Algoritmo Genético Construtivo e Geração de Colunas: uma aplicação para Coloração de Grafos

Geraldo Ribeiro Filho
geraldo@lac.inpe.br
UMC
Av Francisco Rodrigues Filho 399
08773-380 Mogi das Cruzes - SP - Brasil

Luiz Antonio Nogueira Lorena
lorena@lac.inpe.br
LAC/INPE
Caixa Postal 515
12.201-970 São José dos Campos – SP - Brazil

Resumo

Neste trabalho apresentamos o uso combinado de Algoritmos Genéticos (AGs) e geração de colunas para resolver de forma aproximada problemas de coloração de grafos. O método proposto é dividido em duas fases. A fase construtiva produz um conjunto inicial de colunas usando um Algoritmo Genético Construtivo (AGC). Cada coluna forma então um conjunto independente de vértices. A segunda fase resolve através de geração de colunas uma formulação de cobertura de conjuntos. As colunas são geradas resolvendo problemas de conjuntos independentes de vértices, com pesos duais. Alguns resultados computacionais são apresentados.

Palavras chave: Geração de colunas, Algoritmo Genético Construtivo, Coloração de Grafos.

Abstract

We present a combined use of Genetic Algorithms (*GAs*) and column generation to approximately solve graph-coloring problems. The proposed method is divided in two phases. The constructive phase builds the initial pool of columns using a Constructive Genetic Algorithm (*CGA*). Each column forms an independent set. The second phase solves by column generation the set covering formulation. The columns are generated solving weighted independent set problems. Some computational experience is given.

Key words: Column generation, Constructive Genetic Algorithm, Graph coloring

1. Introdução

Seja $G = (V,E)$ um grafo não direcionado. Um k-coloração de G é uma partição de V em k subconjuntos C_i, $i = 1,\ldots,k$, tal que nenhum par de vértices adjacentes pertence ao mesmo subconjunto. O problema chamado de Coloração de Grafos é achar uma k-coloração de G com k tão pequeno quanto possível. Este menor valor de k é conhecido como *número cromático* de G. Sabe-se que este problema é NP-hard [Garey e Johnson (1978)], e heurísticas devem ser usadas para grandes grafos. Cada subconjunto de vértices é um conjunto independente, e o problema de coloração poderia ser visto como um problema de agrupamento para formar conjuntos independentes de vértices.

Coloração de grafos é um problema muito estudado [de Werra (1990), Korman, (1979)]. Aplicações aparecem em *scheduling* (como *timetabling*) [de Werra (1985), Leighton (1979)], designação de freqüências [Gamst (1986), Hale (1980)] e alocação de registradores [Briggs et al. (1989)]. O uso de metaheurísticas tem produzido os melhores resultados para uma classe grande de instâncias. Johnson et al. (1991) e Chams et al. (1987) aplicaram *Simulated Annealing*. Costa e Hertz (1996) aplicaram *Ant Colony* . Friden et al. (1989) e Hertz e de Werra (1987) aplicaram *Tabu Search* e Fleurent e Ferland (1994) aplicaram *Algoritmos Genéticos Híbridos* com uma busca local agressiva.

Os Algoritmos Genéticos (AGs) são muito conhecidos e tem aplicações com sucesso em problemas de Otimização Combinatória (OC) [Fleurant e Ferland (1994), Levine (1993), Lorena e Lopes (1996), Lorena e Lopes (1997), Tam (1992), Ulder et al. (1991)]. Um AG clássico baseia-se na evolução controlada de uma população estruturada. As bases de um AG são os operadores de recombinação e de formação e propagação de *esquemas* através das gerações [De Jong (1975), Goldberg (1989), Holland (1975)]. Para obter sucesso em aplicações de AG para resolver problemas de OC algumas características de um AG clássico têm sido adaptadas e redefinidas.

A abordagem com *Algoritmo Genético Construtivo (AGC)* foi proposta recentemente por Lorena e Furtado (1998) e Ribeiro Filho (1997). Um AGC típico não usa apenas soluções completas do problema, mas também partes de uma solução, conhecidas como *esquemas*. O algoritmo trabalha com uma população inicial formada apenas por esquemas. A teoria de esquemas foi por muito tempo o ponto central do AG clássico, mas tem sido menos explorada nos últimos anos. Um esquema simples não é suficiente para representar uma possível solução para o problema de coloração porque alguns vértices ainda não estão coloridos. Novos esquemas ou mesmo *estruturas* (soluções completas) são produzidas através de combinação de esquemas existentes a cada geração. Um processo dupla-adaptação é usado para avaliar os esquemas e os *bons* esquemas são preservados. Um parâmetro de controle da evolução elimina esquemas que não satisfazem um critério de permanência e o melhor esquema encontrado até o momento é mantido salvo. O processo termina quando um limite do número de gerações é alcançado ou quando eventualmente a população fica vazia.

Uma abordagem por geração de colunas para coloração de grafos foi estudada anteriormente por Mehrotra e Trick (1995) para otimização implícita de programação linear a cada nó de um algoritmo *branch & bound*. Os autores mostram que o método resolve rapidamente problemas pequenos e de tamanho moderado.

2. Revisão do AGC

O AGC é proposto para tratar o problema de avaliação de esquemas e de estruturas de uma forma comum. Enquanto nos outros algoritmos evolutivos avaliações de indivíduos são baseadas em uma única função (a função de adaptação), no AGC esse processo se baseia em duas funções, mapeando o espaço de estruturas e esquemas no $\Re_+$.

2.1 Representação

A representação para estruturas e esquemas considera o problema de coloração como um problema de agrupamento, e usa três símbolos. São eles:

→ símbolo que indica os vértices que não estão associados a qualquer agrupamento;
1 → símbolo para indicar o vértice que é uma "*semente*" para formar um agrupamento; e
0 → símbolo que indica os vértices já associados a algum agrupamento.

O número de vértices é exatamente o número de cores sendo usadas, ou agrupamentos sendo formados. A associação de vértices a agrupamentos é feita por uma heurística apropriada.

Esta associação vértice--agrupamento usa uma adaptação de uma heurística conhecida como *Recursive Large First* (*RLF*) (Leighton, 1979) que tem sido considerada boa quando comparada a outras. O processo pode ser melhor entendido usando um exemplo. Suponha que nós estamos procurando uma *3*-coloração para um grafo com dez vértices e a seguinte matriz de adjacências:

0111000000
1000001000
1001010000
1010101100
0001010110
0010100000
0101000001
0001100010
0000100100
0000001000

Consideremos então os seguintes conjuntos:

C_i é o conjunto de vértices no *i*-ésimo agrupamento,
U_i é o conjunto de todo os vértices de esquema adjacentes para qualquer vértice em C_i,
V_{sch} é o conjunto de todos os vértices no esquema, e
V_i é V_{sch} - U_i.

Consideremos também o seguinte esquema: (#,1,0,1,#,0,0,#,1,0).

Onde: 1 = vértice semente,
0 = vértice a ser associado
# = vértice que não participa do processo de associação.

Assim, inicialmente nós temos:

$C_1 = \{2\}$	$C_2 = \{4\}$	$C_3 = \{9\}$
$V_1 = \{3,6,10\}$	$V_2 = \{6,10\}$	$V_3 = \{3,6,7,10\}$
$U_1 = \{7\}$	$U_2 = \{3,7\}$	$U_3 = \{\ \}$

Em seguida, tome o vértice *v* em V_i, (i=1,2,3) com o maior grau em U_i, e associe *v* a C_i. Atualize os conjuntos C_i, V_i, e U_i.

Nós obtemos:

$C_1 = \{2,10\}$	$C_2 = \{4\}$	$C_3 = \{9\}$
$V_1 = \{3,6\}$	$V_2 = \{6\}$	$V_3 = \{3,6,7\}$
$U_1 = \{7\}$	$U_2 = \{3,7\}$	$U_3 = \{\ \}$

Repetindo o processo:

$C_1 = \{2,10\}$	$C_2 = \{4,6\}$	$C_3 = \{9\}$
$V_1 = \{3\}$	$V_2 = \{\ \}$	$V_3 = \{3,7\}$
$U_1 = \{7\}$	$U_2 = \{3,7\}$	$U_3 = \{\ \}$

E o processo continua até que todos os conjuntos V_i estejam vazios.

No final deste exemplo nós teremos os seguintes agrupamentos :

$C_1 = \{2,3,10\}$ $C_2 = \{4,6\}$ $C_3 = \{7,9\}$

2.2.Modelagem no AGC

Fazemos aqui uma descrição geral do processo de modelagem de um problema, para ser resolvido pelo AGC. Isso envolve a definição de um processo de avaliação dupla de estruturas e esquemas, proporcionando a definição de um problema de otimização bi-objetivo, que guiará o processo evolutivo.

Seja C o conjunto de todas as estruturas e esquemas que podem ser gerados pela representação com seqüências de 0s, 1s e #s descrita na seção 2.1., e considere duas funções f e g, definidas como $f : C \to \Re_+$ e $g : C \to \Re_+$, tais que $f(s_i) \leq g(s_i)$, para todo $s_i \in C$. Chamamos a esta dupla avaliação de adaptação de uma estrutura ou esquema s_i de *adaptação-fg*.

O AGC implementa a adaptação-*fg* com dois objetivos:

(*minimização de intervalo*)	Procura $s_i \in C$ com diferença mínima $\{g(s_i) - f(s_i)\}$
(*maximização de g*)	Procura $s_i \in C$ de máximo $g(s_i)$

Considerando a representação de esquemas e estruturas, a avaliação *g* deve ser definida de forma a aumentar com a diminuição dos símbolos #, e portanto as estruturas apresentarão avaliação *g* maiores que os esquemas.

Para atingir estes propósitos, um problema a ser resolvido usando AGC é modelado como os seguintes *Problemas de Otimização de Duplo-Critério* (POD):

$$\begin{array}{ll} Min & \{g(s_i) - f(s_i)\} \\ Max & g(s_i) \\ \text{sujeito a} & g(s_i) \geq f(s_i) \\ & s_i \in C \end{array}$$

As funções *f* e *g* devem ser definidas adequadamente para representar os objetivos de otimização do problema tratado.

2.3. Adaptação-*fg*

Para o problema de coloração, as funções *f* e *g* usadas são:

$$g(s_i) = \sum_{p=1}^{k} \left[\left(\left| C_{ip} \right| - 1 \right) . \left| C_{ip} \right| \right] / 2$$

$$f(s_i) = g(s_i) - \sum_{p=1}^{k} \left| E\left(C_{ip}\right) \right|$$

e

$$g_{\max} = mult.k. \left[\frac{\left(\lfloor n/k \rfloor - 1 \right) . \lfloor n/k \rfloor}{2} \right]$$

Onde *k* é um número de cores pré-fixado, C_{ip} é o conjunto de vértices que recebe a cor *p* no esquema s_i (a notação $\left| C_{ip} \right|$ representa o número de vértices do conjunto C_{ip}), e $E\left(C_{ip}\right)$ é o conjunto de arestas com ambos o vértices terminais em C_{ip}. A expressão $\left(\left| C_{ip} \right| - 1 \right) . \left| C_{ip} \right| / 2$ dá o número de arestas de um grafo completo com $\left| C_{ip} \right|$ vértices.

A função $g(s_i)$ pode ser interpretada como o número total de arestas se *k* grafos completos de tamanhos $\left| C_{ip} \right|$ são considerados. A função $f(s_i)$ diminui deste número o número de arestas realmente ligando vértices nos conjuntos C_{ip}. Quando $f(s_i) = g(s_i)$ os *k* conjuntos C_{ip} são independentes.

Para obter o limite superior $g_{\max}$, primeiro divide-se o número de vértices *n* em *k* conjuntos com aproximadamente o mesmo número de elementos (a expressão $\lfloor n / k \rfloor$ dá o maior inteiro menor

que *n/k*), então o mesmo procedimento usado para $g(s_i)$ é aplicado, onde o inteiro positivo *mult* é usado para garantir que $g_{\max} > \underset{s_i \in P_a}{Max}\, g(s_i)$.

2.4. O processo de evolução

O processo de evolução no AGC é conduzido para atingir ambos os objetivos (*minimização de intervalo g-f* e *maximação de g*) do POD. No início do processo, os seguintes dois *valores esperados* são dados: um número real não-negativo $g_{\max} > \underset{s_i \in P_a}{Max}\, g(s_i)$, que é um limite superior para $g(s_i)$, para cada $s_i \in C$, e o comprimento de intervalo $d\ g_{\max}$, obtido de $g_{\max}$ usando um número real $0 < d \leq 1$.

O processo de evolução é então conduzido considerando um limiar adaptativo de rejeição, que contempla ambos os objetivos do POD. Dado um parâmetro $a \geq 0$, a expressão,

$$g(s_i) - f(s_i) \geq d\, g_{\max} - a \cdot d[g_{\max} - g(s_k)] \qquad (2.4.1)$$

apresenta uma condição para rejeição de um esquema ou estrutura s_i, considerando a população atual.

O lado direito da desigualdade (2.4.1) é o limiar, composto do valor esperado para o intervalo de minimização d $g_{\max}$ e da medida $[g_{\max} - g(s_k)]$, que mostra a diferença entre as avaliações $g(s_i)$ e $g_{\max}$. Para $a = 0$, (2.4.1) é equivalente a comparar comprimento do intervalo obtido para s_i e comprimento esperado $d\ g_{\max}$. Esquemas ou estruturas são descartadas se a expressão (2.4.1) é satisfeita. Quando $a > 0$, esquemas têm maior possibilidade de serem descartados do que estruturas, pois estruturas apresentam, em geral, diferenças menores $[g_{\max} - g(s_k)]$ do que os esquemas.

O parâmetro a é relacionado ao tempo no processo de evolução. Considerando que os bons esquemas precisam ser preservados para recombinação, o *parâmetro de evolução a* começa com 0 e aumenta lentamente, em intervalos de tempo pequenos, de geração para geração. A população no momento de evolução a, denotada por P_a, é dinâmica em tamanho de acordo com o valor do parâmetro a, e pode ser esvaziada durante o processo.

O parâmetro a é isolado na expressão (2.4.1), produzindo a expressão seguinte que corresponde a um *rank* para s_i:

$$a \geq \frac{dg_{\max} - [g(s_i) - f(s_i)]}{d[g_{\max} - g(s_i)]} = d(s_i)$$

No momento em que são criados, estruturas e/ou esquemas recebem seus valores correspondentes $d(s_i)$. O *rank* de cada esquema ou estrutura é comparado com o valor atual do parâmetro de evolução a. No instante em que uma estrutura ou esquema é criado, é então possível ter uma previsão de sua sobrevida. Quanto maior o valor de $d(s_i)$, melhor é a estrutura ou esquema para o POD, e eles terão mais tempo de sobrevivência para recombinação.

2.5. Seleção e recombinação

A população é mantida em uma ordem não-decrescente de acordo com uma chave dada por

$$\Delta(s_i) = (1 + d_i)/(n - n_{\#})$$

onde $d_i = [g(s_i) - f(s_i)]/g(s_i)$, e $n_{\#}$ é o número de símbolos # em s_i. Esquemas com $n_{\#}$ pequeno e/ou que apresentem pequeno d_i são melhores e aparecem nas primeiras posições.

Na seleção, um primeiro esquema é escolhido entre as *n* primeiras posições na população (*esquema base*) e um segundo esquema na população inteira (*esquema guia*).

Antes da recombinação, o primeiro esquema é complementado para gerar uma estrutura que representa uma possível solução, i.e. todos #' s são substituídos por 0's. Esta estrutura completa sofre mutação e é comparada à melhor solução encontrada até o momento (que é mantida salva ao longo do processo).

A recombinação usa informações de ambos esquemas selecionados, mas conserva o número de símbolos 1s (número de cores) no novo esquema gerado .

Recombinação

if $s_{base}(j) = s_{guide}(j)$ *então* $s_{new}(j) \neg s_{base}(j)$
if $s_{guide}(j) = \#$ *então* $s_{new}(j) \neg s_{base}(j)$
if $s_{base}(j) =$ *# ou 0 e* $s_{guide}(j) = 1$ *então*
    $s_{new}(j) \neg 1$ *e* $s_{new}(i) \neg 0$ *para algum* $s_{new}(i) = 1$
if $s_{base}(j) = 1$ *e* $s_{guide}(j) = 0$ *então*
    $s_{new}(j) \neg 0$ *e* $s_{new}(i) \neg 1$ *para algum* $s_{new}(i) = 0$

A cada geração, exatamente *n* novos esquemas são criados através de recombinação. Se um novo esquema for gerado, então ele é inserido na população; caso seja gerada uma estrutura representando uma possível solução, ela sofre mutação e é comparada à melhor solução encontrada até o momento. O pseudo-código seguinte descreve o processo de mutação:

Processo de mutação

1: Para cada agrupamento
    Mover a semente para o vértice com o maior grau no agrupamento
    Re-associar os vértices usando a abordagem RLF
    Contar conflitos e salvar o melhor movimento neste *loop*
2: Se o melhor encontrado no *loop* acima é melhor que a solução original
    Substituir a original por este melhor e voltar ao passo 1
    senão
    Parar.

2.6. O algoritmo AGC

O AGC pode ser resumido no seguinte pseudo-código:

AGC

Dados g_{max} and d ;
$\alpha := 0$;
$\varepsilon := 0.05$; *{ intervalo de tempo }*
Inicializar P_α ; *{ população inicial }*
Avaliar P_α ; *{ avaliação-fg }*
Para todos $s_i \hat{I} P_a$ *computar* $d(s_i)$ *{ rank }*
Fim Para
Enquanto (not condição de parada) **faça**
    Para todo $s_i \hat{I} P_a$ satisfazendo $\alpha < d(s_i)$ **faça** *{ teste de evolução }*
        $\alpha := \alpha + \varepsilon$;
        Selecionar P_α from $P_{\alpha-\varepsilon}$; *{ operador de reprodução }*
        Recombinar P_α ; *{ operadores de recombinação }*
        Avaliar P_α ; *{ avaliação-fg }*
    Fim Para
    Para todo novo $s_i \hat{I} P_a$ *computar* $d(s_i)$ *{ rank }*
    Fim Para
Fim Enquanto

3. A geração de colunas

O processo de geração de colunas foi proposto por Mehrotra e Trick (1995). O problema mestre (PM) usado é:

$$\text{Min} \quad \sum_{j \in J} x_j$$

$$\text{Sujeito a} \quad \sum_{j \in J} a_{ij} x_j \geq 1, \; i = 1,...,|V|$$

$$x_{ij} \in \{0,1\}, \; i = 1,...,|V|; j \in J.$$

Onde $a_{ij} \in \{0,1\}$ e J é o conjunto de todos os conjuntos independentes maximais de G. PM é um problema de cobertura de conjuntos com um grande número de colunas (geralmente) desconhecidas, que quando necessário são geradas resolvendo-se o seguinte Problema de Máximo Conjunto Independente com Pesos (PMCIP):

$$\text{Max} \quad \sum_{i \in V} l_i z_i$$

$$\text{Sujeito a} \quad z_i + z_j \leq 1, \forall (i, j) \in E$$

$$z_i \in \{0,1\}, \forall i \in V.$$

Onde l_i são variáveis duais para cada restrição no PM. Um conjunto inicial de colunas deve ser dado para formar o PM inicial, e colunas são escolhidas colunas para entrar no PM se elas devolvem limites maiores que 1 quando resolvendo o PMCIP.

O AGC descrito na seção 2 foi usado aqui para formar o conjunto inicial de colunas para o PM. Inicialmente é escolhido um número cores. Se o AGC encontrar uma coloração, este número é reduzido, até que nenhuma melhoria seja encontrada. Um número de conjuntos independentes encontrados durante a última aplicação do AGC é armazenado para compor o primeiro conjunto de colunas. A solução ótima do PM pode dar um limite inferior para o problema de coloração, e as variáveis duais são salvas para serem usadas no PMCIP.

Na seqüência, o mesmo AGC é usado para resolver aproximadamente o PMCIP, escolhendo o último número de cores usado diminuído de um, e armazenando os conjuntos independentes encontrados. Estes novos conjuntos independentes são juntados ao conjunto prévio de colunas e o PM é resolvido novamente. O processo continua até que nenhuma coluna seja encontrada para ser adicionada ao PM.

Um limite inferior para o problema de coloração é obtido a cada iteração aplicando o limite de Farley (Farley, (1990)), dado por $v(MP)/v(WMIP)$, onde v (.) é o valor ótimo do problema correspondente. Estes valores mudam a cada iteração no processo.

4. Testes Computacionais

Foram feitos testes computacionais com várias instâncias tomadas de grupos diferentes:

- *grafos de livros* (Anna, David, Huck e Jean - cada vértice representa uma personagem e são conectados dois vértices se as personagens correspondentes encontram uma a outra na trama do livro);
- *grafos de jogos* (Games120 - cada vértice representa um time e são conectados dois vértices se eles jogarem um com o outro durante o campeonato);
- *grafos de distância* (Miles250, Miles500 e Miles750 - são conectados os vértices que representam cidades, se estas são próximas o suficiente);
- *grafos de registradores* (Musol_1, Musol_2, Zeroin_1 e Zeroin_2 - baseados em alocação de registradores para variáveis em código de programa);
- *grafos de Mycielski* (Myciel5, Myciel6 e Myciel7 - grafos baseados na transformação de Mycielski); e
- *grafos de rainhas* (Queen55, 66, 77, 88 e 99 - um grafo com N^2 vértices, cada um correspondendo a um quadrado em um tabuleiro de xadrez NxN, e dois vértices são conectados se os quadrados correspondentes estão na mesma linha, coluna ou diagonal).

A Tabela 1 a seguir mostra alguns resultados computacionais. A tabela contém números médios de vértices, arestas e de conflitos para cada grupo de instâncias. Todas as experiências foram feitas com três execuções para cada instância, todas procurando o número ótimo de cores.

A qualidade dos resultados pode ser vista facilmente, especialmente para os grafos de rainhas, considerados difíceis. O problema Queen99 foi o único para o qual o número cromático não foi alcançado pelo AGC.

Então nós prosseguimos com testes de geração de colunas usando a instância Queen99. A Tabela 2 mostra os resultados.

Grupo	*Instâncias*	*Vértices*	*Arestas*	*RLF CGA*
Books	4	94.7	363.5	0
Games	1	120	638	0
Miles	3	128	1223.3	0
Register	4	204.8	3848.6	0
Mycielski	3	111	1117	0
Queen	5	51	753.2	0.5

Tabela 1: resultados computacionais do AGC

Iteração	*Número de cores*	*Melhor número de conflitos com AGC*	*Limite do PM*	*Limite de Farley*	*Tempo (seg.)*
0	10	2	9.226	8.359	295
1	9	10	9.059	8.155	283
2	8	25	9.007	8.542	246
3	7	47	9.000	-	177
4	6	75	-	-	121

Tabela 2: Processo de geração de colunas para Queen99

Os tempos na Tabela 2 correspondem à aplicação do AGC. O CPLEX 6.5 resolve o PM muito rapidamente, e os seus tempos não são informados. A melhor solução encontrada pelo AGC foi 11 cores. Considerando o melhor PM e o limite de Farley, pode ser conjeturado que o melhor número de cores para Queen99 é 9, 10 ou 11 (atualmente na literatura o melhor número é 10). Quando o número de cores diminui, aumenta o número de conflitos em soluções do AGC, mas as novas colunas melhoram os limites do PM. O limite de Farley tem um comportamento oscilante devido ao fato de que o PMCIP não foi resolvido de forma exata. Foi usada uma máquina SUN-ULTRA30 e os parâmetros do AGC foram: limite de gerações = 20, incremento de a = 0.01, d = 0.15 e *mult* = 2.0.

Para complementar este trabalho, devem ser feitos outros testes com grafos maiores e também os parâmetros de CGA devem ser analisados.

Agradecimentos: O segundo autor agradece ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (proc. 350034/91-5, 520844/96-3, 680082/95-6) e Fundação para o Amparo a Pesquisa no Estado S. o Paulo - FAPESP (proc. 95/9522-0 e 96/04585-6).

Referências

Briggs, P.; Cooper, K.; Kennedy, K. and Torczon, L. (1989) Coloring heuristics for register allocation. In *ASCM Conference on Program Language Design and Implementation*, pp.275-284.

Chams, M.; Hertz A.and Werra, D. (1987) Some experiments with simulated annealing for coloring graphs. *European Journal of Operations Research* 32:260-266.

Costa, D. and .Hertz, A. (1996) Ants can color graphs. *Journal of the Operational Society* 47:1-11.

De Jong, K. A. (1975) Analysis of the behaviour of a class of genetic adaptive systems. *Ph.D. Dissertation - Department of Computer and Communication Sciences* - University of Michigan , Ann Arbor.

de Werra, D. (1985) An introduction to timetabling. *European Journal of Operational Research* 19(2):151-162.

de Werra, D (1990) Heuristics for Graph Coloring. In *Computational Graph Theory*, ed. Tinhofer, G. ; Mayr, E. ; Noltemeier, H. and Syslo, M., Springer-Verlag , Berlin, pp.191-208.

Farley, A A (1990) A note on bounding a class of linear programming problems, including cutting stock problems. *Operations \research* 38(5): 922-923.

Fleurent, C. and Ferland, J. A. (1994) Genetic and Hybrid Algorithm for Graph Coloring, *Technical report* - Université de Montréal.

Friden, C.; Hertz, A. and de Werra, D. (1989) STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing* 42:35-44, 1989.

Gamst, A. (1986) Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology* 35 (1):8-14.

Garey, M. R. and Johnson, D. S. (1978) *Computers and Intractability: a Guide to the Theory of NP-Completeness.* San Francisco, W. H. Freemann.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, New York.

Hale, W. K. (1980) Frequency assignment: Theory and applications. *Proceedings of the IEEE* 68(12):1497-1514.

Hertz A. and Werra, D. (1987) Using tabu search techniques for graph coloring. *Computing* 39:345-351.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*. The University of Michigan Press. , Ann Arbor.

Johnson, D. S.; Aragon, C. R. ; McGeoch, L. A. and Schevon, C. (1991) Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research* 39(3):378-406.

Korman, S. M. (1979) The graph-coloring problem. In Christofides, N. ; Mingozzi, A.; Toth, P. and Sandi, C. editors, *Combinatorial Optimization*: 211-235. JohnWiley & Sons, Inc., New York.

Leighton, F. T. (1979) A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards* 84: 489-506.

Levine, D. M. (1993) A genetic algorithm for the set partitioning problem. In *Proceedings of the 5th International Conference on Genetic Algorithms* , pp. 481-487.

Lorena, L.A.N. and Furtado, J.C. *Constructive genetic algorithm for clustering problems.* Submitted for publication – Evolutionary Computation. Presented at the Optimization 98 congress - Coimbra, Portugal - July 1998. Available from http://www.lac.inpe.br/~lorena/cga/cga_clus.PDF

Lorena, L.A.N. and Lopes, L.S. (1996) Computational Experiments with Genetic Algorithms Applied to Set Covering Problems. *Pesquisa Operacional* 16: 41-53.

Lorena, L.A.N. and Lopes, L.S. (1997) Genetic Algorithms Applied to Computationally Difficult Set Covering Problems. *Journal of the Operational Research Society* 48, 440-445.

Mehrotra, A and Trick, M. A (1995) A column generation approach for graph coloring. Available from http://mat.gsia.cmu.edu/color.ps

Ribeiro Filho, G. (1996) Uma heurística Construtiva para Coloração de Grafos. *Master thesis*, INPE.

Tam, K. Y. (1992) Genetic algorithm, function optimization and facility layout design. *European Journal of Operational Research* 63:322-240.

Ulder, N. L. J. ; Aarts, E. H. L. ; Bandelt, H.-J. ; vanLaarhoven, P. J.M.and Pesch, E (1991) Genetic local search algorithms for the traveling salesman problem. In H.-P. Schwafel and R. Manner, editors, Springer-Verlag, *Proceedings 1st International Workshop on Parallel Problem Solving from Nature, Lecture Notes in Computer Science* 496: 109-116.