitui uma forte evidência de que não existem algoritmos polinomiais que possam ser utilizados em sua resolução, justificando portanto, o emprego de métodos enumerativos ou desenvolvimento e uso de heurísticas [3].

Uma coloração em um grafo $G=(V,E)$ consiste na atribuição de cores aos vértices em V de modo que vértices adjacentes tenham cores distintas. A entrada do problema é composta por um grafo G e um inteiro $k>0$ enquanto que a saída resume-se em determinar se há uma coloração com k cores ou menos. Em grafos planares existe uma correspondência entre este problema e o problema da coloração de mapas, no qual somente é possível distinguir regiões vizinhas se as mesmas tiverem cores diferentes [3] [10].

A solução do problema do número cromático em grafo G tem como objetivo determinar o menor valor de k que possibilite uma coloração, enquanto que no problema do índice cromático deseja-se colorir as arestas de G de maneira que duas arestas adjacentes tenham cores distintas. De acordo com o Teorema de Vizing o índice cromático de G é igual

a δ para grafos bipartidos e δ ou $(\delta + 1)$ para grafos em geral, onde δ é o grau máximo dos vértices de G [3].

O problema do índice cromático pode ser solucionado por meio de um algoritmo polinomial para algumas classes de grafos, a exemplo dos grafos bipartidos, porém este comportamento não é verificado para grafos arbitrários. Desta maneira, embora o valor de δ possa ser obtido para qualquer grafo em tempo polinomial o problema de decisão associado, isto é, verificar se o índice cromático de G é igual a δ é um problema $NP - Completo$, sendo de difícil solução mesmo para pequenos valores de δ. De modo semelhante, dados um grafo $G = (V, E)$ e um inteiro positivo $k \leq |V|$, determinar se Gé k-colorível também constitui um problema de decisão classificado como $NP - Completo$ [3][10].

II. NÚMERO CROMÁTICO DE UM GRAFO

O problema de coloração de grafos que tem recebido maior atenção da literatura implica em colorir os vértices de um grafo, uma cor para cada vértice, de modo que os vértices adjacentes sejam coloridos de forma diferente. Embora as cores usadas na representação do problema possam ser elementos de qualquer conjunto, cores reais (tais como verde, vermelho, azul e amarelo) somente são utilizadas quando um pequeno número de cores está sendo considerado. Em muitas situações inteiros positivos $(1,2,\dots,k)$ são empregados para representar as cores e a razão desta substituição é que na maioria das vezes o principal interesse consiste em determinar somente a quantidade de cores utilizada na solução do problema [4].

De modo formal uma coloração consiste em função $c: V(G) \rightarrow \mathbb{N}$, tal que $c(u) \neq c(v)$, se u e v são adjacentes em G. Se cada cor utilizada no grafo for uma das k cores estabelecidas, denomina-se a coloração resultante como uma k-coloração [4].

Suponha que c é uma k-coloração de um grafo G, na qual os inteiros $1, 2, \dots, k$ representam as cores que estão sendo utilizadas. Se V_i $(1 \leq i \leq k)$ é o conjunto de vértices em G, destacados com a cor i, então cada conjunto não vazio V_i é nomeado como uma classe de cor e todos os elementos não vazios de $\{V_1, V_2, \dots, V_k\}$ constituem uma partição de $V(G)$. Uma vez que não existem dois vértices adjacentes de G associados por c à mesma cor i, cada uma das classes de cor V_i $(1 \leq i \leq k)$ representa um conjunto independente de vértices de G [4].

Um grafo G é k-colorível se houver uma coloração de G construída a partir de um conjunto de k cores. O menor inteiro positivo k para o qual G é k-colorível é o número cromático de G. Este valor é representado por $\chi(G)$ e determina o número mínimo de conjuntos independentes por meio dos quais $V(G)$ pode ser subdividido. Um grafo G com um número cromático k é um grafo k-cromático, desta forma, se $\chi(G) = k$ existe uma k-coloração de G embora não exista uma $(k-1)$-coloração. Um grafo é k-colorível se e somente se $\chi(G) \leq k$, e, necessariamente, se uma k-coloração de um grafo k-cromático é dada, então todas as k cores estão sendo utilizadas [4].

De maneira geral para demonstrar que um dado grafo G tem um número cromático k é necessário mostrar que existe uma k-coloração de G $(\chi(G) \leq k)$ e que cada coloração de G requer pelo menos k cores $(\chi(G) \geq k)$. Não existe uma fórmula geral para determinar o número cromático de um grafo, por conseguinte, a análise deve estar concentrada em identificar o número cromático de alguns grafos específicos, ou de grafos que pertencem a alguma classe de interesse, ou ainda distinguir os limites superior ou inferior do número cromático de um grafo. Decerto, para um grafo G qualquer de ordem n, $1 \leq \chi(G) \leq n$ [4].

Para determinadas categorias de grafos o número cromático é um valor que pode ser presumido de modo trivial. Por exemplo [3]:

- Se K_n for um grafo completo com n vértices então $\chi(K_n) = n$;
- Se T for um uma árvore (grafo conexo sem ciclos) com pelo menos 2 vértices, então $\chi(T) = 2$;
- $\chi(K_{n,m}) = 2$ se $K_{n,m}$ for um grafo bipartido com pelo menos uma aresta;
- Se C_n for um grafo com n vértices formando um único ciclo então $\chi(C_n) = 2$, quando n for par e $\chi(C_n) = 3$, quando n for ímpar.

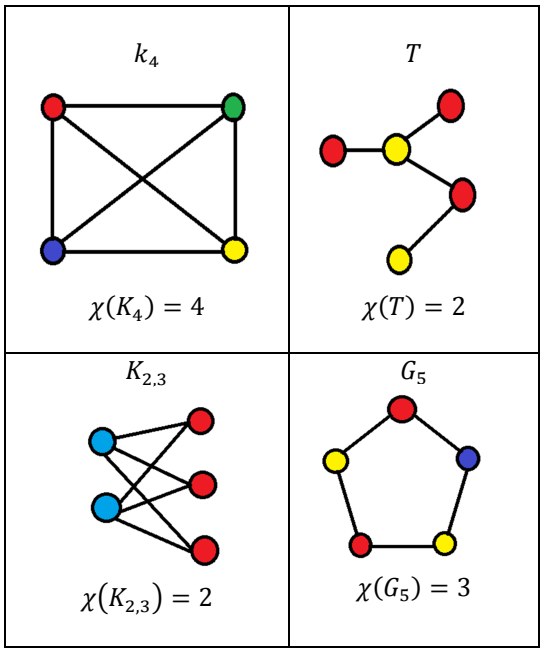

Fig. 2 - Categorias triviais quanto a $\chi(G)$ [3].

III. ESTADO DO CONHECIMENTO

As primeiras abordagens heurísticas empregadas na solução do problema da coloração de grafos eram baseadas em métodos gulosos, os quais realizavam a coloração seqüencial dos vértices guiados por uma função gulosa pré-definida. Embora sejam muito rápidos, a qualidade das soluções obtidas são em geral insatisfatórias e dentre os algoritmos que fazem parte desta classe destacam-se as heurísticas *Degree of Saturation* (DSATUR) e *Recursive Largest First* (RLF). Nas últimas décadas, essas heurísticas gulosas têm sido freqüentemente utilizadas para gerar soluções iniciais para meta-heurísticas mais avançadas [9].

Em contrapartida, algoritmos baseados em meta-heurísticas de busca local têm sido amplamente utilizados para abordar o problema da coloração. Um dos exemplos mais representativos é o então denominado algoritmo *Tabucol* o qual representou a primeira aplicação da meta-heurística *Tabu Search* na resolução do problema da coloração de grafos. O método *Tabucol* foi posteriormente melhorado por outros pesquisadores e passou a ser empregado como um dos elementos de algoritmos de coloração mais sofisticados. Diferentes meta-heurísticas baseadas em busca local incluem ainda *Simulated Annealing*, *Iterated Local Search*, GRASP e *Clustering-Guide Tabu Search*, entre outras [9].

Em paralelo ao desenvolvimento dos métodos fundamentados em busca local, pesquisadores propuseram diferentes técnicas para solucionar o problema da coloração, em particular para grafos de grandes dimensões, construídos de maneira randômica. Uma das mais recentes e promissoras abordagens consiste na hibridação que incorpora algoritmos de busca local à métodos baseados em algoritmos evolucionários a fim de obter um melhor equilíbrio entre intensificação e diversificação [9].

Outra forma de solucionar o problema da coloração para grafos de maiores dimensões consiste em extrair do grafo original diversos sub-conjuntos independentes e então determinar a coloração aplicando o método escolhido ao grafo residual. Esta técnica é particularmente útil para grafos realmente extensos, sendo utilizada em geral como uma etapa de pré-processamento à aplicação de outros algoritmos de coloração [9].

IV.ALGORITMOS

Uma vez que o problema da determinação de uma partição cromática mínima é classificado como $NP - Completo$, qualquer algoritmo exato empregado em sua resolução terá complexidade exponencial, desta maneira, torna-se importante se dispor de técnicas heurísticas capazes de identificar soluções de maneira satisfatória [1].

Considere $G = (N, M)$ um grafo simples não orientado com n vértices e m arestas e $\Gamma(x)$ o conjunto dos vértices adjacentes a $x \in N$. Os algoritmos heurísticos capazes de colorir os vértices de G em ordem decrescente dos graus são conhecidos como seqüenciais, e, a seguir, descrevem-se algumas das técnicas que empregam esses e outros critérios na determinação da partição cromática [3].

A. Algoritmo de Welsh e Powell

Foi proposto por Welsh e Powell em 1976. É considerado de natureza míope, visto que determina a cor do vértice j após os $j - 1$ vértices terem sido coloridos, tendo sempre como propósito minimizar o número de cores necessárias [3].

Algoritmo WP $(G = (N, \Gamma))$
1 ler $\{n, \Gamma(j), j = 1, 2, \dots, n\}$
2 organizar os vértices $\{x_1, x_2, \dots, x_n\}$ de modo que $|\Gamma(x_j)| \geq |\Gamma(x_{j+1})|, j = 1, 2, \dots, n$
3 para $j = 2$ até n faça
4 $\quad C_j \leftarrow \emptyset$
5 fim-para
6 $C_1 \leftarrow \{x_1\}$
7 para $j = 2$ até n faça
8 $\quad k \leftarrow mínimo\{i | \Gamma(x_j) \cap C_i = \emptyset, i = 1, 2, \dots, n\}$
9 $\quad C_k \leftarrow C_k \cup \{x_j\}$
10 fim-para
11 $k \leftarrow máximo\{j | C_j \neq \emptyset, j = 1, 2, .. n\}$
12 escrever k

A intenção principal desta heurística é determinar o número cromático $\chi(G)$ analisando os vértices em ordem decrescente de seus graus, de forma que na j -ésima iteração os vértices $x_{1,}x_{2,}\dots,x_{j-1}(1 \leq k \leq j-1)$ já terão sido examinados e coloridos com k cores. Para o vértice x_j, se houver alguma cor $i, 1 \leq i \leq k \leq j-1$, tal que $\Gamma(x_j) \cap C_i = \emptyset$, então esta cor pode ser atribuída ao vértice x_j, ou seja, $C_i \leftarrow C_i \cup \{x_j\}$. De outro modo atribui-se a cor $(k+1)$ a x_j, isto é, $C_{k+1} \leftarrow C_{k+1} \cup \{x_j\}$. A probabilidade de ocorrer $\Gamma(x_j) \cap C_i \neq \emptyset$, para $1 \leq i \leq k$, será diretamente proporcional ao grau do vértice x_j e a obtenção da coloração é garantida pois, para cada vértice $x_j \in N$ ao qual é atribuída a cor i, tem-se $\Gamma(x_j) \cap C_i = \emptyset$ [3].

B. Algoritmo de Coloração Seqüencial

Este algoritmo considera a existência de uma classe de cor $C_i, i = 1,2,..,n$ para cada vértice x_i do grafo G que estarão, no primeiro momento, vazias. Durante sua execução o algoritmo inicializa sucessivamente as classes de cores por meio da inserção individual dos vértices na classe de menor ordem k, desde que a interseção entre o conjunto de vizinhos do vértice e os elementos que pertencem à classe seja vazia. Quando não se consegue colocar o vértice em uma das classes já inicializadas o algoritmo promove a abertura de nova classe de ordem $k+1 \leq n$. A complexidade do algoritmo de coloração seqüencial é $O(n+m)$, onde n é o número de vértices e m o número de arestas [1].

Algoritmo CS $(G = (N, \Gamma))$

```
1   para i = 1 até n faça
2      C_i ← ∅
3   fim-para
4   k ← 1
5   para i = 1 até n faça
6      enquanto não atribuir x_i a C_k faça
7         se Γ(x_i) ∩ C_k = ∅ então
8            C_k ← C_k ∪ {x_i}
9         senão
10           k ← k + 1
11        fim-se
12     fim-enquanto
13     k ← 1
14  fim-para
```

De modo similar ao que ocorre com outros algoritmos heurísticos, o resultado alcançado será dependente da rotulação adotada para os vértices. Por exemplo, com a primeira rotulação da figura 3 a seguir obtém-se uma 4 -coloração, enquanto que com a segunda rotulação o algoritmo computa uma 3-coloração (mínima) [1].

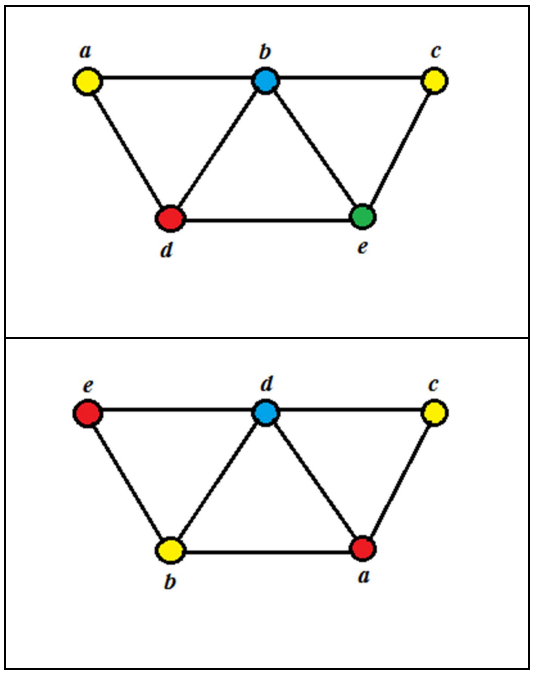

Fig. 3 - Efeito da rotulação no resultado da coloração [1].

Se sucessivas rotulações forem realizadas, inevitavelmente a coloração mínima será obtida, porém existem $n!$ possíveis rotulações, o que implica em um tempo de execução exponencial. Um forma de atenuar este efeito consiste em rotular os vértices na ordem decrescente dos graus dos seus vértices, aumentando o número de incompatibilidades verificadas em cada etapa com a conseqüente redução da cardinalidade da coloração resultante [1].

C. Algoritmo de Coloração por Classe

Esta técnica também considera a existência de uma classe de cor $C_i, i = 1,2,..,n$ para cada vértice x_i do grafo G que estarão inicialmente vazias. Durante a execução do algoritmo cada classe de cor é organizada de uma só vez por meio da inserção dos vértices na classe de menor ordem k, desde que a interseção entre o conjunto de vizinhos dos vértices e os elementos que pertencem à classe seja vazia. Em cada etapa, os vértices que forem rejeitados serão candidatos à formação da classe de ordem seguinte $k+1 \leq n$. A complexidade deste método também é $O(n+m)$. [1].

Algoritmo CC $(G = (N, \Gamma))$

```
1   para i = 1 até n faça
2      C_i ← ∅
3   fim-para
4   Y ← X
5   k ← 1
6   enquanto Y ≠ ∅ faça
```

```
7       para x_i ∈ Y faça
8         se Γ(x_i) ∩ C_k então
9           C_k ← C_k ∪ {x_i}
10          Y ← Y − {x_i}
12        fim-se
12    ....fim-para
13      k ← k + 1
14    fim-enquanto
```

D. Algoritmo DSATUR

Diversas variações da heurística seqüencial são propostas na literatura, diferindo em geral somente pelo método empregado na ordenação dos vértices. Algumas dessas variações exibem melhores resultados computacionais, e, dentre elas, destaca-se a heurística *Degree of Saturation* (DSATUR), a qual foi proposta por Brélaz em 1979 [3].

Seja G um grafo e C uma coloração parcial dos vértices de G, define-se o grau de saturação de um vértice v como o número de diferentes cores apresentadas pelos vértices adjacentes a v. O algoritmo DSATUR, assim denominado em função do emprego do grau de saturação, pode ser então descrito como um procedimento que compreende a execução das seguintes etapas [2]:

1. Ordene os vértices de G em ordem decrescente de graus;
2. Atribua ao vértice de maior grau a cor 1;
3. Selecione o vértice com maior grau de saturação. Se houver vértices com mesmo grau de saturação, opte por qualquer um de grau máximo pertencente ao sub-grafo ainda não colorido;
4. Atribua ao vértice selecionado a cor de menor índice disponível;
5. Se todos os vértices estiverem coloridos, pare. Caso contrário, retorne à etapa 3.

O pseudo-código para o algoritmo DSATUR é apresentado a seguir.

```
Algoritmo DSATUR (G = (N, Γ))
1   ler {n, Γ(j), j = 1, 2, ..., n}
2   organizar os vértices {x_1, x_2, ..., x_n} de
    modo que |Γ(x_j)| ≥ |Γ(x_{j+1})|, j =
    1, 2, ..., n
3   para j = 2 até n faça
4     C_j ← ∅
5   fim-para
6   C_1 ← {x_1}
7   enquanto houver vértice sem cor faça
8     calculaVertice(W)
9     k ← máximo{i | i ∈ W}
10    j ← 1
11    enquanto x_k não tiver cor faça
12      se Γ(x_k) ∩ C_j = ∅ então
13        C_j ← C_j ∪ {x_k}
14      senão
15        j ← j + 1
16      fim-se
17    fim-enquanto
18  fim-enquanto
```

O procedimento $calculaVertice$ atualiza o conjunto W com a relação dos índices dos vértices de G de maior grau de saturação que ainda não estão coloridos. Em seguida, k recebe o valor do maior elemento de W a fim de garantir que um vértice arbitrário seja sempre selecionado, até mesmo nas situações em que o grau máximo de saturação seja comum a mais de um vértice de G.

O algoritmo DSATUR possui complexidade $O(n^2)$ é exato para grafos bipartidos sendo ainda capaz de produzir bons resultados quando aplicado a grafos em geral por ser um método de baixa complexidade [3] [8].

E. Algoritmo de Zykov

O problema da coloração também é passível de ser solucionado por meio do emprego de algoritmos exatos os quais podem ser obtidos utilizando-se a chamada árvore de Zykov. Dado um grafo $G = (N, M)$, uma árvore de Zykov Z_G pode ser construída a partir da aplicação iterativa em G de dois tipos de alterações estruturais: adição e condensação. Para que as operações de adição e condensação tenham significância, o grafo G deve conter pelo menos dois vértices não adjacentes $u, v \in N$, pois, do contrário, o problema torna-se trivial visto que para um grafo completo, $G = K_n, \chi(K_n) = n$ [3].

Durante a construção da árvore de Zykov, que na verdade é uma arborescência, as operações de adição e condensação são aplicadas em vértices não adjacentes de G, produzindo grafos homoformos do grafo original, ou seja, grafos que preservam as relações de adjacência [1].

Dado um grafo G, $G_1 = \beta_{u,v}(G)$ será o grafo obtido pela adição da aresta que une os vértices não adjacentes u e v, enquanto que $G_2 = \gamma_{u,v}(G)$ é o grafo resultante da condensação de u e v. A operação de adição preserva o número de vértices e aumenta em uma unidade o número de arestas, ao passo que a condensação minora o número de vértices em uma unidade e reduz a uma única aresta todo par de arestas $(u, z), (v, z)$ onde $z \in \Gamma(u) \cap \Gamma(v)$ [1] [3].

G	$G_1 = \beta_{u,v}(G)$

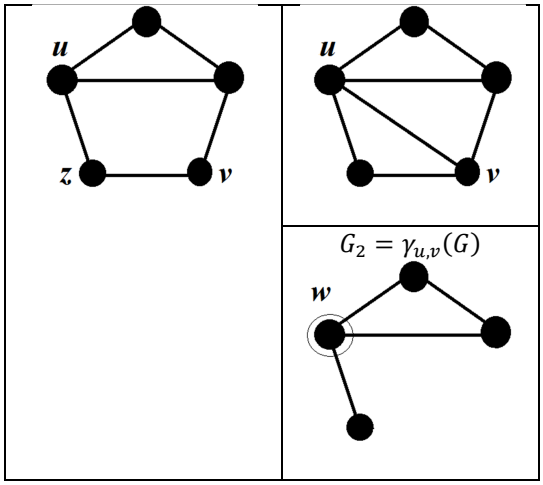

Fig. 4 - Adição e condensação [3].

É possível concluir que, pela natureza das alterações estruturais aplicadas em G, $\chi(G)$ será o mínimo entre os números cromáticos dos grafos assim formados, ou seja, $\chi(G) = min\{\chi(\beta_{u,v}(G)), \chi(\gamma_{u,v}(G))\}$ [3].

No intuito de determinar os números cromáticos de $\beta_{u,v}(G)$ e $\gamma_{u,v}(G)$, as operações de adição e condensação são repetidas de maneira recursiva, até que todos os grafos obtidos sejam cliques. As arestas mais à direita de Gformam um caminho constituído pela seqüência de condensações, aplicadas da raiz até uma folha K_r, o qual é denominado seqüência de condensações completa de Z_G [3].

Desta forma, uma árvore rigorosamente binária Z_G é construída, na qual cada vértice corresponde a um grafo da forma $\beta_{u,v}(G')$ ou $\gamma_{u,v}(G')$ e cuja a raiz está relacionada ao grafo original G. Esta é a então denominada árvore de Zykov e o número cromático de G será correspondente ao número de arestas do menor clique existente entre suas folhas [3].

O algoritmo de Zykov apresentado a seguir utiliza o procedimento recursivo $Cor(G', \rho)$ aplicado um grafo $G' = (N', M')$, onde a variável ρ armazenará ao final da execução o tamanho do menor clique [3].

Procedimento $Cor\ (G' = (N', M'), \rho)$
1 $q \leftarrow |N'|$
2 se $G' = K_q$ então
3 $\rho \leftarrow min\{\rho, q\}$
4 senão
5 $Obter(u, v) \notin M'$
6 $Cor\ (\beta_{u,v}(G'), \rho)$
7 $Cor\ (\gamma_{u,v}(G'), \rho)$
8 fim-se

Algortimo Zykov $(G = (N, M), \rho)$
1 ler $\{G = (N, M)\}$
2 $\rho \leftarrow |N|$
3 $Cor(G, \rho)$
4 $\chi(G) \leftarrow \rho$
5 escrever $\chi(G)$

A figura a seguir ilustra a aplicação do algoritmo de Zykov sobre um grafo G cujo número cromático $\chi(G) = 2$ corresponde à ordem do homomorfismo de comprimento 2, criado pela seqüência de condensações que resultaram na imagem homomorfa $h(G) = K_2$.

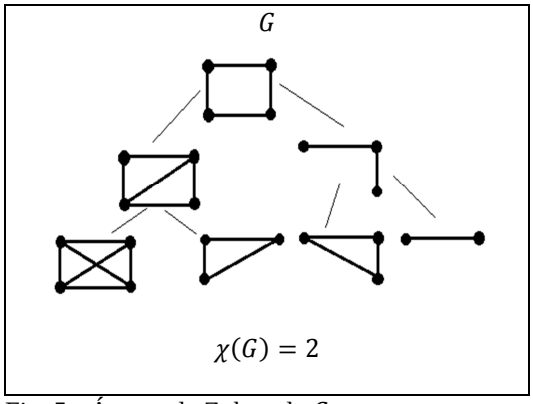

Fig. 5 - Árvore de Zykov de G.

F. Algoritmo de Dutton e Brigham (DB)

Consiste numa heurística de ordem $O(n^3)$ formulada por Dutton e Brigham em 1981 e que também utiliza a árvore de Zykov em sua implementação. A heurística constrói um caminho na árvore por meio de operações de condensação realizadas de tal maneira que maximizam as possibilidades de se obter a folha na qual o clique é mínimo. A qualidade do caminho obtido resulta da seleção dos vértices v_1 e v_2 de G, sobre os quais será aplicada a operação de condensação a fim de se obter $G_2 = \gamma_{v_1,v_2}(G)$. Desta forma, no intuito de antecipar a formação do clique, seleciona-se o par de vértices com o maior número de vértices adjacentes em comum [3].

No contexto deste método um bom algoritmo deve selecionar os pares de vértices de modo a adiar ao máximo possível a formação de um grafo completo. Como em cada condensação o número de vértices de G é minorado em somente uma unidade deve-se optar pela escolha do par de vértices que promova a maior redução possível na quantidade de arestas com o propósito de maximizar as operações de condensação disponíveis nas iterações subseqüentes [5].

O algoritmo proposto parte de um grafo inicial G, constituído por um conjunto de vértices $\{v_1, v_2, \dots, v_n\}$, e, a cada iteração, reduz o tamanho de G em um vértice. Em um dado momento durante a execução o vértice v_i representa não somente o vértice original v_i, mas também todos os vértices que foram direta ou indiretamente condensados em v_i pelas iterações precedentes [5].

Algortimo Dutton e Brigham

1 Para todos os vértices não adjacentes v_i e v_j calcule c_{ij} , número de vértices adjacentes em comum
2 Se não houver mais vértices não adjacentes, pare. Se ainda houver determine o par v_i e v_j para o qual $c_{ij} \geq c_{rs}$ para cada par de vértices não adjacentes v_r e v_s
3 Condense v_i e v_j. Ajuste os valores de c_{rs} para todos os pares não adjacentes afetados pela condensação. Faça $n \leftarrow n - 1$ e repetia a etapa 2

O valor de n obtido quando da parada do algoritmo, determinado na etapa 2, representará a estimativa do número cromático do grafo e para $1 \leq i \leq n$ todos os vértices condensados em v_i podem ser associados a i-ésima cor [5].

G. Algoritmo Cosine

Foi apresentado por Hertz em 1991. É uma variação do algoritmo DB na qual a escolha do par de vértices u, v a ser condensado é determinada pelo seguinte critério: se o vértice resultante da operação de condensação anterior não estiver conectado a todos os vértices do grafo $\gamma_{u,v}(G)$ então este vértice é selecionado como u. De modo contrário, a operação de condensação é aplicada ao par u, v, onde u é escolhido aleatoriamente (ação sempre tomada na primeira iteração) e v é o vértice que maximiza o número de vizinhos em comum, ou seja $|\Gamma(u) \cap \Gamma(v)|$, entre os vértices não adjacentes u e v [3].

O algoritmo Consine é descrito a seguir [8]:

Algortimo Cosine$(G = (N, M))$

1 $G_0 \leftarrow G$
2 $k \leftarrow 0$
3 enquanto G_k *não for um clique* faça
4 se *existe pelo menos um vértice não adjacente a* $(uv)_k$ então
5 $u_k \leftarrow (uv)_k$
6 senão
7 *selecione para* u_k *qualquer vértice que não seja adjacente a todos os outros vértices de* G_k
8 fim-se
9 *escolha para* v_k *qualquer vértice não adjacente a* u_k *para o qual* $|\Gamma(u_k) \cap \Gamma(v_k)|$ *é máximo*
10 *construa* G_{k+1} *condensado* u_k, v_k *em um vértice* $(uv)_{k+1}$
11 $k \leftarrow k + 1$
12 fim-enquanto
13 *atribua cores a* G_k
14 enquanto $k \neq 0$ faça
15 $k \leftarrow k - 1$
16 *descomprima* G_{k+1} *atribuindo a* u_k *e* v_k *a mesma cor de* $(uv)_{k+1}$
17 fim-enquanto

O procedimento é interrompido quando o grafo original G é convertido em um clique. Seja então k o número de vértices do clique final. Colorindo este clique com k cores, descomprimindo o grafo e atribuindo a u e v a mesma cor de (uv) obtém-se a k-coloração do grafo de G [8].

Os algoritmos DSATUR, DB e Cosine podem ser utilizados para determinar um limite superior para o número cromático de G que posteriormente pode ser melhorado por meio do seguinte método heurístico de coloração [8]:

1 $k \leftarrow$ *limite superior de* $\chi(G)$, *calculado pelos métodos DSATUR, DB ou Cosine*
2 $coninue \leftarrow verdadeiro$
3 enquanto $coninue$ faça
4 *use o método heurístico* $\mathcal{M}$ *para tentar determinar uma* $(k-1)$*-coloração de* G
5 se $\mathcal{M}$ *produziu uma* $(k-1)$*-coloração de* G então
6 $k \leftarrow k - 1$
7 senão
8 $continue \leftarrow falso$
9 fim-se
10 fim-enquanto

H. Algoritmo Las Vegas

O problema da coloração de grafos que, conforme já descrito, consiste em determinar para um grafo G e um inteiro k se $\chi(G) \leq k$, é um problema de decisão $NP - Completo$, e, para algumas de suas instâncias, não pode ser solucionado por um método com tempo de execução polinomial. Uma maneira de sobrepujar a $NP - Completude$ de um problema consiste em elaborar algoritmos que, embora não sejam capazes de obter soluções para todas as instâncias, possam funcionar bem para a maioria dos casos. São apresentadas na literatura duas categorias de algoritmos que podem ser empregadas na abordagem de problemas classificados como $NP - Completos$ [6].

Um algoritmo de decisão do tipo Monte Carlo sempre terminará sua execução em um tempo polinomial, produzindo a resposta correta para quase todas as instâncias do problema, porém com uma pequena probabilidade de obter uma resposta equivocada. Um algoritmo Las Vegas corresponderá àquele que sempre obterá uma resposta correta e que quase sempre terminará em um tempo polinomial, ou seja, há uma pequena chance do algoritmo demandar um tempo de

execução não-polinomial. Algoritmos da classe Las Vegas são atrativos quando a presença de uma resposta incorreta for inaceitável, sendo preferível assumir o risco do término não-polinomial à obtenção de uma solução de qualidade imprevisível [3].

Algoritmo	Tempo Polinomial	Resposta Correta
Monte Carlo	Sempre	Quase sempre
Las Vegas	Quase sempre	Sempre

Tabela 1 - Características dos algoritmos Las Vegas e Monte Carlo [3].

O algoritmo de coloração Las Vegas é um método de tempo de execução quase polinomial capaz de determinar se, para um grafo G e um inteiro k, G é k-colorível. O algoritmo cria e transversa uma árvore de Zykov oriunda de G. As principais características deste algoritmo são [6]:

- Percorre conforme necessário toda a árvore de Zykov. Quando uma folha, isto é, um clique, de dimensão maior do que k é alcançado uma operação de retrocesso é realizada e a busca continua;
- Em cada ramificação da árvore, formada pela adição de arestas ou condensação de vértices, um teste é realizado a fim de antecipar o surgimento de um $(k+1)$-clique. O teste é rápido, uma vez que é restrito à parte do grafo que foi submetida às operações de adição de arestas e condensação de vértices. Conseqüentemente, a obtenção de uma resposta negativa pode estar incorreta, visto que é possível existir um $(k+1)$-clique em alguma outra porção do grafo não avaliada pelo teste. A aplicação do teste tem dois propósitos principais: em primeiro lugar ele possui a tendência de podar a árvore de maneira a reduzir o espaço de busca de soluções; em segundo lugar ele assegura que o procedimento termine de forma correta para grafos que não são k-coloríveis;
- A maneira como o par de vértices é selecionado para condensação e adição de arestas é de fundamental importância para o método e será descrita em seguida.

Procedimento Colorir $(G=(N,M),k)$

```
1   se G ≤ k vértices então
2      encontrouCor ← verdadeiro
3   senão
4      Obter(u, v)
5      G2 ← Condensar(γu,v(G))
6      G1 ← Criar(βu,v(G))
7      se TesteClique(G2, k) = falso então
8         Colorir(G2, k)
9      fim-se
10     se encontrouCor = falso e
       TesteClique(G1, k) = falso então
11        Colorir(G1, k)
12     fim-se
13  fim-se
```

Algortimo LasVegas $(G=(N,M),k)$

```
1   ler {G, k}
2   encontrouCor ← falso
3   Inicializar
4   Colorir(G, k)
5   se encontrouCor = verdadeiro então
6      escrever {G é k − colorível}
7   senão
8      escrever {G não é k − colorível}
9   fim-se
```

Os procedimentos $Criar(\beta_{u,v}(G))$ e $Condensar(\gamma_{u,v}(G))$ criam, respectivamente, os grafos G_1 e G_2 por meio das operações estruturais de adição de arestas e condensação de vértices sobre o grafo G. O procedimento *Inicializar* configura as estruturas de dados necessárias e determina um clique focal que será submetido às alterações estruturais. A escolha do clique focal é realizada pela função *Clique*, descrita a seguir, que seleciona de maneira randômica um vértice x de grau máximo e determina um conjunto de vértice que constituem um clique contendo x [6].

Função Clique(V)

```
1   S ← V
2   K ← ∅
3   enquanto S ≠ ∅ faça
4      selecione de modo randômico x ∈ S de
       grau máximo
5      K ← K ∪ {x}
6      S ← S ∪ Γ(x)
7   fim-enquanto
8   Clique ← K
```

O procedimento $Obter(u,v)$ seleciona dois vértices não adjacentes para as operações de adição e condensação de modo que um dos vértices pertence ao clique focal e o outro é aquele que possui a maior quantidade de vizinhos no clique. Se há no grafo mais de um par de vértices que atendem aos critérios de escolha o par com o maior número de vizinhos em comum é o selecionado. As operações de adição de arestas e condensação de vértices podem promover o crescimento do clique, desta forma, a função *TesteClique* é utilizada com o objetivo de verificar se o clique focal tornou-se maior do que k [6].

O procedimento recursivo *Colorir* modifica o valor da variável *encontrouCor* que será

verdadeira ao término da execução caso o grafo G seja k -colorível e falsa, caso contrário [3].

I. Algoritmo de Memória Adaptativa - AMACOL

Estratégias evolucionárias, que abrangem algoritmos genéticos, colônias de formigas, algoritmos de memória adaptativa, entre outros, podem ser definidas como procedimentos iterativos que utilizam uma memória central de onde informações são obtidas durante o processo de pesquisa. Cada iteração ou geração é constituída de duas fases complementares que modificam a memória principal. Na fase cooperação um operador de recombinação é utilizado para criar novas soluções filhas a partir das soluções armazenadas, enquanto que na fase de auto-adaptação as soluções filhas obtidas são modificadas individualmente. As soluções filhas resultantes das fases de cooperação e auto-adaptação são utilizadas para atualizar o conteúdo da memória principal, e, o término do processo de pesquisa pode ser determinado pelo número máximo de iterações do algoritmo, pela qualidade da solução obtida ou por outro critério de parada qualquer, previamente estabelecido. As heurísticas evolucionárias mais eficientes são as que fazem uso de algoritmos híbridos nos quais uma técnica de busca local é aplicada durante a fase de auto-adaptação [7].

A estratégia evolucionária híbrida de maior popularidade é provavelmente o algoritmo genético de busca local, que une a busca local padrão, a qual é utilizada na fase de auto-adaptação, com o algoritmo genético padrão. Nesta abordagem, a memória central de uma busca local genética é constituída de uma população de soluções e o operador de recombinação é um *crossover* que produz uma ou mais soluções filhas utilizando um par de soluções selecionadas da população. O algoritmo de memória adaptativa, que é uma das mais recentes heurísticas evolucionárias híbridas, armazena partes de soluções (ao invés de soluções completas) na memória central, e, enquanto que no algoritmo de busca local genética duas soluções pais são combinadas para produzir uma solução filha, na técnica de memória adaptativa todas as peças de soluções armazenas na memória principal podem contribuir para criação da solução filha [7].

Dado um inteiro fixo k, pode-se definir um problema de otimização denominado k-problema de coloração de grafo (PCG), o qual consiste em determinar uma k -coloração do grafo G que minimiza o número de vértices adjacentes que possuem a mesma cor. Se o valor ótimo do k-PCG for zero afirma-se que G possui uma k-coloração. O número cromático de G pode ser determinado partindo-se de um limite superior para este valor, calculado por meio de um método seqüencial construtivo, para em seguida resolver-se uma série k-PCGs com valores decrescentes de k até que uma k-coloração inválida seja obtida [7].

Um dos eficientes métodos heurísticos utilizados para o k -PCG é uma busca local genética, proposta por Galinier e Hao, a qual será denominada de algoritmo GH. Como a maioria das heurísticas evolucionárias genéticas híbridas para o k -PCG, o algoritmo GH faz uso do algoritmo TABUCOL como método de busca local. A superioridade do algoritmo GH quando comparado a outras heurísticas híbridas evolucionárias é provavelmente decorrente do operador de recombinação, que ao invés de determinar a solução filha colorindo metade dos seus vértices com as cores do primeiro pai e a outra metade com as cores do segundo pai, estabelece que a solução filha seja obtida combinando as classes de cores das soluções pais, as quais correspondem aos conjuntos de vértices que possuem a mesma cor [7].

Galinier, Hertz e Zufferey propuseram um algoritmo de memória adaptativa, chamado AMACOL, para solução do k-PCG que é mais simples e mais flexível do que algoritmo GH, com a vantagem adicional de apresentar resultados tão bons quanto o GH quando submetido à resolução de instâncias clássicas do problema de coloração, disponíveis na literatura.

O operador de recombinação adotado no AMACOL é inspirado em algumas das idéias usadas no algoritmo GH, porém o primeiro difere do segundo em virtude de alguns importantes aspectos. A memória central do AMACOL não armazena soluções obtidas das gerações anteriores mas sim classes de cores extraídas dessas soluções. Além disso, uma solução filha pode ser produzida usando classes de cores originadas de mais do que duas soluções [7].

O algoritmo de memória adaptativa foi inicialmente proposto para solucionar o problema do roteirização de veículos. É um heurística evolucionária híbrida que utiliza uma memória central $\mathcal{M}$ para armazenar partes de soluções. Um esquema elementar para este algoritmo é apresentado a seguir [7].

Algortimo de Memória Adaptativa

1 *inicialize a memória central $\mathcal{M}$ com partes de soluções.*
2 enquanto $critério\ de\ parada = falso$ faça
3 *crie uma solução filha s usando um operador de recombinação*
4 *aplique um operador de busca local em s e verifique se s' representa uma solução válida*
5 *use partes de s' para atualizar $\mathcal{M}$*
6 fim-enquanto

O processo de criar uma solução filha s', aplicar uma busca local em s' e atualizar a memória

principal com a solução obtida é denominado de geração. A fim de criar a solução filha o operador de recombinação seleciona partes de soluções de $\mathcal{M}$. No contexto do k-PCG, $\mathcal{M}$ deve conter soluções estáveis (sem vértices adjacentes com a mesma cor) de forma que o operador escolhe k conjuntos estáveis $S_1, S_2, \ldots, S_k$ a fim de construir a nova solução. A saída do operador é um k-coloração (não necessariamente válida) que será a entrada para o operador de busca local [7].

Conforme já mencionado, a memória $\mathcal{M}$ conterá um conjunto de grafos estáveis e o número de elementos de $\mathcal{M}$ será um múltiplo de k, $|\mathcal{M}| = k \cdot p$, onde p é um parâmetro fornecido ao método [7].

O procedimento CLEAN, descrito a seguir, é capaz de transformar qualquer subconjunto de vértices em um conjunto maximal estável, sendo utilizado no preenchimento e atualização da memória principal $\mathcal{M}$ [7].

Procedimento CLEAN(S)

1 enquanto *S tem pelo menos um vértice conflitante* faça
2 *escolha um vértice x em S que tem o número máximo de vizinhos em S e remova-o de S*
3 fim-enquanto
4 *seja C o conjunto de vértices que não pertencem a S e que não têm vizinhos em S*
5 enquanto $C \neq \emptyset$ faça
6 *escolha um vértice x em C que tem o número mínimo de vizinhos em C, insira x em S e remova x e todos os seus vizinhos de C*
7 fim-enquanto

O primeiro *loop* do procedimento CLEAN transforma S em um conjunto estável, removendo os vértices conflitantes. O segundo *loop* insere vértices em S até que S torne-se um conjunto maximal estável [7].

O procedimento de inicialização da memória principal consiste em construir p k-colorações, que são posteriormente aplicadas ao operador de busca local a fim de serem possivelmente melhoradas. As classes de cores das soluções resultantes são convertidas em conjuntos maximais estáveis por meio do procedimento CLEAN, que são finalmente introduzidos em $\mathcal{M}$ [7].

Procedimento de Iniclização de $\mathcal{M}$

1 $\mathcal{M} \leftarrow \emptyset$
2 para $i = 1$ até p faça
3 *gere uma k-coloração randômica associando de modo aleatório uma cor a cada vértice*
4 *aplique o operador de busca local para no máximo It_{max} iterações e seja s' a k-coloração resultante*
5 *aplique o procedimento CLEAN em cada classe de cor em s' e introduza cada conjunto maximal estável em $\mathcal{M}$*
6 fim-para

A fim de facilitar o entendimento da diferença entre o operador de recombinação do algoritmo GH e o operador de recombinação do algoritmo AMACOL, descreve-se em detalhes a seguir o procedimento executado pela heurística GH.

Operador de Recombinação GH

1 *selecione duas k-colorações $s = (S_1, \ldots, S_k)$ e $s' = (S'_1, \ldots, S'_k)$ da memória principal onde S_i e S'_i correspondem ao conjunto de vértices com a cor i em s e s', respectivamente*
2 */*construa uma k-coloração parcial s'' na qual não há vértices duplicados, mas com possíveis vértices não coloridos*/*
3 para $i = 1$ até k faça
4 se *i é ímpar* então
5 *escolha o conjunto S_j em s que contém o número máximo de vértices e faça $S''_i \leftarrow S_j$*
6 senão
7 *escolha S'_j em s' que contém o número máximo de vértices e faça $S''_i \leftarrow S'_j$*
8 fim-se
9 *remova os vértices de S''_i das soluções pais s e s'*
10 fim-para
11 */* selecione a cor dos vértices não coloridos */*
12 *se existem vértices que não pertencem a $S''_1 \cup \ldots \cup S''_k$ então selecione de modo randômico uma cor para cada vértice (ou seja, insira cada vértice em um conjunto S''_i)*
13 */* a partição $s = (S''_1, \ldots, S''_k)$ é uma k-coloração filha de s e s'*/*

No intuito de ilustrar o procedimento de recombinação acima, suponha um grafo G constituído de 10 vértices $a, b, \ldots, j$ sobre o qual tenta-se aplicar uma 3-coloração. Seja $s = \{\{a,b,c\},\{d,e,f,g\},\{h,i,j\}\}$ e $s' = \{\{c,d,e,g\},\{a,f,i\},\{b,h.j\}\}$. Tem-se que S_2 é a maior classe de cor em s, assim S''_1 é configurado como igual a $S_2 = \{d,e,f,g\}$. Os vértices de S_2 são removidos de s e s', portanto, passamos a ter $s = \{\{a,b,c\},\{h,i,j\}\}$ e $s' = \{\{c\},\{a,i\},\{b,h,j\}\}$. A maior classe em s' será S'_3, logo $S''_2 = S'_3 = \{b,h,j\}$. Esta atribuição leva a $s = \{\{a,c\},\{i\}\}$ e $s' = \{\{c\},\{a,i\}\}$, o que por sua vez resulta em $S''_3 = S_1 = \{a,c\}$. Todos os vértices estão agora coloridos em s'', exceto o vértice i que deverá ser

colocado randomicamente em S''_1, S''_2 ou S''_3. Se o vértice i for associado a S''_3 teremos que a solução filha será a k -coloração $s'' = \{\{d, e, f, g\}, \{b, h, j\}, \{a, c, i\}\}$ [7].

No algoritmo AMACOL a solução filha $(S_1, \ldots, S_k)$ é construída classe por classe. Suponha que as classes de cores $S_i, \ldots, S_{i-1}$ já estão construídas. No intuito de criar a solução S_i o operador de recombinação seleciona de modo randômico uma amostra $\{W_1, \ldots, W_q\}$ de q grafos estáveis de $\mathcal{M}$. O conjunto W_r na amostra com o máximo número de vértices não coloridos é escolhido e a S_i é atribuído o conjunto de vértices não coloridos de W_r. Uma vez que as k classes de cores tenham sido organizadas, vértices não coloridos podem ainda existir, e estes são tratados de modo semelhante ao empregado no algoritmo GH, ou seja, cada vértice não colorido é inserido em um conjunto S_i selecionado de modo randômico. Embora esta estratégia aleatória eventualmente ocasione uma grande quantidade de arestas conflitantes, estes conflitos serão manipulados pelo operador de busca local [7].

Operador de Recombinação AMACOL

1 */*construa uma k-coloração na qual não há vértices duplicados, mas com possíveis vértices não coloridos*/*
2 para $i = 1$ até k faça
3 *selecione uma amostra $W_1, \ldots, W_q$ randômica de q elementos de $\mathcal{M}$*
4 *determine um conjunto W_r com o número máximo de vértices não presentes em $S_1 \cup \ldots \cup S_{i-1}$*
5 *configure $S_i = W_r - \{S_1 \cup \ldots \cup S_{i-1}\}$*
6 fim-para
7 */* selecione a cor dos vértices não coloridos */*
8 *se existem vértices que não pertencem a $S_1 \cup \ldots \cup S_k$ então selecione de modo randômico uma cor para cada vértice (ou seja, insira cada vértice em um conjunto S_i)*
9 */* a partição $s = (S''_1, \ldots, S''_k)$ é uma k-coloração filha de s e s'*/*

A escolha do valor de q torna possível ajustar o nível de aleatoriedade do procedimento. Um valor de q elevado transforma o método numa heurística gulosa, ao passo que pequenos valores para q ocasionam a escolha de conjuntos de $\mathcal{M}$ com uma pequena quantidade de vértices não coloridos. Recomenda-se que o valor escolhido para q seja k [7].

A solução s produzida pelo operador de recombinação serve de entrada para o operador de busca local, que procura melhorar a solução s por no máximo It_{max} iterações. Para realizar a busca local utiliza-se o mesmo algoritmo de busca tabu empregado do algoritmo GH, o qual é um versão aprimorada do algoritmo TABUCOL [7].

No algoritmo TABUCOL o espaço de busca é o conjunto de k-colorações e a função objetivo f a ser minimizada consiste no número total de arestas conflitantes. Uma solução vizinha é obtida por meio da alteração da cor de um dos vértices conflitantes. Quando a cor do vértice x é modificada de i para j, a cor i passa a ser um tabu para x por um certo número de interações (limite tabu) e todas as soluções onde o vértice x tem a cor i serão consideradas soluções tabu. Em cada iteração, TABUCOL determina o melhor vizinho s' da solução s de forma que s' não seja uma solução tabu ou que $f(s') = 0$, ou seja, s' seja uma k-coloração. O limite tabu será igual a $rnd(0{,}9) + 0{,}6 \cdot NVC(s')$, onde $rnd(a, b)$ retorna um inteiro aleatoriamente escolhido entre a e b e $NVC(s')$ corresponde ao número de vértices conflitantes em s' [7].

A solução s' resultante do operador de busca local é utilizada para atualizar a memória central $\mathcal{M}$. Cada classe de cor em s' é convertida em um conjunto maximal estável por meio do procedimento CLEAN. Desta forma, os k -conjuntos maximais estáveis são introduzidos em $\mathcal{M}$ em substituição a k elementos de $\mathcal{M}$, selecionados aleatoriamente [7].

Procedimento para Atualização de $\mathcal{M}$

1 *selecione aleatoriamente k elementos de $\mathcal{M}$ e remova-os de $\mathcal{M}$*
2 *aplique o procedimento CLEAN em cada classe de cor S_i da solução $s' = (S_1, \ldots, S_k)$, calculada pelo operador de busca local*
3 *atualize $\mathcal{M}$ com o conjunto maximal estável resultante*

O algoritmo AMACOL interrompe seu processamento quando uma k-coloração é obtida ou quando o limite do número total de iterações realizados pelo operador de busca local, desde o início da execução do algoritmo, é alcançado. Um terceiro critério de parada está relacionado ao conceito de diversidade das soluções armazenadas na memória. No AMACOL a perda de diversidade é caracterizada pelo fato de que $\mathcal{M}$ contém k grupos de conjuntos maximais bastante semelhantes. Quando este fenômeno ocorre o operador de recombinação sempre irá produzir resultados similares em cada iteração, tornando portanto inútil o esforço empregado no procedimento de pesquisa. A fim de medir a diversidade da memória considera-se $|S \cap S'|$ como a medida de similaridade entre dois elementos S e S' de $\mathcal{M}$ e denota-se por $N(S)$ o conjunto dos $p/2$ elementos de $\mathcal{M}$ que são mais similares a S. Para

cada elemento S de $\mathcal{M}$ calcula-se o seguinte valor, representado por $\sigma(S)$ [7]:

$$\sigma(S) = \frac{\sum_{S' \in N(S)} |S \cap S'|}{|N(S)| \cdot |S|}$$

Segue-se que, se houver pelo menos $|N(S)| = p/2$ conjuntos estáveis em $\mathcal{M}$ que são idênticos a S, então $\sigma(S) = 1$. Por fim, calcula-se a grandeza da diversidade $D(\mathcal{M})$ [7]:

$$D(\mathcal{M}) = 1 - \frac{1}{|\mathcal{M}|} \sum_{S \in \mathcal{M}} \sigma(S)$$

Observe que $D(\mathcal{M}) = 1$ quando, supostamente, $\mathcal{M}$ contiver $|\mathcal{M}| = k \cdot p$ conjuntos disjuntos, enquanto que $D(\mathcal{M}) = 0$ se $\mathcal{M}$ possuir k agrupamentos de p conjuntos idênticos [7].

Conforme já mencionado o método AMACOL utiliza três parâmetros p, q e It_{max}. O parâmetro p é utilizado para fixar o tamanho da memória principal $|\mathcal{M}| = k \cdot p$, enquanto o parâmetro q determina o tamanho da amostra selecionada de $\mathcal{M}$ pelo operador de recombinação. Com base em experimentos anteriores os valores de p e q são fixados, respectivamente, em 10 e k [7].

O esforço do algoritmo AMACOL é medido pela quantidade de iterações realizadas pelo operador de busca local e o parâmetro It_{max}, utilizado por este operador, é um fator determinante da performance do algoritmo. Valores elevados para It_{max} são capazes de preservar melhor a diversidade de $\mathcal{M}$ do que pequenos valores. Desta forma, a aplicação do método à resolução de instâncias complexas do problema da coloração demanda a utilização de valores maiores para It_{max}, pois, do contrário, a diversidade das soluções armazenadas na memória principal pode se exaurir rapidamente, antes que uma k-coloração seja obtida. Ou seja, valores elevados para o parâmetro tendem a aumentar a robustez do algoritmo [7].

No intuito de se determinar o valor adequado para It_{max} adota-se a seguinte estratégia. Fixa-se inicialmente It_{max} como igual ao número de vértices a serem coloridos. Se a medida da diversidade $D(\mathcal{M})$ torna-se menor do que $0{,}1$, multiplica-se It_{max} por $\sqrt{2}$ contanto que 100 gerações do AMACOL sejam realizadas sem ocorra melhoria na solução mais apta s^*, computada até então. Este procedimento é realizado até que uma k-coloração seja obtida ou que o número total de iterações da pesquisa local ($TotalIt_{max} = 250\ milhões$) seja alcançado desde que o processo de busca se iniciou [7].

Algoritmo AMACOL

1 $It_{max} \leftarrow |V|$
2 $It \leftarrow 0$
3 repita
4 *inicialize $\mathcal{M}$; seja s^* a melhor solução encontrada até então*
4 $continue \leftarrow 1$
5 $ultimaChance \leftarrow 0$
6 enquanto $continue = 1$ faça
7 *crie uma solução filha s por meio do operador de recombinação*
8 *aplique a busca local sobre s por, no máximo, It_{max} iterações; seja s' a solução resultante; se $f(s') < f(s^*)$ então $s^* \leftarrow s$*
9 $It \leftarrow It + It_{max}$
10 *atualize $\mathcal{M}$*
11 *determine $D(\mathcal{M})$*
se $D(\mathcal{M}) < 0{,}1$ e $ultimaChance = 0$ então $ultimaChance \leftarrow 1$
se $ultimaChance = 1$ e s^*não sofreu melhora nas últimas 100 gerações então $continue \leftarrow 0$
12 fim-enquanto
13 $It_{max} \leftarrow It_{max} \cdot \sqrt{2}$
14 até $f(s') = 0$ ou $It > It_{max}$

V. APLICAÇÕES DA COLORAÇÃO DE GRAFOS

Existem diversos problemas que podem ser analisados e às vezes solucionados por meio da modelagem da situação descrita no problema através de um grafo e pela definição apropriada de uma coloração no grafo utilizado [4].

Para um determinado grupo de indivíduos, suponha que algumas comissões têm que ser constituídas e que o mesmo indivíduo possa participar de modo concomitante de diversas comissões distintas. Presuma ainda que um horário de reunião deve ser associado a cada comissão e que duas comissões que possuem um membro em comum não podem se reunir simultaneamente. A situação descrita pode ser modelada com o emprego de um grafo G no qual os vértices simbolizam as comissões e dois vértices são adjacentes sempre que as comissões que representam possuam membros em comum. Alguns exemplos específicos da situação apresentada serão referidos a seguir [4].

Considere um grupo constituído por oito empregados de uma empresa, representados por $A = \{a_1, a_2, \ldots, a_8\}$, que têm que se reunir em comissões de três indivíduos com o objetivo de discutir sete temas de relevância para companhia. As comissões formadas para este propósito serão representadas por: $A_1 = \{a_1, a_2, a_3\}$, $A_2 = \{a_2, a_3, a_4\}$, $A_3 = \{a_4, a_5, a_6\}$, $A_4 = \{a_5, a_6, a_7\}$, $A_5 = \{a_1, a_7, a_8\}$, $A_6 = \{a_1, a_4, a_7\}$ e $A_7 = \{a_2, a_6, a_8\}$. Se cada comissão tem que ser reunir durante os intervalos: de 13h às 14h, de 14h às 15h, de 15h às 16h, de 16h às 17h e de 17h às18h, qual o número mínimo de períodos de tempo necessários para que todas as sete comissões possam se encontrar? Duas comissões não podem ser reunir durante o mesmo intervalo de tempo se algum dos empregados faz parte das duas comissões. Defina

um grafo G constituído pelo conjunto de vértices $V(G) = \{A_1, A_2, \dots, A_n\}$ no qual dois vértices A_i e A_j são adjacentes se $A_i \cap A_j \neq \emptyset$, denotando que A_i e A_j devem se reunir em intervalos de tempo distintos [4].

O grafo G é demonstrado na figura 6. De acordo com [4], o número mínimo de períodos de tempo necessários para que as sete comissões possam se reunir será $\chi(G) = 4$ e uma das possíveis formas de distribuição das comissões entre os intervalos de tempo seria: de 13h às 14h, comissões A_1 e A_4; de 14h às 15h, comissões A_2 e A_5; de 15h às 16h, comissão A_3; de 16h às 17h, comissões A_6 e A_7.

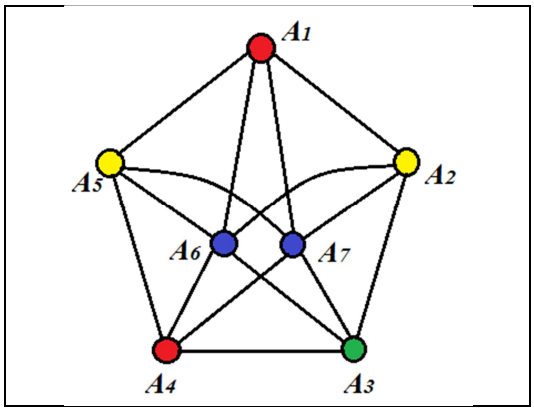

Fig. 6 - Grafo das sete comissões de empregados [4].

Suponha uma comunidade rural na qual existem dez crianças (representadas por $c_1, c_2, \dots, c_{10}$), que habitam em diferentes residências e que necessitam de sessões de fisioterapia semanais. Dez fisioterapeutas de uma comunidade vizinha se ofereceram para visitar algumas dessas crianças durante a semana, levando em consideração a limitação de que nenhuma criança pode ser visitada mais de um vez no mesmo dia. O conjunto de crianças visitadas num determinado dia é referido como um *tour*, sendo estabelecido que o número ideal de crianças a visitar em um *tour* é 4. Os seguintes dez *tours* são então acordados: $T_1 = \{c_1, c_2, c_3, c_4\}$, $T_2 = \{c_3, c_5, c_7, c_9\}$, $T_3 = \{c_1, c_2, c_9, c_{10}\}$, $T_4 = \{c_4, c_6, c_7, c_8\}$, $T_5 = \{c_2, c_5, c_9, c_{10}\}$, $T_6 = \{c_1, c_4, c_6, c_8\}$, $T_7 = \{c_3, c_4, c_8, c_9\}$, $T_8 = \{c_2, c_5, c_7, c_{10}\}$, $T_9 = \{c_5, c_6, c_8, c_{10}\}$ e $T_{10} = \{c_6, c_7, c_8, c_9\}$ [4].

É preferível que todos os *tours* ocorram entre segunda e sexta-feira, porém os voluntários estão dispostos a trabalhar no final de semana, caso necessário. É possível afirmar que todas as crianças podem ser visitadas sem que nenhum fisioterapeuta tenha que estar disponível durante o final de semana [4]?

Considere um grafo G constituído pelo conjunto de vértices $\{T_1, T_2, \dots, T_{10}\}$ no qual T_i é adjacente a $T_j (i \neq j)$ se $T_i \cap T_j \neq \emptyset$, conforme demonstrado por meio da figura 6. O número mínimo de dias necessários para realizar todos os tours é $\chi(G) = 6$. Desta forma, como o número mínimo de dias é superior a 5 conclui-se que alguns dos fisioterapeutas terão que trabalhar nos finais de semana [4].

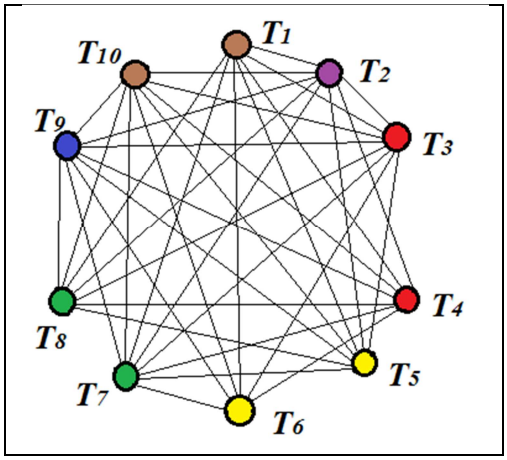

Fig. 7 - Grafo dos dez *tours* de fisioterapeutas [4].

VI. CONCLUSÃO

Este trabalho descreveu o problema de coloração em grafos e apresentou alguns exemplos de técnicas que podem ser utilizadas em sua resolução. Observa-se, por meio dos métodos expostos, que diversas abordagens têm sido desenvolvidas, o que denota o grande esforço empreendido no aperfeiçoamento dos algoritmos até então propostos para abordagem deste problema. Verifica-se ainda a existência de diversas situações passíveis de serem modeladas e solucionadas por meio do problema de coloração, o que por si só, já constitui um fator motivador para que novos métodos sejam desenvolvidos.

REFERENCES

[1] BOAVENTURA NETTO, P. O. Grafos: Teoria, Modelos, Algoritmos. 3 ed. São Paulo: E. Blücher Ltda., 2003.

[2] BRÉLAZ, D. New Methods to Color the Vertices of a Graph, Communications of the ACM, v. 22, n. 4, p. 251-256, 1979.

[3] CAMPELLO, R. E.; MACULAN, N. Algoritmos e Heurísticas: desenvolvimento e avaliação de performance. 1 ed. EDUFF, 1994.

[4] CHARTRAND, G.; ZHANG, P. Chromatic Graph Theory, 1st ed. Chapman and Hall CRC, 2008.

[5] DUTTON R. D.; BRIGHAM R. C. A new graph coloring algorithm. The Computer Journal, v. 24, n. 1, p. 85-86, 1981.

[6] ELLIS J. A.; LEPOLESA, P. M. A Las Vegas Colouring Algorithm. The Computer Journal, v. 32, n. 5, p. 474-476, 1989.
[7] GALINIER, P.; HERTZ, A.; ZUFFEREY, N. An adaptive memory algorithm for the K-colouring problem, Discrete Applied Mathematics, v. 156, n. 2, p. 267–279, 2008.
[8] HERTZ A. Cosine: A new graph coloring algorithm. Operation Research Letters, v. 10, p. 411-145, 1991.
[9] LU, Z.; HAO, J. A memetic algorithm for graph coloring. European Journal of Operation Reasearch, v. 203, p. 241-250, 2009.
[10] VIANA, G. V. R. Meta-Heurísticas e Programação Paralela em Otimização Combinatória. 1 ed. EUFC, 1998.