UMA HEURÍSTICA BASEADA EM ALGORITMOS GENÉTICOS

APLICADA AO PROBLEMA DE COBERTURA DE CONJUNTOS

Luciana de Sousa Lopes

Dissertação de Mestrado em Computação Aplicada, orientada pelo Dr. Luiz Antônio Nogueira Lorena

INPE
São José dos Campos
Março de 1995

RESUMO

Uma aplicação de Algoritmos Genéticos tem sido encontrar soluções heurísticas para problemas difíceis da otimização combinatória. Aplicações bem sucedidas de algoritmos genéticos a esses problemas requerem uma codificação apropriada do espaço de soluções, uma função de adaptação específica ao problema, o estabelecimento de um número de parâmetros e, possivelmente, operadores genéticos especializados. Esse trabalho descreve um estudo da aplicação de algoritmos genéticos a um importante problema da otimização combinatória - o Problema de Cobertura de Conjuntos. Ao algoritmo genético clássico foi incorporado um operador de viabilidade, de forma a restringir a busca genética ao espaço das soluções viáveis. A busca genética foi conduzida a partir de uma população inicial obtida através de uma heurística de busca local. Um operador de mutação adaptativo foi implementado com o objetivo de evitar a convergência prematura do algoritmo. Os resultados computacionais demonstram que a heurística baseada em algoritmos genéticos é capaz de produzir boas soluções para o problema de cobertura de conjuntos.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

CAPÍTULO 1

Introdução

Uma grande classe de problemas interessantes de otimização não possuem algoritmos conhecidos que sejam rápidos e eficientes na sua solução. Em algumas aplicações, uma solução próxima do ótimo é aceitável se puder ser computada com razoável rapidez. Uma possível abordagem na busca de tais soluções é utilizar um algorítmo probabilístico que, dado um tempo suficiente, pode encontrar soluções aceitáveis.

Algoritmos genéticos são uma classe de algoritmos probabilísticos que, a partir de uma população inicial de soluções candidatas, "evoluem" em direção a melhores soluções aplicando operadores modelados em processos genéticos que ocorrem na natureza [28]. Algoritmos genéticos diferem dos algoritmos de busca mais tradicionais (por exemplo, descida na direção do gradiente, "hill-climbing", "simulated annealing", etc.) no sentido de que sua busca é conduzida usando a informação de uma população de estruturas, em vez de uma única estrutura. A motivação para essa abordagem é que, considerando várias estruturas como soluções potenciais, o risco da busca chegar a um mínimo local é bastante reduzido.

A utilização de algorítmos genéticos na solução de problemas de otimização tem mostrado que estes, embora não necessariamente encontrem soluções ótimas para o problema, são capazes de encontrar boas soluções para problemas que são resistentes a outras técnicas conhecidas [36,46,51].

O objetivo desse trabalho é investigar o desempenho de algoritmos genéticos na solução de um problema importante da otimização combinatória, o Problema de Cobertura de Conjuntos (PCC). Esse é um problema NP-completo [23] frequentemente

encontrado em aplicações tais como na alocação de facilidades [47] e escalonamento de tripulações [24,35].

Nesse trabalho são descritos os resultados obtidos na solução desses problemas, pela aplicação de uma heurística baseada em algoritmos genéticos. Ao algoritmo genético básico foi incorporada uma heurística de busca local para inicialização da população, um operador de viabilidade e um operador de mutação adaptativo.

O experimento computacional foi realizado utilizando dois grupos de problemas. O primeiro grupo é constituído por um conjunto de problemas de grande porte obtidos dos trabalhos de Balas e Ho [3] e de Beasley [7], os quais constituem um grupo rico de problemas com soluções ótimas conhecidas. No segundo grupo de problemas está incluída uma classe de problemas de cobertura de conjuntos computacionalmente difíceis que surgem em sistemas de triplas de Steiner. Fulkerson et al. [22] sugerem que esses são bons problemas para o teste de novos algoritmos para programação inteira e cobertura de conjuntos.

No próximo capítulo fazemos uma revisão bibliográfica do Problema de Cobertura de Conjuntos e a apresentação de uma classe de problemas computacionalmente difíceis. O Capítulo 3 descreve os métodos mais conhecidos para solução do PCC, com destaque para algumas abordagens necessárias para situar o presente trabalho. O Capítulo 4 introduz os conceitos básicos de algoritmos genéticos e sua fundamentação teórica, além de apresentar a versão de um algoritmo genético clássico. O Capítulo 5 descreve as formulações, estruturas de dados e técnicas em busca genética utilizadas na heurística proposta nesse trabalho. No capítulo 6 são descritos e analisados os resultados dos experimentos realizados com o algoritmo proposto. O último capítulo apresenta uma discussão a respeito das limitações do algoritmo proposto, extensões e potencial para trabalho futuro.

CAPÍTULO 2

O Problema de Cobertura de Conjuntos

2.1. Formulação do problema e aplicações

Considere que em uma malha urbana estejam localizados um conjunto de pontos geradores de demanda e um conjunto de pontos candidatos para a localização de facilidades. Nos referiremos ao primeiro conjunto simplesmente como "conjunto de demanda" do problema e aos pontos do segundo conjunto como "conjunto de facilidades". O problema consiste em encontrar o menor número de pontos no conjunto de facilidades necessário para cobrir o conjunto de pontos de demanda, sendo que cada ponto no conjunto de demanda deve estar localizado a uma distância máxima de um ponto no conjunto de facilidades. Por exemplo, considere os conjuntos de pontos de demanda $X_{m=}$ {E...H} e pontos de facilidades $Y_{n=}$ {W...Z}. Suponhamos que cada ponto de demanda deva estar localizado a menos de 20 minutos de distância em relação a uma das facilidades. A matriz de distâncias mínimas (em minutos) entre os pontos da rede é dada a seguir:

$$[d(i,j)] = \begin{array}{c} \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccc} W & X & Y & Z \end{array} \\ \begin{bmatrix} 3 & 25 & 13 & 22 \\ 25 & 21 & 12 & 24 \\ 12 & 22 & 27 & 8 \\ 26 & 4 & 14 & 31 \end{bmatrix} \end{array}$$

A matriz $[d(i,j)]$ pode ser diretamente convertida em uma matriz zero-um $[A(i,j)]$, fazendo:

$$A(i,j) = \begin{cases} 1, \text{ se } d(i,j) \leq 20 \\ 0, \text{ caso contrario} \end{cases}$$

$$[A(i,j)] = \begin{array}{c} \\ E \\ F \\ G \\ H \end{array} \begin{array}{c} \begin{array}{cccc} W & X & Y & Z \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

A matriz A é denominada *matriz de incidência*. Dizemos que um ponto $y_j \in Y_n$ "cobre" ("não cobre") um ponto $x_i \in X_m$, se o ponto y_j satisfaz (não satisfaz) o limite de distância com respeito ao ponto x_i, ou seja , $d(i,j) \leq 20$. O problema de cobertura de conjuntos consiste então em encontrar o número mínimo k^*, de pontos do conjunto Y_n tal que todos os pontos em X_m sejam cobertos.

O Problema de Cobertura de Conjuntos (PCC) pode ser definido formalmente como: Seja $I = \{1,...,m\}$ o conjunto de índices de linha, $J = \{1,...,n\}$ o conjunto de índices de coluna e $P = (P_1,...P_n\}$ um conjunto de subconjuntos de I, onde $P_j = \{i \hat{I} \ I \mid A_{ij} = 1\}, j \in J$. P_j é o conjunto de índices de linha que possuem um 1 na j-ésima coluna. $\mid P_j \mid$ é a cardinalidade de P_j. Um conjunto $J^* \subseteq J$ é chamado uma *cobertura* se

$$\bigcup_{j \in J^*} P_j = I \qquad (1)$$

O problema consiste em encontrar a cobertura com custo mínimo, ou seja, um conjunto mínimo de subconjuntos P_j que contenha todos os índices de linha.

Outra notação muito comum na literatura, define matematematicamente o PCC como:

$$\text{Min} \quad cx \tag{2}$$

$$\text{sujeito a} \quad Ax \geq e \tag{3}$$

$$x \in \{0,1\}^n \tag{4}$$

onde $A(i,j)$ é uma matriz m x n zero-um, x é um vetor coluna de variáveis x_j (j=1,...,n), com x_j = 1 se a coluna j da matriz A pertence à solução e x_j = 0, caso contrário; c é um vetor linha de dimensão n associado aos custos de cada variável x_j; e é um vetor coluna de uns com dimensão m.

Note que os custos c_j associados às colunas da matriz de incidência não são necessariamente iguais. Quando um número positivo (tipicamente um) é associado a todas as colunas de A, temos um caso especial do PCC, o problema denominado como Problema de Cobertura de Conjuntos de Mínima Cardinalidade (PCCMC). Esse problema é, na prática, usualmente mais difícil de resolver que o PCC com custos diferenciados [49].

Outro caso particular do PCC, o Problema de Partição de Conjuntos (PPC), é obtido substituindo (3) por

$$Ax = e \tag{3a}$$

O PCC é um problema NP-completo clássico [23]. Algumas aplicações do PCC incluem problemas de localização de facilidades p/ atendimento emergencial [42,47,51], balanceamento de linhas de montagem [43] e recuperação de informações [14].

Aplicações do modelo de partição de conjuntos surgem em problemas de escalonamento de unidades de transporte [5,20] e de tripulações [39,40]. Outras aplicações incluem o projeto de circuitos digitais [41] e zoneamento político [25].

2.2 - Reduções do PCC

Frequentemente é possível eliminar, a priori, determinadas linhas e colunas da matriz *A*. São muitos os testes de redução conhecidos na literatura para eliminação de linhas e colunas do PCC original. Descrições detalhadas desses métodos podem ser encontradas em [6] e [26]. Alguns dos métodos comumente utilizados são os seguintes:

(1) Não viabilidade

Se pelo menos uma linha i da matriz A, $i = 1,...m$, possui somente elementos nulos, então não existe solução viável, já que a i-ésima restrição não pode ser satisfeita.

(2) Inclusão de coluna

Se qualquer linha i de A possui apenas um elemento não nulo, diga-se na coluna j^*, então a coluna j^* deve estar em toda solução viável e a linha i pode ser excluída. Automaticamente, todas as linhas t cobertas por essa coluna, ou seja, $t \in P_{j^*}$, também podem ser excluídas.

(3) Dominância de linhas

Se o conjunto das colunas que cobrem um linha k está contido no conjunto de colunas que cobrem uma linha i, $i \neq k$, então a linha k pode ser removida do problema, pois qualquer coluna que cubra a linha i cobrirá automaticamente a linha k.

(4) Coluna nula

Se uma coluna j da matriz A é um vetor nulo, então j pode ser excluída do problema e as variáveis x_j devem ser fixadas em zero em toda solução viável.

(5) Dominância de colunas

Se uma coluna j da matriz A é tal que todas as linhas i, $i \in P_j$, podem ser cobertas por outras colunas com um custo menor que c_j, então j pode ser removida do problema.

2.3 - Um problema de cobertura de conjuntos computacionalmente difícil

Fulkerson et al. [22] destacam a existência de muitas abordagens na literatura para algoritmos que resolvem problemas de cobertura de conjuntos com um número grande número de variáveis. Os problemas que surgem no cálculo da dimensão-1 das matrizes de incidência de sistemas de triplas de Steiner provêm um conjunto de problemas compactos facilmente gerados e que propiciam um desafio para novas idéias e técnicas em programação inteira .

A dimensão-β, de uma matriz zero-um *A*, é o número mínimo de colunas que pode ser selecionado de *A*, tal que todas as somas de elementos das linhas da sub-matriz resultante seja pelo menos β. As matrizes de incidência obtidas de sistemas de triplas de Steiner possuem as seguintes características :

(1) possuem precisamente 3 uns em cada linha. Dizemos que $\{i,j,k\}$ é uma tripla de *A* se existe uma linha *r* de *A* tal que $a_{ri} = a_{rj} = a_{rk} = 1$.

(2) para cada par de colunas j_1 e j_2, existe exatamente uma linha *i* tal que $a_{ij1} = a_{ij2} = 1$.

(3) tais matrizes existem se e somente se $n \geq 3$ e mod(n,6) = 3, sendo $m = \frac{1}{6} n (n\text{-}1)$.

(4) cada coluna de *A* contém exatamente $\frac{1}{2} (n - 1)$ entradas não nulas.

Hall [31] discute essa estrutura em detalhe e mostra uma técnica padrão para gerar recursivamente sistemas de Steiner para os quais $n = 3^k$ (k = 1,2,3,...). A_3 é a matriz 1 X 3 de uns. A_{3n} é obtida de *A* da seguinte forma: as colunas de A_{3n} são indexadas $\{(i,j): 1 \leq i \leq n, 1 \leq j \leq 3\}$. O conjunto $\{(i,r),(j,s),(k,t)\}$ é uma tripla de A_{3n} se e somente se uma das seguintes condições é verdadeira:

(i) $r = s = t$ e $\{i,j,k\}$ é uma tripla de A_n, ou
(ii) $i = j = k$ e $\{r,s,t\} = \{1,2,3\}$ ou
(iii) $\{i,j,k\}$ é uma tripla de A_n e $\{r,s,t\} = \{1,2,3\}$

Para auxiliar a compreensão do procedimento de construção pode ser dada a seguinte descrição informal: As linhas de A_{3n} podem ser divididas em três partes correspondentes às condições (i)-(iii). As primeiras duas partes se assemelham a

$$\begin{bmatrix} A_n & 0 & 0 \\ 0 & A_n & 0 \\ 0 & 0 & A_n \\ I & I & I \end{bmatrix}$$

onde I é a matriz identidade de dimensão apropriada. A terceira parte consiste de 3! blocos correspondentes às permutações de {1,2,3}; para cada permutação Π, as triplas do bloco correspondente são $\{(i, \Pi(1)),(j, \Pi(2)),(k, \Pi(3))\}$, onde (i,j,k) cobrem todas as triplas de A_n. Como exemplo, a figura abaixo mostra A_9, de dimensões 9 x 12.

$$A_9 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Fulkerson et. al [22] reportam um método para construção de A_{15} e A_{45}. A matriz de incidência A_{15} possui as seguintes características:

$$A_{15} = \begin{bmatrix} Z & E & 0 \\ 0 & Z & E \\ E & 0 & Z \\ I & I & I \end{bmatrix}$$

onde

$$Z = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad E = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

e I é a matriz identidade 5 x 5. A matriz A_{45}, de dimensões 330 x 45, é construída da seguinte forma:

$$A_{45} = \begin{bmatrix} A_{15} & 0 & 0 \\ 0 & A_{15} & 0 \\ 0 & 0 & A_{15} \\ C^1 & I & P^1 \\ \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots \\ C^{15} & I & P^{15} \end{bmatrix}$$

onde I é a matriz identidade 15 x 15; para $k = 1,...,15$, C^k é a matriz 15 x 15 tal que a k-ésima coluna contém somente uns e todas as demais colunas contêm zeros. P^k é uma matriz permutação com $\sum_{k=1}^{15} P^k = J$, onde J é a matriz de uns.

Os problemas gerados por sistemas de triplas de Steiner, os quais denotaremos por S_n ,são instâncias do Problema De Cobertura De Conjuntos de Mínima Cardinalidade (PCCMC), com custos unitários associados às colunas de A_n.

Sistemas de triplas de Steiner são casos particulares de configurações conbinatórias conhecidas como designs de blocos incompletos balanceados [22] , os quais são usados no design estatístico de experimentos. Por exemplo, se *n* drogas devem ser testadas em *m* pacientes, e cada paciente dever receber três drogas, um sistema de triplas de Steiner provê um design no qual cada par de drogas é testado em um paciente.

Fulkerson et al. [22] discutem experimentos computacionais com S_9, S_{15}, S_{27} e S_{45}. S_9 foi solucionada com um código de planos de corte após a geração de 44 cortes, mas essa abordagem falhou com os demais problemas. Utilizando um algoritmo de enumeração implícita similar ao descrito por Geoffriom [27], foi possível resolver os problemas S_{15} e S_{27}, mas não S_{45}. Avis [1] reporta que S_{45} foi solucionada em 1979, requerendo cerca de 2 ½ horas de processamento. Feo e Resende [18] utilizam uma heurística probabilística na solução dos problemas S_9, S_{15}, S_{27}, S_{45}, S_{81} e S_{243}. A heurística obtém a solução ótima para os problemas S_9, S_{15}, S_{27}, e S_{45}, enquanto que, para as demais instâncias, obtém o melhor resultado conhecido. Avis [1] destaca a dificuldade de solução desses problemas, mostrando que um algoritmo de enumeração utilizando relaxação por programação linear e/ou eliminação por dominância requer a verificação de $2\,exp(\sqrt{2n/3})$ Bsoluções parciais, onde *n* é o número de variáveis.

CAPÍTULO 3

Algoritmos para o Problema de Cobertura de Conjuntos

Estudos de problemas de cobertura de conjuntos foram conduzidos por Garfinkel e Nemhauser [24], Christofides e Korman [13] e Fisher e Kedia [21]. Algoritmos exatos foram desenvolvidos por Salkin e Koncal [44], Etcheberry [17] e Beasley [6]. Heurísticas foram apresentadas por Balas e Ho [3], Vasko e Wilson [48], Beasley [7] e Lopes [37].

Nesse capítulo são apresentados alguns dos principais algoritmos desenvolvidos para resolver o PCC de forma exata ou heurística. São destacados os métodos mais importantes para situar o desenvolvimento desse trabalho. Também são descritas com maior detalhe as heurísticas utilizadas para efeito de comparação com os algoritmos propostos.

3.1 - Métodos exatos

Os métodos mais conhecidos utilizados para a solução do PCC de forma exata são os métodos enumerativos e os de planos de corte. Nos métodos enumerativos é realizada uma enumeração, de forma implícita ou explícita, das soluções viáveis para o problema. Uma solução viável que minimiza a função objetivo é uma solução ótima. Algoritmos enumerativos mantêm um caminho de enumeração e utilizam um algoritmo de

busca em lista ou em árvore. Através das restrições do problema e da integralidade das variáveis, é possível abandonar a busca em certos nós que não conduzem à soluções melhores, de forma a promover a enumeração implícita de um grande número de nós. Algoritmos enumerativos foram propostos por Lemke et. al [34] e Etcheberry [17].

Algoritmos de planos de corte deduzem desigualdades suplementares às restrições do PCC a partir das próprias restrições e da condição de integralidade das variáveis. Estas desigualdades "cortam" parte da região viável do problema de programação linear correspondente, enquanto deixam a região viável do problema original intacta. Quando um número suficiente de hiper-planos forem gerados, o problema original terá a mesma solução ótima do problema de programação linear correspondente. Em geral, uma abordagem por planos de corte envolve a solução de uma sequência de problemas, da seguinte forma: Dado um PCC com uma matriz {0,1} A^p, inicialmente encontra-se um vetor solução inicial x^p, de custo c^p. Então, se uma nova desigualdade (ou corte) pode ser obtida tal que

(a) todas as soluções com custo menor que c^p a satisfazem e
(b) x^p nao a satisfaz,

então uma nova matriz A^{p+1} é obtida pela inclusão da desigualdade à matriz A^p. Se em qualquer iteração p, tal desigualdade não pode ser deduzida, o procedimento termina com uma solução ótima x^* com custo c^*, onde $c^* = \min_{0<k<p^*}[c^k]$. Bellmore e Ratliff [10] e Balas [3] descrevem métodos de planos de corte para solução do PCC.

Embora garantam uma solução ótima, os métodos exatos são, em geral, custosos do ponto de vista computacional, principalmente se aplicados a problemas de grande porte.

3.2 - Métodos heurísticos

3.2.1 - Heurísticas gulosas

Métodos heuríticos formecem "boas" soluções viáveis ao PCC, mas sem garantia de otimalidade. Muitas heuríticas para o PCC são por natureza construtivas. Primeiramente nenhuma coluna é representada no vetor solução, isto é, $x_j=0$ (j=1,...n). Iterativamente, baseado em algum critério, escolhe-se um índice q e faz-se $x_q = 1$. Esse critério é usado repetidamente até que uma solução viável seja encontrada. Em seguida, a solução pode ser melhorada removendo-se todas as colunas redundantes. Uma busca por vizinhança também pode ser utilizada para obter um ótimo local. Heurísticas gulosas foram apresentadas por Chvátal [12], Balas e Ho [3] e Vasko e Wilson [48].

O pseudo-código abaixo descreve uma heurística gulosa para o PCC. O algoritmo recebe como entrada os n conjuntos discretos $P_1,...P_n$, definidos no capítulo 2 e o vetor custo c, retornando uma cobertura J^*.

procedimento GULOSO (n, $P_1,...,P_n$, c, J^*)

1. **Faça** $J^* = \varnothing$;
2. **Enquanto** $\in j$ ($j = 1,...,n$) tal que $P_j \neq \varnothing$ **faça**;
3. k = argopt $\{f(c_j, |P_j|)\}$: $1 \leq j \leq n\}$;
4. $J^* = J^* \cup \{k\}$;
5. **para** j=1,...,n **faça**
6. $P_j = P_j - P_k$;
7. **fim para**;
8. **fim enquanto;**

fim GULOSO;

Na linha 1 do pseudo-código a cobertura é inicializada como um conjunto vazio. O loop delimitado pelas linhas 2-8 é repetido até que os n conjuntos $P_1,...,P_n$ estejam vazios, isto é, até que uma cobertura seja construída. Na linha 3 o índice k que maximiza ou minimiza $\{f(c_1, |P_1|),..., f(c_n, |P_n|)\}$ é selecionado, sendo f uma função que estabelece uma relação entre o custo das colunas e o número de linhas cobertas por elas. O índice k é adicionado a J^* na linha 4, e nas linha 5 e 6, P_k é subtraído dos conjuntos $P_1,...P_n$.

3.2.1.1 - Heurística de Chvátal

A heurística de Chvátal [12] é um caso particular do procedimento guloso descrito anteriormente. No seu trabalho, Chvátal destaca:

> Intuitivamente, a desejabilidade de se incluir uma coluna j em uma solução ótima aumenta com a razão $|P_j|/c_j$, que nos dá o número de pontos cobertos por P_j por unidade de custo.

Dessa observação resulta a função $f(c_j,|P_j|)$, dada pela razão $|P_j|/c_j$. Uma desvantagem dessa heurística é não garantir a obtenção de soluções mínimas, isto é, a cobertura construída ao final do procedimento pode possuir elementos supérfluos.

3.2.1.2- Heurística de Balas e Ho

Balas e Ho [3] propõem novas funções $f(c_j, |P_j|)$, de forma que a seleção do índice k na linha 3 do algoritmo guloso é realizada de forma a minimizar $f(c_j, |P_j|)$. Cinco funções são consideradas:

(1) c_j;
(2) $c_j / |P_j|$;
(3) $c_j / \log_2 |P_j|$;
(4) $c_j / |P_j| \log_2 |P_j|$;
(5) $c_j / |P_j| \ln |P_j|$;

Nos casos (3) e (4), $\log_2 |P_j|$ é substituído por 1 quando $|P_j| = 1$, e no caso (5), $\ln |P_j|$ é substituído por 1 quando $|P_j| = 1$ ou 2;

A função (1) inclui na cobertura J^* a cada iteração, a coluna de menor custo. A função (2) minimiza o custo unitário de cobertura de uma linha não coberta. As funções (3) a (5) selecionam a mesma coluna que (2) sempre que c_j=1, j=1,...n, mas (3) atribui menos peso enquanto (5) atribui mais e (4) ainda mais peso ao número $|P_j|$ de linhas cobertas pela coluna j, versus o custo c_j. Os testes de Balas e Ho mostraram que nenhuma das funções é significativamente melhor que outra. A melhor solução encontrada por qualquer das cinco funções não desviou o valor ótimo mais que 10,8%.

A heurística de Balas e Ho ainda inclui ao algoritmo guloso um passo adicional ao final do algoritmo, para remover as colunas redundantes da cobertura. Considerando os elementos de J^* na ordem decrescente dos valores c_j, são excluídos um a um elementos de J^*, enquanto J^* ainda for uma solução viável.

3.2.1.3 - Heurística de Vasko e Wilson

Vasko e Wilson [48] apresentam uma heurística gulosa (SCHEURI), onde utilizam além das funções apresentadas por Balas e Ho, duas novas funções $f(c_j, |P_j|)$, definidas como:

(6) $c_j / |P_j|^2$
(7) $(c_j)^{½} / |P_j|^2$

Vasko e Wilson, além da remoção das colunas redundantes da solução viável J^*, também acrescentam um novo passo ao algoritmo básico. O novo passo consiste em realizar uma busca na vizinhança de J^* por uma unidade, permutando uma coluna de J^* com uma coluna não pertencente a J^*. Se uma solução viável vizinha com menor custo é encontrada, a coluna permutada de J^* é substituída pela coluna correspondente na solução viável final.

A heurística SCHEURI ainda é modificada de modo que todas as sete funções possam ser usadas para gerar uma única solução para o PCC. Na heurística SCFUNC1TO7, a heurística SCHEURI é aplicada, sendo a função $f(c_j, |P_j|)$ determinada aleatoriamente a cada vez que uma coluna é incluída na solução J^*. Isto é, a cada execução do loop compreendido entre as linhas 2 e 8 do algoritmo guloso, é gerado um número aleatório entre 1 e 7 (uniformemente), e a função correspondente é aplicada na linha 3 do algoritmo.

Os resultados apresentados por Vasko e Wilson mostram que, a melhor das cinco soluções gerados por SCHEURI usando as funções (1) a (5) nunca superou a melhor solução obtida por SCFUNC1TO7. A melhor solução encontrada por SCFUNC1TO7 foi estritamente superior em 50% dos problemas testados.

As heurísticas gulosas apresentadas neste capítulo se caracterizam por obterem soluções com tempos computacionais consideravelmente pequenos, em apenas

uma iteração do algoritmo básico. As soluções encontradas são, no entanto, de qualidade regular, se comparadas com heurísticas de programação matemática mais sofisticadas.

3.2.2 - Heurística Probabilística

A heurística probabilística apresentado por Feo e Resende [18] é uma variação não determinística da abordagem gulosa de Chvátal, realizando uma busca em "boas" vizinhanças para as suas melhores soluções. O método distingue-se d o anterior na linha 3 do pseudo-código, onde o índice k é selecionado. Em vez de selecionar o índice k correspondente à coluna de maior razão $|P_j|/c_j$, a seleção é feita aleatoriamente de um conjunto de índices candidatos que possuem razão $|P_j|/c_j$ igual ou superior a

α x max $\{|P_j|/c_j : 1 \leq j \leq n\}$, onde $0 \leq \alpha \leq 1$. Além disso, os elementos supérfluos são eliminados da cobertura parcial $J^0 \cup \{k\}$. A seguir é apresentado o pseudo-código para implementação dessas variações:

procedimento PROBABILÍSTICO (n, P_1,...,P_n, c, a, I, J*)

1. **Faça** $\varepsilon = \infty$;
2. **Para** i=1,...,I **faça**
3. Para j=1,...,n faça $P^0_j = P_j$; $x_j^0 = 0$;
4. **Faça** $J^0 = \varnothing$;
5. **Enquanto** $P^0_j \neq \varnothing$, para algum j = 1,...,n **faça**;
6. $\tau = \max \{|P^0_j|/c_j : 1 \leq j \leq n\}$;
7. $K = \{j : |P^0_j|/c_j \geq a \text{ x } \tau, 1 \leq j \leq n\}$;
8. Selecione k aleatoriamente de K;
9. $J^0 = J^0 \cup \{k\}$; $x_k^0 = 1$;
10. **para** j=1,...,n **faça**

11. $P^0_j = P^0_j - P^0_k$;
12. **fim para**;
13. Remova os elementos supérfluos de J^0;
14. **fim enquanto;**
15. **se** $c.x^0 < \varepsilon$ **faça**
16. $\varepsilon = c.x^0$; $J^* = J^0$;
17. **fim para**;

fim PROBABILÍSTICO;

O algoritmo recebe como entrada os *n* conjuntos discretos $P_1,...P_n$, o vetor custo *c*, o parâmetro a e o número de iterações *I*, e retorna uma cobertura *J**. O método de Chvátal é executado uma vez, enquanto a versão probabilística é repetida *I* vezes no loop delimitado pelas linhas 2-17. Na linha 1, e, o custo da melhor cobertura encontrada até o momento é inicializado com um número consideravelmente grande. Na linha 3, os conjuntos de trabalho $P^0_1,...,P^0_n$ e o vetor solução x^0 são inicializados e na linha 4 a cobertura para a repetição *i*, J^0, é inicializada como vazia. No loop delimitado pelas linhas 5-14, a *i*-ésima cobertura é construída, Na linha 6, a razão $|P^0_j|/c_j$ máxima, *t*, dos conjuntos de trabalho $P^0_1,...,P^0_n$ é calculada. *K*, na linha 7, é o conjunto de índices correspondentes aos conjuntos cuja inclusão na cobertura J^0 irá cobrir pelo menos *a* x *t* elementos do conjunto $J = \{1,...,m\}$ por unidade de custo. Na linha 8 o índice *k* é selecionado aleatoriamente do conjunto *K* de índices candidatos, e o índice é adicionado a J^0 na linha 9. Na linha 10 os conjuntos $P^0_1,...,P^0_n$ são atualizados pelo conjunto P^0_k e na linha 13 qualquer elemento supérfluo é removido da cobertura parcial. A remoção de um número máximo de elementos supérfluos de uma cobertura parcial é, por si só, um problema de cobertura de conjuntos. Isto é, encontrar o subconjunto do conjunto selecionado de elementos J^0 que cubra as restrições satisfeitas até aquele ponto com o menor custo. Finalmente, na linha 15, se uma cobertura de menor custo é encontrada na iteração *i*, essa cobertura J^*, é guardada. A heurística gulosa de Chvátal é um caso especial do procedimento probabilístico, onde $\alpha = 1.0$ e $I = 1$;

Feo e Resende testaram sua heurística utilizando a classe de problemas de triplas de Steiner descritos no Capítulo 2 e compararam os resultados obtidos à heurística de Chvátal. A heurística probabilística provê as soluções ótimas para esses problemas, dadas as instâncias para as quais a solução ótima é conhecida. Além disso, provê as melhores soluções conhecidas para as outras instâncias testadas. O desempenho apresentado pela heurística probabilística em relação à heurística de Chvátal pode ser explicado da seguinte forma: primeiro, a heurística de Feo e Resende sempre produz soluções mínimas ao PCC, pela eliminação dos elementos supérfluos das coberturas parciais. Segundo, a introdução de um critério de seleção probabilístico dos elementos *k* a serem incluídos na cobertura parcial permite evitar a obtenção de soluções de mínimo local.

3.2.3 - Heurísticas de Relaxação

3.2.3.1 - Relaxação e otimização por subgradientes

Algoritmos baseados em métodos de relaxação são baseados na observação de que muitos problemas de programação inteira são modelados através de um conjunto de restrições relativamente fáceis e de um outro conjunto de restrições difíceis [21]. Cria-se então um problema relaxado, no qual as restrições difíceis são adicionadas na função objetivo, através de um vetor de multiplicadores, e em seguida eliminadas do conjunto total de restrições. O problema relaxado deve ser fácil de resolver e provê um limite inferior f_{li} para a solução ótima do problema original, o qual pode substituir uma relaxação de programação linear em, por exemplo, um algoritmo de enumeração. O método dos subgradientes tem a finalidade de maximizar o limite inferior obtido pelo problema relaxado, através do ajuste dos multiplicadores.

Em geral, as heurísticas de relaxação com otimização por subgradientes são caracterizadas pelos seguintes aspectos:

(1) utilizam uma relaxação do problema original para definir um limite inferior para o PCC;

(2) tentam maximizar o limite inferior via otimização de subgradientes;

(3) produzem soluções viáveis a partir das soluções do problema relaxado; o custo da solução viável define um limite superior f_{ls} para o PCC;

(4) através dos limites inferior e superior, tentam identificar as colunas e linhas que podem ser removidas do problema;

Consideremos o problema relaxado $R(w)$, onde $w \in R^m_+$ é o vetor de de multiplicadores associado à relaxação. Para ajustar os multiplicadores de forma a maximizar o limite inferior obtido pela solução de $R(w)$ é necessário resolver, a cada iteração, o dual associado ao problema $R(w)$, definido por:

$$\begin{aligned} (DR)\quad & \text{Max} \quad v(R(w)) \\ & \text{sujeito a } w \in R^m_+ \end{aligned}$$

onde $v(.)$ indica a solução ótima de (.).

Um método de gradientes pode ser usado para resolver o problema DR, de acordo com a função $v(R(w))$.

O método clássico [19] para obter o vetor de subgradientes g_w em uma dada iteração é dado por:

$$g_w = e - Ax_w$$

onde x_w é a solução do problema relaxado. A cada iteração é realizada a atualização dos multiplicadores, da seguinte forma:

$$w \leftarrow \max [0, w + t.g_w],$$

onde t é o tamanho do passo na direção dos subgradientes. Sabe-se que a escolha de t é fundamental para a eficiência do método. De acordo com [19], o tamanho de t deve ser tal que:

$$t_k \rightarrow 0 \text{ e } \sum_{i=1}^{k} t_i \, C \rightarrow \infty \text{ quando } k \rightarrow \infty ,$$

onde k é o número de iterações.

A fórmula clássica para t é

$$t = u(f_{ls},f_{li}) / \| g_w \|^2,$$

onde $u(f_{ls},f_{li})$ é uma função que tem como parâmetros os limites inferior e superior para a solução ótima do PCC.

Entre os diversos testes de parada do algoritmo, podemos citar os critérios de parada por satisfação das seguintes condições:

(1) subgradiente nulo ;
(2) $f_{ls} = f_{li}$;
(3) $(f_{ls}-f_{li}) \leq 1$;
(4) o tamanho de passo é muito pequeno para que haja uma alteração significativa no valor de f_{li} de uma iteração para outra;

(5) o valor de f_{li} não apresenta melhorias significativas durante um certo número de iterações consecutivas;
(6) um número máximo de iterações é atingido;

Ao final da heurística, f_{ls} é o valor da melhor solução viável encontrada e f_{li} é o valor do melhor limite inferior encontrado para o valor ótimo do problema original. A qualidade da solução final obtida pode ser avaliada pela expressão:

$$(f_{ls}-f_{li})/ f_{li}$$

O pseudo-código abaixo descreve uma heurística geral de subgradientes:

1. Inicialize $f_{ls} \leftarrow + \infty$; $f_{li} \leftarrow - \infty$;
2. Defina os multiplicadores iniciais, tal que $w_i \geq 0$ e $w_i \neq 0$, $i = 1,...,m$;
3. **Enquanto** (testes de parada = FALSO) **faça**
4. Resolva $R(w)$ e obtendo a solução x_w com custo f_r;
5. Construa uma solução viável x_v para o PCC usando x_w;
6. Faça $f_v \leftarrow \sum_{j=1}^{n} c_j x_{vj}$ D;
7. Faça $f_{ls} \leftarrow \min(f_{ls},f_v)$;
8. Faça $f_{li} \leftarrow \max(f_{li},f_r)$;
9. Execute os testes de redução do problema;
10. Faça $g_{wi} \leftarrow e_i - \sum_{j=1}^{n} a_{ij} x_{wj}$ E, $i = 1,...m$;
11. Calcule o novo tamanho do passo $t \leftarrow u(f_{ls},f_{li}) / \| g_w \|^2$;
12. Faça $w_i \leftarrow \max \{0,w_i + t.g_{wi}\}$, $i = 1,...,m$;
13. **fim enquanto**

3.2.3.2 - Relaxação Lagrangeana

A relaxação Lagrangeana clássica para o PCC é definida por:

$(L(w))$ Min $cx + w(e - Ax)$,
sujeito a $x \in \{0,1\}^n$

A solução x_w para o problema $L(w)$ pode ser obtida fazendo $x_{wj} = 1$ se
$(c_j - wa_j) \leq 0$; $x_{wj} = 0$ caso contrário, para $j = 1,...,n$.

A cada iteração é resolvido o Dual Lagrangeano (DL) associado a $L(w)$, definido por:

(DL) Max $v(L(w))$
sujeito a $w \in R^m_+$

É provado que $v(L(w))$ é uma função linear por partes e côncava em w[19], o que facilita a aplicação de um método de gradientes para resolver o problema DL.

A heurística de Beasley [7] usa a relaxação Lagrangeana $L(w)$ definida acima. A heurística é um caso particular do algoritmo de subgradientes genérico descrito na seção anterior, sendo caracterizada pela inicialização dos multiplicadores, a construção de soluções viáveis, o tamanho do passo e direção dos subgradientes, testes de redução e testes de parada. Beasley comparou sua heurística com a heurística de Balas e Ho [3] e as heurísticas de Vasko e Wilson [48]. A heurística de Beasley obteve soluções melhores (ou iguais) à heurística de Balas e Ho ou SCHEURI e SCFUNC1TO7 em 93 dos 96 problemas testes. Para os 76 problemas onde a solução ótima é conhecida a heurística de Beasley foi, entre todas, a que encontrou um número maior de soluções

ótimas. Além disso, a Heurística de Beasley apresentou o menor desvio percentual médio da solução ótima, dado por

(100 (solução encontrada - solução ótima) / solução ótima)

Essas observações apontam a heurística de Beasley como a heurística que apresenta melhores resultados entre as heurísticas testadas. No entanto, seu custo computacional é consideravelmente maior.

3.2.3.3 - Relaxação Surrogate

A relaxação Surrogate tem sido aplicada com sucesso em problemas de programação matemática. Em Lorena e Plateau [38] uma relaxação Surrogate é aplicada ao problema da mochila 0-1 e em Lopes [37] , podemos encontrar uma aplicação desse tipo de relaxação ao PCC. Esses trabalhos destacam a superioridade da relaxação Surrogate em relação à relaxação Lagrangeana, principalmente quanto a estabilidade na convergência da sequência de valores relaxados.

A relaxação Surrogate, como a relaxação Lagrangeana, tem o objetivo de criar um problema mais fácil de ser resolvido, provendo um limite para a solução ótima do problema original.

Para um vetor de multiplicadores $w \in R^m_+$, podemos definir a relaxação Surrogate Contínua do PCC, denotada por $\overline{S}\,F(w)$, como:

$$(\overline{S}\,G(w)) \qquad \text{Min} \quad cx$$
$$\text{sujeito a} \quad wAx \leq we$$

$$x \hat{I} [0,1)^n$$

onde ($\overline{S}$ H(w)) Irefere-se ao problema *S(w)* com todas a restrições de integralidade das variáveis relaxadas.

O Dual Surrogate associado é:

$$(\overline{DS}) \qquad \text{JMax} \quad v(\overline{S}\,K(w))$$
$$\text{sujeito a} \quad w \in R^m_+$$

O Dual Surrogate é normalmente mais difícil de ser resolvido do que o Dual Lagrangeano. Isto porque $v(\overline{S}\,L(w))$ é uma função quase côncava e semi-contínua superior em *w*, e todo método de busca para determinar os multiplicadores duais é complicado pela presença de patamares planos na função.

Em [37] é descrita uma propriedade do problema de programação linear $\overline{S}$ M(w), da qual pode ser derivada uma expressão analítica para sua solução ótima. Essa expressão é computacionalmente custosa. Lopes [37] descreve um método mais eficiente para resolver a relaxação, através da obtenção de um multiplicador ótimo λ^* para o dual D(λ) de $\overline{S}$ N(w), sendo

$$(D(\lambda)) \quad \text{Max} \quad S(\lambda)$$
$$\text{sujeito a} \quad \lambda > 0$$

$$\text{onde} \quad S(\lambda) \quad \text{Min} \quad [cx + l.w(e - Ax)]$$
$$\text{sujeito a} \quad x \in [0,1]^n$$

$S(\lambda)$ é uma função linear por partes e côncava em λ, com pontos de quebra $d_j = c_j / wa_j$, $j = 1,...n$. Para resolver $D(\lambda)$ basta encontrar o multiplicador ótimo $\lambda^* = d_j^* = c_{j^*} / wa_{j^*}$ que maximiza $S(\lambda)$.

Lorena e Plateau [38] relacionam as relaxações Surrogate e Lagrangeana através do dual $D(\lambda)$). Para $u = \lambda^*.w$, temos que:

$$v(\overline{S}\, O(w)) = v(D(\lambda)) = v(L(u))$$

Dessa forma, é possível usar o método dos subgradientes aplicado na relaxação Lagrangeana para atualizar os multiplicadores no caso Surrogate, devendo, contudo, multiplicá-los por λ^*.

A heurística de relaxação Surrogate (HS) proposta em [37], é a mesma heurística de subgradientes descrita na seção 3.2.3.1, com a solução a cada iteração de uma relaxação Surrogate contínua.

Em relação á heurística geral de subgradientes, a heurística HS apresenta as seguintes particularidades: a inicialização do vetor de multiplicadores, o procedimento utilizado para construção de uma solução viável, o controle do tamanho dos passos t tomados na direção dos subgradientes através do uso de λ^*, o qual reproduz os métodos de subgradientes tradicionais para a relaxação Lagrangeana, acelera a convergência e evita comportamento oscilatório, além dos testes de redução e testes de parada adotados.

Ao final do algoritmo de subgradientes é executada uma pesquisa de vizinhança de uma troca, similar à busca local descrita no passo 4 da heurística de Vasko e Wilson (Seção 3.2.1.3). Essa heurística, chamada Heurística de Final (HF), é aplicada a x_v e f_v, com o objetivo de melhorar a solução obtida por HS.

A heurística HS foi testada com o conjunto de problemas obtidos dos experimentos de Balas e Ho [3] e Beasley [7]. Se comparada com a heurística de

Beasley, HS apresenta resultados equivalentes com a metade do tempo computacional. HS apresenta um desvio percentual médio da solução ótima próximo ao da heurística de Beasley, mas obtém um número maior de soluções ótimas.

CAPÍTULO 4

Algoritmos Genéticos

4.1 - Conceitos básicos

Algoritmos Genéticos (AGs) [28] são uma classe de procedimentos iterativos que simulam o processo de evolução de uma população de estruturas sujeitas às forças competitivas prescritas no princípio de "sobrevivência do mais bem adaptado" de Darwin. O processo de evolução é aleatório, porém guiado por um mecanismo de seleção baseado na adaptação de estruturas individuais. A cada iteração do algoritmo, denominada nesse novo contexto como geração, um novo conjunto de estruturas é criado a partir de estruturas bem adaptadas selecionadas da geração anterior, pela troca de informação (bits ou blocos) entre essas estruturas. Novas informações são geradas aleatoriamente com uma dada probabilidade, e incluídas nas estruturas descendentes. Se projetado e implementado de forma adequada, um AG irá exibir um comportamento similar ao descrito na teoria de evolução de Darwin - estruturas com adaptação relativamente superior possuem uma chance maior de sobreviver e produzir descendentes ainda mais bem adaptados. O resultado será um aumento na adaptação global da população a cada nova geração.

Estabelecendo uma analogia entre o processo de evolução simulado por um AG e um processo de busca conduzido na solução de um problema de otimização de função, podemos perceber que ambos envolvem uma busca através de um conjunto de alternativas :

ESTRUTURA $\leftrightarrow$ SOLUÇÃO

ADAPTAÇÃO $\leftrightarrow$ VALOR DA FUNÇÃO OBJETIVO

EVOLUÇÃO $\leftrightarrow$ BUSCA

Supondo que um mapeamento um-para-um entre uma estrutura e o espaço de soluções possa ser encontrado; um ponto no espaço de soluções pode ser codificado como uma estrutura em um AG. Dada uma transformação apropriada, a adaptação de uma estrutura pode ser interpretada como uma aproximação da função objetivo original.

As principais diferenças entre os AGs e os procedimentos de busca tradicionais são descritas a seguir:

(a) AGs trabalham com uma codificação do conjunto de variáveis *x* em vez de trabalhar com as variáveis individuais.

O objetivo da busca genética é encontrar o conjunto de variáveis (estruturas) que obtenha o maior valor possível para a função objetivo do problema. Já os métodos de otimização tradicionais trabalham diretamente com as variáveis individuais, trocando seus valores de acordo com uma regra de transição particular.

Uma característica importante de um AG é a capacidade de explorar as similaridades de codificação das estruturas para guiar a busca no espaço de soluções, o que torna o método mais flexível em relação às limitações dos outros métodos (incluindo continuidade, existência de derivadas, unimodalidade, etc.).

(b) AGs realizam uma busca a partir de uma população de pontos, e não um único ponto no espaço de busca:

Enquanto a maioria das técnicas de busca seguem um mecanismo de busca por ponto, um AG mantém uma população de pontos a serem explorados. Nos métodos de busca convencionais, o processo de busca é originado a partir de um ponto único (uma busca por vizinhança) a cada iteração, utilizando alguma regra de transição. Essa busca ponto-a-ponto frequentemente leva à obtenção de picos falsos em espaços de busca multimodais. Pouco é aprendido durante o processo de busca mesmo que informação importante sobre o perfil da função possa ser recuperada a partir do espaço de busca já explorado. Um AG vai além de uma mera busca sequencial de cada ponto (estrutura) na população, identificando e explorando blocos de construção comuns de boas estruturas na população e "escalando" vários picos em paralelo. Esses blocos de construção correspondem a regiões no espaço onde boas soluções são prováveis de serem encontradas. Um AG usa um conjunto de operadores genéticos para selecionar, recombinar e alterar estruturas existentes de acordo com princípios significativos para direcionar a busca em direção a essas regiões.

(c) AGs utilizam regras de transição probabilísticas e não determinísticas.

Ao contrário de muitos procedimentos de busca, AGs utilizam regras de transição probabilísticas para guiar o processo de busca. O modo como um AG faz uso de probabilidade lhe confere uma característica que o distingue de uma mera busca aleatória. AGs utilizam escolhas aleatórias como uma ferramenta para guiar a busca em direção a regiões do espaço com prováveis melhorias.

Foi provado teoricamente [28] e empiricamente [29,46] que os AGs constituem métodos competitivos para a busca em espaços de busca complexos. O primeiro trabalho na área foi publicado por Holland [32]. Muitos trabalhos publicados estabelecem a validade do método em otimização de funções [29,35,46,50] e

aplicações em controle [33], entre outras [11,28]. Centralizaremos a atenção na sua aplicação em otimização de funções. Diversas tentativas de aplicar AGs a problemas de otimização combinatória têm sido realizadas. Entre os problemas tratados podemos citar o Problema do Caixeiro Viajante [29,36], Design de Layout de Facilidades [46], entre outros.

É usualmente aceito que a implementação de um algoritmo genético para solução de um problema deve possuir os seguintes componentes:

(1) Um mapeamento entre o espaço de soluções e o espaço de estruturas;
(2) Um procedimento de inicialização da população inicial;
(3) Uma medida de adaptação de estruturas;
(4) Um conjunto de operadores genéticos.

Em adição aos itens citados, uma implementação completa de um AG envolve ainda a especificação de um número de parâmetros, tais como o tamanho da população, número de gerações e função de escala.

O pseudo-código a seguir descreve um algoritmo genético genérico:

AG ( )
{ Algoritmo genético para otimização de uma função *f* }
início
    $t = 0$;
    inicializar G_t ;
    avaliar G_t ; ;
    enquanto não (condição de término) **faça**
        $t = t + 1$;
        selecione G_t de G_{t-1} ; {operador de reprodução }
        recombine G_t; {operadores de cruzamento e mutação }

avalie G_t ;;

fim enquanto

fim

Quando aplicado à solução de problemas de otimização de função, um AG opera da seguinte forma : Durante a geração *t* é mantida uma população *G(t)* de estruturas $s_{t1},...s_{tN}$, codificadas como strings de comprimento fixo (o tamanho *N* da população permanece fixo). Cada solução s_{ti} é avaliada calculando $f(s_{ti})$, uma medida de adaptação da estrutura. Uma nova população G_{t+1} é então criada: são selecionadas estruturas para reproduzir com base na sua adaptação relativa, e as estruturas selecionadas são recombinadas usando operadores genéticos. Três operadores básicos formam o núcleo da maioria das implementações de um AG: (1) operador de reprodução (2) operador de cruzamento e (3) operador de mutação. Outros operadores têm sido propostos, mas são ou derivados dos operadores citados ou operadores específicos projetados para um problema particular. Os operadores atuam no sentido de promover a melhoria da qualidade global das soluções a cada geração. A busca genética termina quando um certo critério de parada é atendido. Entre os critérios mais utilizados podemos destacar alguns, os quais implicam na satisfação das seguintes condições:

(1) um nível pré-determinado de qualidade de solução é atingido (por exemplo, adaptação média da população);
(2) a convergência é observada;
(3) um número máximo de gerações é atingido.

4.2 - Aspectos teóricos

4.2.1 - Conceito de esquema

Já foi citado que similaridades importantes entre estruturas mais bem adaptadas podem ser usadas para guiar a busca genética. Um elemento crucial no estudo de AGs é o conceito de esquema [28], o qual permite definir de que maneira uma estrutura codificada como um string de bits pode ser representativa de outras classes de estruturas, através de similaridades em algumas posições.

Em um AG, uma solução x (estrutura) é codificada como um string de comprimento k ($k > 0$) sobre um conjunto alfabeto V. O espaço de estruturas é definido como o conjunto de strings pertencente a V^k. Logo, o comprimento do espaço de estruturas é $|V|^k$, onde $|V|$ é o número de símbolos em V. Uma escolha comum *p*ara V é o conjunto binário {0,1}.

Um esquema é um string de comprimento k definido sobre o conjunto alfabeto $V \cup \{\#\}$. O símbolo # é um símbolo "coringa" e pode ser usado como substituto para qualquer símbolo em V. Um esquema corresponde a um plano no hipercubo definido pelo produto Cartesiano de V por V, sendo também conhecido como hiperplano na literatura. Por exemplo, dada uma estrutura s = 011011 e um esquema H = 0#1011, dizemos que s é um representante de H porque s pode ser derivado de H substituindo o símbolo # em H por '1'. Segue que o tamanho do espaço de esquema é $(|V|+1)^k$.

O número de representantes de um esquema H é denotado por $M(H)$. No exemplo citado anteriormente, M(0#1011) é dois. O conceito de esquema possibilita a definição de similaridades entre strings de comprimento finito sobre um conjunto alfabeto V e permite analisar o efeito dos operadores genéticos sobre os blocos de construção contidos em uma população. Duas importantes propriedades de um esquema H são definidas abaixo:

(1) Ordem de H, denotada por o(H): número de símbolos fixos em H (por exemplo, o(0#1011) = 5)

(2) Comprimento de *H*, denotado por $\delta(H)$: diferença entre as posições do primeiro e do último símbolo fixo em *H* (por exemplo, $\delta(\#1\#01\#\#) = 3$)

Para obter um limite do número de esquemas em uma população, primeiramente é apurado o número de esquemas contido em um string individual e então um limite superior é calculado para toda a população. Como exemplo, considere o string de comprimento 5 a seguir: 11111. Esse string possui 2^5 esquemas porque cada posição pode ter como valor um símbolo "coringa". Em geral, um string particular contém 2^d esquemas, onde *d* é o comprimento do string. Como resultado, uma população de tamanho *N* contém entre 2^d e $N.\ 2^d$ esquemas, dependendo da diversidade da população. Pode-se verificar que uma quantidade considerável de informações sobre similaridades está presente mesmo em populações de tamanho moderado.

4.2.2 - Função de Adaptação

A adaptação de uma estrutura é medida por uma função $m: S \rightarrow R^+$, onde *S* é o conjunto de todas as estruturas (isto é, V^k) e R^+ o conjunto dos números reais não negativos. Se a função *f* do problema de otimização subjacente é sempre positiva, então *f* pode ser usada diretamente como *m*. Caso contrário, *m* será uma transformação de *f*. A transformação dependerá da função objetivo original (se *f* é uma função de minimização ou maximização) e do mecanismo de seleção utilizado.

Os valores da função de adaptação, em conjunto com as similaridades entre as estruturas de uma população, são utilizados para dirigir o processo de busca.

4.2.3 - Operador de reprodução

O operador de reprodução designa a cada estrutura da população G_t uma chance de ser selecionada para permanecer na próxima população G_{t+1} proporcional à adaptação dessa estrutura. Esse operador atribui a cada estrutura uma taxa de amostragem $T(s,t)$, definida como o número esperado de descendentes a serem gerados a

partir dessa estrutura na geração *t*. Para coincidir com a teoria da evolução, dadas duas estruturas *s'* e *s*'', se $m(s') > m(s'')$, então $T(s',t) > T(s'',t)$. A definição mais adotada para a taxa de amostragem é $T(s,t) = m(s)/\overline{m}(s)$P, onde o numerador é a adaptação de *s* e o denominador é a adaptação média da população G_t. Pode-se observar que estruturas com adaptação acima da média possuem uma maior probabilidade de sobrevivência do que aquelas com adaptação abaixo da média.

4.2.4 - Operador de cruzamento

Se apenas o operador de reprodução atuar, a população tenderá a se tornar mais homogênea a cada geração. O operador de cruzamento é incluído em um AG por dois motivos: primeiro, ele introduz novas estruturas recombinando estruturas já existentes; segundo, ele tem um efeito de seleção, eliminando os esquema de baixa adaptação.

Um dos operadores de cruzamento mais simples, o operador de cruzamento por um único ponto, opera da seguinte forma: dadas duas estruturas *s'* e *s*'', elas trocam um substring de acordo com um ponto de cruzamento para formar duas novas estruturas. Supondo as estruturas *s*' e s'' como sendo 011|11 e 101|00 respectivamente, com um ponto de cruzamento indicado por |. Trocando os substrings à direita do ponto de cruzamento, duas novas estruturas 01100 e 10111 são criadas.

Para evitar comportamento caótico, nem todas as estruturas em uma nova população são geradas pelo operador de cruzamento. A probabilidade de aplicar o operador, ou simplesmente taxa de cruzamento, é denotada por p_c.

4.2.5 - Operador de mutação

O operador de mutação introduz mudanças aleatórias às estruturas em uma população trocando um símbolo em uma estrutura com uma probabilidade (ou taxa de mutação) p_m. Por exemplo, se p_m é 0.01, então a cada geração existe uma chance de 1% que uma estrutura da população seja alterada pela troca de um de seus símbolos. Esse operador possui o efeito de aumentar a diversidade da população, ou seja, evita que as estruturas tornem-se muito homogêneas. O aumento na diversidade das estruturas permite reduzir a possibilidade de convergência prematura.

4.2.6 - Teorema do Esquema

Através da análise da atuação dos operadores genéticos é possível derivar um dos resultados teóricos principais para explicar o poder dos AGs, o Teorema do Esquema de Holland [32].

O efeito do operador de reprodução em um esquema particular é fácil de compreender, uma vez que strings com maior adaptação possuem maior probabilidade de seleção. Dessa forma, esquemas com valores de adaptação acima da média da população irão receber um número maior de amostras na próxima geração, enquanto esquemas com valores abaixo da média da população receberão um número menor de amostras.

O operador de cruzamento atua sobre um esquema particular da seguinte forma: o operador preserva o esquema se não o corta, caso contrário, o esquema é rompido. Por exemplo, considere os dois esquemas 1***0 e **11*. O primeiro possui uma boa chance de ser rompido pelo cruzamento, enquanto o segundo é mais difícil de ser destruído. A probabilidade de sobrevivência de um esquema é tanto maior quanto maior for a probabilidade de que o ponto de cruzamento esteja fora dos limites definidos

pelas posições extremas fixas do esquema. Um limite inferior na probabilidade de sobrevivência de um esquema sob o operador de cruzamento pode ser dado por:

$$p_s \geq 1 - p_c \cdot \frac{d(H)}{d - 1}$$

uma vez que o ponto de cruzamento é selecionado aleatoriamente entre os d - 1 pontos possíveis, com uma dada probabilidade p_c.

Como resultado, esquemas de pequeno comprimento são mantidos pelo operador de cruzamento, e reproduzidos com uma boa taxa de amostragem pelo operador de reprodução.

Sob o operador de mutação, uma posição de um string sofre alteração aleatória com uma probabilidade p_m. Para que um esquema H sobreviva, todas as posições fixas nesse esquema devem sobreviver. Uma vez que uma única posição (bit) sobrevive com uma probabilidade (1 - p_m), e cada mutação é estatisticamente independente, um esquema particular sobrevive quando cada uma das $o(H)$ posições fixas de um esquema sobrevive. Multiplicando a probabilidade de sobrevivência (1 - p_m) por ela mesma $o(H)$ vezes, obtemos a probabilidade de sobrevivência sob o operador de mutação, $(1 - p_m)^{o(H)}$. Para pequenos valores de p_m ($p_m << 1$), a probabilidade de sobrevivência de um esquema pode ser aproximada pela seguinte expressão:

$$(1- o(H). p_m)$$

O valor de p_m deve ser suficientemente pequeno para que atuação do operador de mutação não provoque o rompimento de esquemas.

Das observações anteriores podemos entao derivar o seguinte teorema:

Teorema do Esquema: O número de representantes esperados de um esquema H em G_{t+1}, denotado por $M(H,t+1)$, usando os três operadores descritos anteriormente, é dado pela seguinte expressão :

$$M(H,t+1) = M(H,t)\frac{m(H)}{\overline{m}}\left[1 - p_c\frac{d(H)}{d-1} - o(H)\,p_m\right]$$

O teorema acima afirma que H receberá um número crescente de representantes em G_{t+1} se possui um adaptação alta ($m(H)$), pequeno comprimento ($\delta(H)$), e baixa ordem ($o(H)$). Desde que o valor exato de $m(H)$ é desconhecido, ele é estimado pela adaptação média dos representantes de H em G_t. Os resultados de Holland são válidos para as implementações que utilizam uma função de adaptação da seguinte forma:

$$m(H) = af(x) + b,$$

onde a e b são constantes e $b > 0$ se f é uma função de maximização e $b < 0$ caso contrário.

Em uma população, os esquemas que atendem aos três critérios apresentados acima são também denominados *blocos de construção*.

4.2.7 - Propriedade intrínseca de paralelismo de um AG

Como foi visto, o número de esquemas encontrados em uma população é consideravelmente grande mesmo em populações pequenas. Para utilizar todas essas informações em uma geração com um tempo computacional aceitável, é necessário que algum tipo de processamento paralelo seja realizado. Outro resultado teórico importante é dado por uma propriedade verificada por Holland [32], denominada Propriedade Intrínseca de Paralelismo de um AG. Pelo teorema do esquema, sabemos que cada

esquema com pelo menos um representante em G_t irá receber $M(H,t+1)$ representantes em G_{t+1}. Essa propriedade afirma que dos 2^d a $N.\ 2^d$ esquemas contidos em uma população, o número de esquemas efetivamente processados em uma geração é pelo menos $O(N^3)$, onde d é o comprimento de uma estrutura e N é o número de estruturas na população. Em outras palavras, um AG aloca esforço de busca a $O(N^3)$ hiperplanos distintos (isto é, esquemas) no espaço de soluções em paralelo. Mais importante, essa busca é conduzida implicitamente, sem nenhuma memória especial além da população de estruturas. A habilidade de explorar diferentes regiões do espaço de soluções em paralelo é uma característica única dos AGs e a qualidade da solução obtida pode justificar o custo de manter uma população de soluções.

CAPÍTULO 5

Algoritmos genéticos aplicados ao Problema de Cobertura de Conjuntos

5.1 - Esquema de codificação

O primeiro passo para implementar com sucesso um algoritmo genético para problemas de otimização combinatória é usar um esquema de codificação que permita um mapeamento entre o espaço de soluções e o espaço de estruturas de modo a preservar o significado do problema original.

Uma codificação natural e direta pode ser derivada para o PCC, de forma que o vetor solução *x* do problema original seja representado como um string com *n* entradas binárias, onde

$$x_j = \begin{cases} 1 \text{ se a coluna } j \text{ pertence a solucão} \\ 0 \text{ caso contrario} \end{cases}$$

Cada bit em um string é associado a uma coluna da matriz *A*. O bit é um se a coluna *j* está incluída na solução e zero caso contrário. O esquema de codificação adotado permite selecionar um domínio finito e limitado para a busca, utilizando apenas o alfabeto binário {0,1}.

5.2 - Estruturas de dados

AGs processam populações de estruturas. Naturalmente, a estrutura de dados primária para um AG é uma população de strings. Uma população pode ser implementada como um vetor de *N* estruturas, onde cada estrutura contém o genótipo (o cromossomo artificial representado por um string de bits), o fenótipo (a decodificação do genótipo), e um valor de adaptação (função objetivo), juntamente com outras informações auxiliares. O esquema utilizado nesse trabalho para armazenar uma população é ilustrado na figura abaixo:

Número da estrutura	Genótipo (String de bits)	Fenótipo (Custo)	Função de Adaptação μ(x)	Outros
1	0100...0110	67	785	.

2	1110...1001	15	147	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
N	0001...0101	8	42	.

Fig. 5.1 - Esquema para armazenamento de uma população de estruturas em um AG.

No AG implementado nesse trabalho são utilizadas duas populações distintas, de modo a simplificar a formação de descendência e substituição de estruturas pais. Com um esquema de duas populações é relativamente simples criar descendentes a partir dos membros da população original usando os operadores genéticos, alocá-los para uma nova população e tomar essa nova população como origem na próxima geração. Ganha-se pela simplicidade da implementação mas, por outro lado, não garante-se um método eficiente de armazenamento.

Tendo em vista que as matrizes de incidência A do PCC são tipicamente grandes e esparsas, e contêm apenas os valores zero e um, é necessário apenas armazenar os índices das linhas e colunas onde $a_{ij} = 1$ ($i = 1,...m$; $j = 1,...,n$). Nos algoritmos implementados nesse trabalho foi utilizada uma "versão coluna" da matriz, armazenada em um vetor de dimensão n de estruturas de dados. Cada estrutura armazena uma variável inteira correspondente à cardinalidade $|P_j|$ e um ponteiro para um vetor com o conjunto P_j das linhas cobertas por j. Também é utilizado um vetor de dimensão m para armazenamento das linhas, onde são armazenadas duas variáveis inteiras, correspondendo ao número de colunas que cobrem a linha e ao índice da coluna com menor custo que cobre a linha. A figura abaixo mostra um exemplo da estrutura de dados utilizada para armazenar a matriz de incidência do PCC.

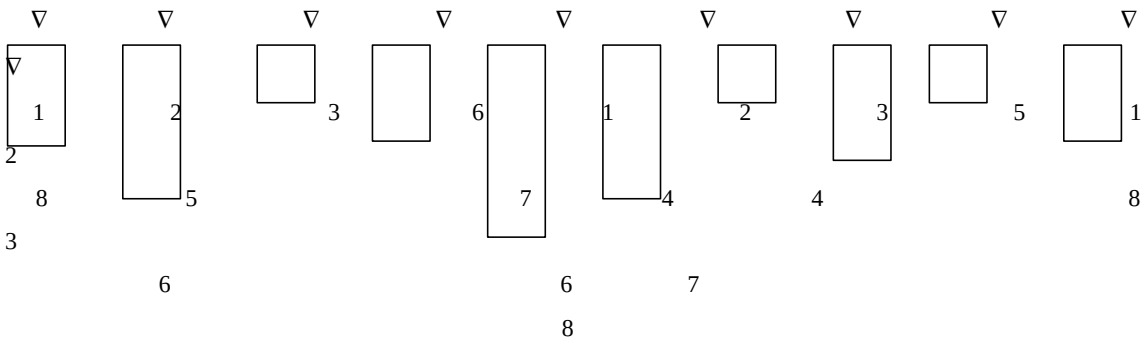
Matriz de incidência:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Estruturas de dados:

Vetor coluna

2	3	1	2	4	3	1	2	1	2



Vetor linha

3	3	3	2	2	3	2	3
1	2	3	5	2	2	4	1

Fig. 5.2 - Estrutura para armazenamento da matriz de incidência do PCC.

As colunas da matriz de incidência são armazenadas no vetor coluna ordenadamente de acordo com seus custos, sendo que as colunas com custos iguais são ordenadas na ordem crescente do número de linhas que elas cobrem.

5.3 - Operador de viabilidade

As soluções geradas pelos operadores de cruzamento e mutação podem violar as restrições do problema, isto é, algumas das linhas podem não ser cobertas. Optou-se por restringir a busca genética a apenas soluções viáveis, utilizando para isso um operador de viabilidade. O operador de viabilidade atua da seguinte forma:

Seja I = conjunto de linhas

J = conjunto de colunas

S = conjunto de colunas em uma solução

Y_i = conjunto de colunas que cobrem a linha i, $i \in I$

X_j = conjunto de linhas cobertas pela coluna j, $j \in J$

w_i = número de colunas que cobrem a linha i, $i \in S$

Z = conjunto de linhas não cobertas

1. Inicialize $w_i = |S \cap Y_i|$, $\forall\ i \in I$

2. Inicialize $Z = \{i \mid w_i = 0, \forall\ i \in I\}$

3. Para cada linha i em Z, em ordem crescente de i:

3.1. Encontre a primeira coluna j em Y_i (em ordem crescente de j)

3.2. Adicione j a S e faça $w_i = w_i + 1$, $\forall\ i \in X_j$. Faça $Z = Z - X_j$;

4. Para cada coluna j em S (em ordem decrescente de j), se $w_i \geq 2$, $\forall i \in X_j$, faça $S = S - j$ e $w_i = w_i - 1$.

O operador de viabilidade, além de garantir a viabilidade de todas as soluções geradas, também atua no sentido de promover uma otimização local, através da remoção das colunas redundantes.

5.4 - Inicialização da população de estruturas

Nas versões clássicas de um AG, as estruturas são inicializadas aleatoriamente, de modo que o espaço de busca de soluções apresente um nível alto de diversidade. Entretanto, em algumas abordagens encontradas na literatura [35], antes de aplicar o AG, as estruturas são primeiramente otimizadas utilizando alguma heurística de busca local. Os resultados obtidos nesses trabalhos comprovam que a diversidade da população inicial não constitui um fator determinante na obtenção de boas soluções viáveis. A idéia consiste em aplicar os operadores genéticos a estruturas que já são bem adaptadas. Nesse trabalho foi utilizada uma heurística de busca local para gerar uma população inicial de estruturas.

As estruturas são inicializadas com uma heurística de busca local baseada na heurística de Chvátal (Seção 3.2.1.1). Inicialmente uma solução viável para o problema é obtida, pela aplicação da heurística de Chvátal. O procedimento para inicialização das demais estruturas é baseado na seguinte observação:

Estando as colunas da matriz A ordenadas na ordem crescente de seus custos, e as colunas com custos iguais ordenadas na ordem crescente do número de linhas que elas cobrem, uma solução viável para o PCC obtida pela heurística de Chvátal apresenta o seguinte aspecto:

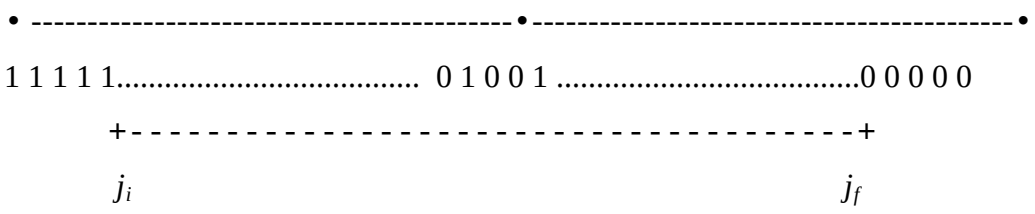


Fig. 5.3 - Solução obtida pela heurística gulosa de Chvátal

isto é, as primeiras colunas sempre estão representadas na solução viável encontrada, as últimas colunas não fazem parte dessa solução, enquanto as colunas intermediárias apresentam um comportamento variado. Isso nos leva a crer que uma melhoria na solução viável só pode ser obtida por uma busca limitada ao intervalo definido por esses pontos extremos.

As outras N - 1 estruturas da população são inicializadas realizando uma busca por unidade na vizinhança da estrutura já inicializada, e limitando a busca ao intervalo descrito acima. Para uma dada estrutura, a vizinhança por unidade é obtida complementando cada bit no string, um por vez. Definimos então θ_j como a variação na adaptação da estrutura pela complementação do j-ésimo bit. A partir da vizinhança, a heurística seleciona e complementa o bit que corresponde à coluna j que minimiza θ_j,

para todo *j*. O procedimento de inicialização por busca local pode ser melhor compreendido através do seguinte pseudo-código:

procedimento INICIALIZAÇÃO POR BUSCA LOCAL

1. Inicialize s_0 aplicando a heurística gulosa de Chvátal;
2. **Para** i=1,...,N -1 **faça**

2.1. Para $j \hat{I} U$, U=$\{j_i,....j_f\}$ é o conjunto das variáveis compreendidas no intervalo de busca, onde $1 < i < f < n$

$k = \{j: \theta_k = \max \theta_j, \}$;

2.2. Construa s_i, permutando o k-ésimo bit da estrutura s_0 e estendendo a solução para uma solução viável utilizando o operador de viabilidade.

2.3. Faça $U = U - \{k\}$

5.5 - Função de Adaptação

Para a busca genética restrita a soluções viáveis, a função de adaptação é diretamente obtida de seu valor de função objetivo.A função de adaptação μ pode então ser calculada por:

$$m = \sum_{j=1}^{n} c_j x_j$$

onde x_j corresponde ao valor do j-ésimo bit da solução e c_j é o custo da coluna correspondente.

5.6 - Operador de reprodução

Em um AG, o operador de reprodução pode ser implementado de diversas formas, selecionando estruturas de uma população para povoar uma nova população e reproduzirem-se de acordo com os valores de suas funções de adaptação.

Para implementar o operador de reprodução foi utilizado o método da roleta ponderada [28]. Nesse método cria-se uma roleta ponderada, de modo que cada estrutura da população corrente ocupe uma "setor" da roleta em proporção à sua adaptação. O operador de reprodução atua fazendo "girar" a roleta e selecionando a estrutura correspondente ao setor sorteado. A cada giro da roleta, uma estrutura da população é selecionada com probabilidade proporcional à sua adaptação. Dessa forma, as estruturas com maiores valores de adaptação possuirão um número maior de descendentes na próxima geração.

Como exemplo, considere a seguinte população de estruturas:

No.	String de bits	Função de adaptação	% do Total
1	0 1 1 0 1	157	13.3
2	1 1 0 0 0	83	7.1
3	0 1 0 0 0	568	48.2
4	1 0 0 1 1	370	31.4
Total		1178	100.0

Somando as adaptações das estruturas obtém-se um total igual a 1178. A percentagem do total de adaptação da população também é demonstrada na tabela. A

roleta ponderada correspondente para a reprodução nessa geração pode ser visualizada pela seguinte figura:

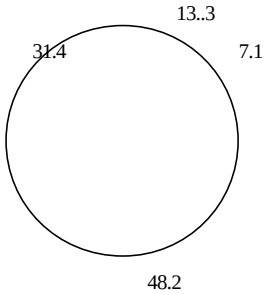


Fig. 5.4 - Roleta ponderada para reprodução de uma população de estruturas

5.7 - Operadores de cruzamento

O operador de cruzamento mais utilizado em abordagens por AG é o operador de ponto único, já descrito nesse capítulo, o qual promove a troca de informação entre duas estruturas de acordo com um ponto escolhido aleatoriamente. O operador de cruzamento por dois pontos também é muito utilizado na literatura e promove a troca dos substrings compreendidos entre dois pontos escolhidos aleatoriamente. O operador pode ser melhor ilustrado pela figura abaixo:

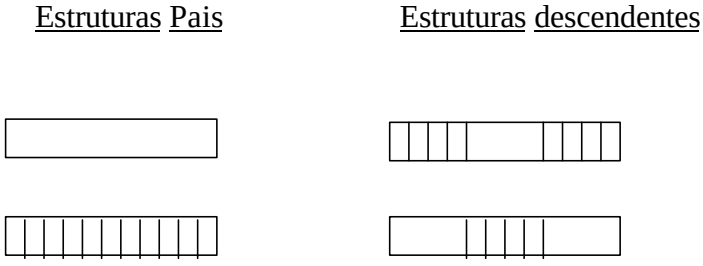


Fig. 5.5 - Operador de cruzamento por dois pontos

Foram testados nesse trabalho as duas variações de operadores de cruzamento.

5.8 - Operador de mutação adaptativo

O operador de mutação atua sobre cada nova solução gerada pelo operador de cruzamento. Uma das principais dificuldades dos AGs (e da maioria dos algoritmos de busca) é a ocorrência de convergência para uma solução sub-ótima. Foi observado que esse problema está intimamente ligado à perda de diversidade na população. Uma fonte de perda de diversidade é o aparecimento ocasional de um "super-indivíduo" que em poucas gerações toma conta da população. Embora vários estudos em torno de um bom conjunto de parâmetros para algoritmos genéticos tenham sido realizados [15,30,45], optou-se por implementar um operador de mutação adaptativo. Inicialmente, em uma execução, uma taxa de mutação baixa pode ser proveitosa para evitar que o trabalho do cruzamento e da busca local sejam destruídos. Entretanto, mais adiante na execução, a população tende a convergir, tornando-se muito homogênea. Nesse ponto uma taxa de mutação maior pode ajudar a introduzir diversidade na população, expandindo assim o espaço de busca e permitindo escapar a possíveis mínimos locais. O valor da taxa de mutação a cada geração é calculado pela seguinte expressão:

$$p_m = p_f * exp\ (\ \eta * (t - T\ )\ /\ T\ )$$

onde p_f é um valor pré-estabelecido para a taxa de mutação final, η é uma constante ($\eta = 2.3049$), t é a geração corrente e T é o número total de gerações. A constante η foi escolhida de forma a inicializar o valor da taxa de mutação em 0.005. O algoritmo inicia, desse modo, com um valor bem pequeno de taxa de mutação, o qual varia exponencialmente ao longo da busca genética.

5.9 - Função de escala

Também para evitar convergência prematura, foi utilizada uma função para escalar os valores das funções de adaptação das soluções encontradas na busca genética. Isso porque, à medida que a população converge, a faixa de valores de adaptação na população diminui e as probabilidades de um indivíduo ser selecionado tornam-se quase iguais. Para favorecer a seleção dos indivíduos relativamente mais bem adaptados, uma função de adaptação relativa é calculada para cada estrutura na população, da seguinte forma:

$$\mu'_i = \mu_i - \min(\mu_i, i = 1,...N)$$

onde μ'_i e μ_i denotam as funções de adaptação escalada e original, respectivamente.

5.10 - Estratégia elitista

Opcionalmente, também foi utilizada uma estratégia elitista: caso a melhor estrutura na população anterior não tenha sido selecionada para inclusão na nova população, essa estrutura é incluída na população de qualquer maneira. A idéia por trás dessa estratégia é não perder "acidentalmente" a melhor estrutura encontrada até o momento. A prática em AGs tem mostrado que a estratégia é usualmente vantajosa [35].

CAPÍTULO 6

Experimento Computacional e Análise de Resultados

O experimento foi conduzido com o objetivo de explorar o potencial dos AGs na solução de problemas de cobertura de conjuntos de grande porte e/ou difíceis computacionalmente. Os resultados obtidos pela aplicação dos algoritmos a duas classes

de problemas foram comparados com os métodos conhecidos como mais eficientes na sua solução.

Os algoritmos foram inicialmente aplicados a um conjunto de problemas teste obtidos dos trabalhos de Balas e Ho [3] (conjuntos de problemas 4, 5 e 6) e de Beasley [7] (conjunto de problemas A) e utilizados por Lopes [37]. Os resultados alcançados foram comparados com os resultados da heurística de relaxação Surrogate (HS). Estes problemas foram obtidos atraves da transferência eletrônica de arquivos de uma biblioteca de problemas da Inglaterra [8].

Antes de aplicar os algoritmos propostos a esses problemas, foi realizada uma redução inicial para diminuir ao máximo o número de linhas e colunas de suas matrizes de incidência. Para isso foram utilizados os testes de redução descritos na Seção 2.2. A seguinte tabela demonstra o tamanho dos problemas teste antes e depois de efetuadas as reduções iniciais (valores médios para cada conjunto de problemas).

Conjunto de Problemas	Número de linhas antes da redução	Número de colunas antes da redução	Densidade antes da redução (%)	Número de linhas depois da redução	Número de colunas depois da redução	Densidade depois da redução (%)	Número de problemas no conjunto
4	200	1000	2	182	202	2.4	10
5	200	2000	2	182	218	2.4	10
6	200	1000	5	200	237	5.3	5
A	300	3000	2	300	394	2.3	5
B	300	3000	5	300	490	5.2	5
C	400	4000	2	400	561	2.2	5

D	400	4000	5	400	697	5.1	5

Tabela 6.1

Os algoritmos foram testados ainda em uma classe de problemas que surgem no cálculo de dimensão-1 das matrizes de incidência de sistemas de triplas de Steiner, tendo em vista que esses problemas possuem muito menos variáveis que numerosos problemas resolvidos na literatura, além de serem difíceis de computar e verificar [18]. Os problemas S_n foram gerados recursivamente através da técnica padrão de Hall [31] para sistemas de Steiner nos quais $n = 3^k$ (k = 1,2,3,...) e das técnicas descritas por Fulkerson [22] para construção de A_{15} e A_{45}. Esses problemas não podem ser reduzidos. Os resultados obtidos foram comparados com os resultados alcançados pela heurística probabilística (HPR).

Para um PCC com densidade f, o número médio de colunas em cada solução é igual a $1/f$. Uma população de tamanho N contém em média N/f colunas. Para que uma população de estruturas cubra todo o domínio das soluções ela deve conter todo o conjunto de colunas, isto é, devemos ter $N/f \geq n$. Ou seja, para prover uma cobertura adequada do domínio de busca para o PCC, o tamanho da população deve ser tal que $N \geq nf$. Para problemas de grande porte, o tamanho da população pode se tornar demasiadamente grande. Por exemplo, um PCC com 500 colunas e 5% de densidade requer uma população de tamanho maior ou igual a 2500 estruturas. Nos experimentos computacionais realizados optou-se por manter uma população de tamanho fixo igual a 100 a cada geração.

O algoritmo foi testado utilizando os operadores de cruzamento por um e dois pontos. Denominaremos as duas versões como AG1 e AG2, respectivamente.

Embora os estudos conduzidos por DeJong [15], Grefenstette [30] e Schaffer et al. [45] indiquem que um bom valor para a escolha da taxa de cruzamento seja p_c =

0.95, optou-se por utilizar uma taxa de cruzamento menor. O algoritmo utilizando o operador de cruzamento por um único ponto foi testado nos menores problemas de cada sub-grupo (4,5,6,A,B,C,D,S). Os testes foram realizados utilizando cinco sementes diferentes para o gerador de números aleatórios. Foram obtidos os seguintes resultados, em termos de desvio percentual médio da solução ótima:

P_c	Desvio percentual médio da solução ótima
0.5	0.5227
0.6	0.4004
0.7	0.4790
0.8	0.4728
0.9	0.4146

Tabela 6.2

Selecionamos a taxa de cruzamento que apresentou menor desvio percentual médio da solução ótima, ou seja, $p_c = 0.6$.

A taxa de mutação varia exponencialmente ao longo da busca genética, conforme descrito na seção 5.8. Foi estipulado como limite para a taxa de mutação o valor $p_f = 0.05$.

Ambos os algoritmos foram testados em todos os problemas de cada grupo, utilizando cinco sementes diferentes para o gerador de números aleatórios. Isto porque algoritmos genéticos constituem técnicas estocásticas, e cada busca provê resultados potencialmente diferentes para cada semente aleatória utilizada. Um número máximo de gerações igual a 1000 foi pré-estabelecido como critério de parada dos algoritmos.

Optou-se por utilizar a linguagem C para codificar os algoritmos, uma vez que essa linguagem provê um conjunto e funções para alocar e manipular vetores dinamicamente, possibilitando a implementação eficiente das estrututras de dados

utilizadas.Os algoritmos foram implementados utilizando o compilador Borland C^{++} 3.1. Os testes foram realizados em um computador PC-486 DX-2 de 66MHz, com sistema operacional MS-DOS 6.22.

As tabelas seguintes identificam os problemas teste do primeiro grupo, os tamanhos das soluções ótimas e os resultados obtidos por AG1 e AG2, comparados à heurística HS. São demonstrados nas tabelas, além dos limites obtidos para as soluções viáveis dos problemas em cada execução, a geração média em que cada solução viável mínima foi obtida, o tempo médio de processamento (em segundos) até sua obtenção e o tempo total médio de execução do algoritmo.

AG1

Problemas	Solução viável mínima					Solução ótima	Geração média melhor solução	Tempo médio melhor solução	Tempo médio busca genética
	1	2	3	4	5				
4.1	433	433	433	430	433	429	308	77.04	247.97
4.2	◆	◆	◆	◆	◆	512	575	202.36	344.34
4.3	◆	◆	◆	◆	◆	516	283	108.80	345.93
4.4	495	495	495	495	495	494	59	32.51	330.22
4.5	514	514	514	514	◆	512	754	264.01	349.12
4.6	562	562	564	562	561	560	751	279.12	371.15
4.7	432	432	◆	◆	432	430	66	30.29	284.12
4.8	493	493	493	493	493	492	86	43.92	346.37
4.9	646	647	646	646	646	641	466	183.55	384.23
4.10	◆	◆	◆	◆	◆	514	136	48.24	284.56
5.1	◆	257	254	254	254	253	951	335.40	353.02
5.2	307	307	307	307	307	302	635	252.76	390.49
5.3	228	228	228	228	228	226	254	100.93	352.58
5.4	243	◆	◆	243	◆	242	120	51.54	341.92
5.5	◆	◆	◆	◆	◆	211	360	98.24	258.07
5.6	214	◆	216	◆	◆	213	940	301.81	322.53
5.7	◆	300	◆	◆	◆	293	491	173.27	342.31
5.8	289	289	289	290	289	288	974	370.27	380.33
5.9	◆	◆	◆	◆	◆	279	476	178.99	360.16
5.10	◆	◆	◆	◆	◆	265	316	112.29	328.90

6.1	◆	◆	◆	◆	◆	138	577	217.12	374.89
6.2	149	149	148	149	148	146	145	60.42	402.86
6..3	148	148	148	148	148	145	176	73.59	392.58
6.4	◆	◆	◆	◆	◆	131	306	116.66	367.31
6.5	◆	◆	◆	◆	◆	161	550	229.23	421.92

Tabela 6.3 ◆ -

Solução ótima

AG1

Problemas	Solução viável mínima 1	2	3	4	5	Solução ótima	Geração média melhor solução	Tempo médio melhor solução	Tempo médio busca genética
A.1	255	256	255	256	255	253	802	497.36	609.95
A.2	257	256	253	257	258	252	748	481.11	634.01
A.3	238	237	236	236	237	232	821	532.41	641.21
A.4	235	235	236	236	236	234	334	396.90	605.88
A.5	◆	237	◆	◆	◆	236	680	427.01	608.79
B.1	◆	◆	◆	◆	◆	69	509	364.59	762.80
B2.	◆	◆	◆	◆	◆	76	362	276.13	782.64
B.3	◆	◆	◆	◆	◆	80	473	358.79	812.03
B.4	◆	◆	◆	◆	◆	79	381	284.67	794.23
B.5	◆	◆	◆	◆	◆	72	47	47.25	764.34
C.1	229	229	229	230	◆	227	768	671.87	915.88
C.2	221	◆	◆	◆	221	219	551	507.69	927.53
C.3	248	249	247	252	249	243	482	506.98	1020.71
C.4	220	222	222	222	220	219	803	694.56	907.58
C.5	◆	216	217	216	217	215	468	425.11	899.29
D.1	◆	◆	◆	◆	◆	60	333	340.22	1087.75
D.2	67	67	◆	67	67	66	260	286.48	1144.40
D.3	75	74	76	76	74	72	164	202.42	1164.89
D.4	◆	64	63	63	63	62	90	101.70	1089.34
D.5	◆	◆	◆	◆	◆	61	601	616.98	1089.01

Tabela 6.4
AG2

Problemas	Solução viável mínima 1	2	3	4	5	Solução ótima	Geração média melhor solução	Tempo médio melhor solução	Tempo médio busca genética
4.1	430	433	430	430	430	429	32	23.62	354.95
4.2	◆	◆	◆	513	◆	512	486	252.09	497.31
4.3	◆	◆	◆	516	520	516	271	141.21	463.13
4.4	495	495	495	495	495	494	342	152.80	422.47
4.5	514	514	514	514	514	512	645	316.76	479.67
4.6	562	562	561	561	562	560	670	313.13	502.47
4.7	432	◆	◆	◆	◆	430	518	208.02	414.29
4.8	497	499	493	497	497	492	22	35.16	482.14
4.9	646	650	641	655	645	641	605	298.02	496.32
4.10	517	◆	517	516	◆	514	164	76.26	410.11
5.1	◆	253	254	254	254	253	700	317.97	461.70
5.2	307	307	◆	307	307	302	730	369.45	523.02
5.3	228	228	228	228	228	226	223	123.46	477.03
5.4	243	◆	◆	◆	◆	242	114	67.58	443.74
5.5	◆	◆	◆	◆	◆	211	159	61.98	334.34
5.6	◆	◆	216	◆	◆	213	275	128.79	438.35
5.7	◆	297	◆	297	◆	293	101	64.95	452.42
5.8	289	289	289	289	289	288	116	85.55	501.04
5.9	◆	◆	◆	◆	◆	279	215	120.66	485.66

5.10	◆	◆	◆	◆	◆	265	329	152.47	438.79
6.1	142	◆	◆	140	◆	138	634	320.82	521.92
6.2	147	148	148	148	147	146	849	473.52	565.49
6.3	148	147	148	148	148	145	24	24.40	512.36
6.4	◆	◆	◆	◆	◆	131	269	145.16	506.65
6.5	◆	◆	◆	◆	◆	161	406	232.86	573.63

Tabela 6.5

AG2

Problemas	Solução viável mínima 1	2	3	4	5	Solução ótima	Geração média melhor solução	Tempo médio melhor solução	Tempo médio busca genética
A.1	255	255	255	255	255	253	620	528.35	836.81
A.2	254	255	255	257	258	252	807	729.62	900.55
A.3	237	236	235	236	234	232	215	256.15	882.25
A.4	237	◆	236	235	◆	234	743	640.93	851.10
A.5	237	◆	◆	237	◆	236	90	130.33	845.99
B.1	70	70	◆	◆	70	69	585	546.92	923.57
B2.	◆	78	◆	◆	◆	76	117	139.84	975.49
B.3	◆	◆	◆	◆	◆	80	312	309.34	1021.26
B.4	◆	◆	82	80	◆	79	891	898.85	1022.03
B.5	◆	◆	◆	◆	◆	72	12	40.22	928.13
C.1	229	232	229	231	231	227	226	352.47	1204.01
C.2	221	221	221	221	◆	219	142	254.01	1235.88
C.3	249	251	250	251	250	243	683	917.91	1303.68
C.4	222	220	◆	222	222	219	377	502.09	1203.57
C.5	217	216	217	217	216	215	245	170.82	1170.05
D.1	◆	◆	◆	◆	◆	60	226	298.90	1346.26
D.2	67	68	68	68	68	66	197	292.20	1430.44
D.3	73	75	74	74	74	72	740	1061.48	1483.24
D.4	63	63	63	◆	63	62	503	642.64	1350.00
D.5	62	◆	◆	◆	◆	61	449	606.65	1369.89

Tabela 6.6

As tabelas abaixo identificam os problemas teste do segundo grupo, os tamanhos das melhores soluções conhecidas e os resultados obtidos por AG1 e AG2, comparados à heurística HPR. Apenas os problemas S_9, S_{15}, S_{27} e S_{45} possuem soluções ótimas conhecidas.

AG1

Problemas	Solução viável					Solução ótima	Geração média	Tempo méd. melhor sol.	Tempo total médio busca genética
S.9	◆	◆	◆	◆	◆	5	0	0	36.48
S15	◆	◆	◆	◆	◆	9	0	0	45.91
S27	◆	◆	◆	◆	◆	18	0	0	134.29
S45	◆	◆	◆	◆	◆	30	17	18.91	567.83
S81	▪	▪	▪	▪	▪	61	20	23.41	865.88
S243	▪	▪	▪	▪	▪	203	241	3206.65	8246.21

Tabela 6.7

◆ - Solução ótima
▪ - Melhor solução conhecida

AG2

Problemas	Solução viável					Solução ótima	Geração média	Tempo médio melhor solução	Tempo total médio busca genética
S.9	◆	◆	◆	◆	◆	5	0	0	41.32
S15	◆	◆	◆	◆	◆	9	0	0	53.57
S27	◆	◆	◆	◆	◆	18	0	0	176.04
S45	◆	◆	◆	◆	◆	30	26	31.49	649.88
S81	▪	▪	▪	▪	▪	61	31	52.75	1433.24
S243	▪	▪	▪	▪	▪	203	316	4867.89	12054.56

Tabela 6.8

Examinando as tabelas de resultados acima podemos observar que:

. Para os problemas do primeiro grupo (4,5,6,A,B,C,D), os algoritmos encontraram em pelo menos uma das execuções, as soluções ótimas de 27 dos 45 problemas.

. Para os problemas do segundo grupo (S), os algoritmos encontraram em todas as execuções as soluções ótimas para os problemas em que estas são conhecidas, e também todas as melhores soluções conhecidas para os demais problemas.

. O maior desvio percentual médio da solução ótima apresentado por uma solução obtida pelos algoritmos em todos os problemas testados nunca ultrapassou 4.16%.

Os resultados obtidos mostraram que nenhum dos algoritmos utilizando os diferentes operadores de cruzamento se apresentou significativamente superior a outro, no que se refere à obtenção de bons limites para a função objetivo original do PCC. Em termos de desvio percentual médio da solução ótima, obtivemos os seguintes valores:

AG1	AG2
0.7175	0.7570

Tabela 6.9

Esses resultados indicam uma relativa independência da heurística baseada em AGs na obtenção de soluções para o PCC, com relação ao operador de cruzamento utilizado.

Comparativamente às demais heurísticas apresentadas nesse trabalho, temos:

HEURÍSTICA	Desvio percentual médio da solução ótima
Balas e Ho	7,416%
SCHEURI	3,311%
SCHEURI1T07	6,428%
Heurística de relaxação Lagrangeana	0,638%
Heurística de relaxação	0,641%

Surrogate	
Heurística baseada em AGs	0,737%

Tabela 6.10

CAPÍTULO 7

Considerações Finais

Esse trabalho compreende uma revisão bibliográfica do Problema de Cobertura de Conjuntos e dos principais métodos exatos e heurísticos conhecidos para sua solução. Também são apresentados os fundamentos de uma abordagem recente aplicada a problemas de otimização combinatória: os algoritmos de busca genética. Os resultados obtidos pelos algoritmos implementados demonstram que a

utilização de uma taxa de mutação adaptativa e de um operador de viabilidade podem ser boas alternativas para melhorar a performance de um AG. O algoritmo mostrou ser capaz de gerar soluções ótimas ou próximas do ótimo para problemas de cobertura de conjuntos de grande porte. As melhores soluções encontradas, mesmo para um número reduzido de execuções, estão sempre a um pequeno percentual em relação às soluções ótimas conhecidas. Em particular, o algoritmo se mostrou bastante eficiente na obtenção de soluções ótimas para os PCCs que surgem no cálculo da dimensão-1 das matrizes de incidência de sistemas de triplas de Steiner. Para as instâncias em que as soluções ótimas são conhecidas, o algoritmo sempre conseguiu encontrá-las. Para os demais casos, sempre foram encontrados as melhores soluções conhecidas. Os resultados provam que os AGs, apesar de serem uma ferramenta de otimização de uso genérico, são ao capazes de obter boas soluções para o PCC.

O tempo computacional dos AGs é relativamente grande, e dificilmente um AG pode se aproximar de um algoritmo de programação matemática mais sofisticado, o qual pode obter uma solução em um número de iterações da ordem do tamanho do problema. Já em uma busca genética, cada geração envolve a construção de um conjunto potencial de soluções para o problema.

Os experimentos realizados permitiram derivar algumas observações interessantes a respeito dos AGs, e, de certa forma, vislumbrar alguns possíveis caminhos para torná-los ainda mais competitivos na solução do problema tratado.

Quanto à eficiência das estruturas de dados utilizadas, algumas observações podem ser feitas. Um esquema de armazenamento mais eficiente poderia ser obtido se fosse mantida apenas uma população com sobreposição. Esse esquema iria implicar na utilização de um operador de seleção mais simples, pois a cada nova solução gerada alteram-se as probabilidades das soluções serem selecionadas para reprodução. O operador de “torneio binário” descrito por Goldberg [28] seria uma

alternativa interessante. Esse operador trabalha tomando duas estruturas escolhidas aleatoriamente e selecionando a estrutura que apresentar o melhor valor para a função de adaptação. Outro método viável seria construir um subconjunto com as estruturas mais bem adaptadas e selecionar uniformemente nesse conjunto uma estrutura para reprodução.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Avis, D. A note on some computationally difficult set covering problems. **Mathematical Programming**, 18: 138-145, 1980.

[2] Balas, E. Cutting planes from conditional bounds: a new approach to set covering. **Mathematical Programming Study**, 12: 19-36, 1980.

[3] Balas, E., Ho, A. Set covering algorithms using cutting planes, heuristics and subgradient optimization: a computational study. **Mathematical Programming**, 12: 37-60, 1980.

[4] Balas, E., Zemel, E. An algorithm for large zero-one knapsack problems. **Operations Research**, 28(5): 1130-1154, 1980.

[5] Balinsk, M.L., Quandt, R.E. On a integer program for a delivery problem. **Operations Research**, 12: 300-304, 1964.

[6] Beasley, J.E. An algorithm for set covering problem. **European Journal of Operational Research**, 31: 85-93, 1987.

[7] Beasley, J.E. A lagrangian heuristic for set covering problems. **Naval Research Logistics**, 37: 145-164, 1990.

[8] Beasley, J.E. OR-Library: Distributing tests problems by eletronic mail . **Journal of Operations Research Society**. London, Imperial College, Management School, 1990.

[9] Beasley, J.E., Chu, P.C. A genetic algorithm for the set covering problem. Working paper. London, Imperial College, Management School, 1994.

[10] Belmore, M., Ratliff, H.D. Set covering and involutary bases. **Management Science**, 18: 194-206, 1971.

[11] Booker, L.B., Goldberg, D.E., Holland, J. H. Classifier systems and genetic algorithm (Technical Rep. No.8). Ann Arbor, University of Michigan, Cognitive Science and Machine Intelligence Laboratory, 1987.

[12] Chvátal, V. A greedy heuristic for the set covering problem. **Mathematics of Operations Research,** 4: 233-235, 1979.

[13] Christofides, N., Korman, S. A computational survey of methods for the set covering problem. **Management Science**, 21(5): 591-599, 1975.

[14] Day, R.H. On optima extracting from a multiple file data storage system: an application of integer programming. **Operations Research**, 13(3): 482-494, 1965.

[15] De Jong, K.A. Analysis of the behavior of a class of genetic adaptive systems. Ph.D. Dissertation. Department of Computer and Communicaton Sciences, University of Michigan, Ann Arbor, MI, 1975.

[16] Dudzinski, K, Walukiewicz, S. Exact methods form the knapsack problem and its generalizations. **European Journal of Operational Research**, 28: 3-21, 1987.

[17] Etcheberry, J. The set covering problem: a new implicit enumeration algorithm. **Operational Research**, 25: 760-772, 1977.

[18] Feo, T.A.,Resende, M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, 8: 67-71, 1989.

[19] Fisher, M.L. The Lagrangian relaxation method of solving integer programming problems. **Management Science**, 27(1): 1-18, 1981.

[20] Fisher, M.L., Rosenwein, M.B. An interactive optimization system for bulk cargo ship scheduling. **Naval Research Logistics Quaterly**, 36: 27-42, 1989.

[21] Fisher, M.L., Kedia, P. Optimal solution of set convering/partitioning problems using dual heuristics. **Management Science**, 36(6): 674-688, 1990.

[22] Fulkerson, D.R., Nemhauser , G.L., Trotter, L.E. Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems. **Mathematical Programming Study**, 2: 72-81, 1974.

[23] Garey. M.R., Johnson, D.S. **Computer and intractability - a guide to the theory of NP-completeness**. San Francisco, W.H. Freeman and Company, 1979.

[24] Garfinkel, R.S., Nemhauser, G.L. The set partitioning problem: set covering with equality constraints. **Operations Research**, 17: 848-856, 1969.

[25] Garfinkel, R.S, Nemhauser, G.L. Optimal political districting by implicit enumeration techniques. **Management Science**, 16: B495-B508, 1970.

[26] Garfinkel, R.S., Nemhauser, G.L. **Integer Programming**. New York, John Wiley, 1972.

[27] Geoffrion, A.M. An improved implicit enumeration approach for integer programming. **Operations Research**, 17: 437-454, 1969.

[28] Goldberg, D.E. **Genetic Algorithms in Search, Optimization, and Machine Learning,** Adison-Wesley, 1989.

[29] Grefenstette, J.J.; Gopal, R.; Rosmaita, B.J.; Gucht, D.V. Genetic algorithms for the Traveling Salesman Problem. **Proceedings of an International Conference on Genetic Algorithms and Their Applications,** 160-168, 1985.

[30] Grefenstette, J.J. Optimization of control parameter for genetic algorithms. **IEEE Transactions on Systems, Man, and Cybernetics**, 16: 122-128, 1986.

[31] Hall, M. Jr. **Combinatorial theory**, Blaisdell Company, Waltham, MA, 1967.

[32] Holland, J.H. **Adaptation in natural and artificial system.** The University of Michigan Press, Ann Arbor, MI, 1975.

[33] Kreshnakumar, K., Goldberg, D.E. Genetic algorithms in control system optimization. **AIAA Guidance, Navigation and Control Conference,** 1568-1577**,** 1991**.**

[34] Lemke, C.E., Salkin, H.M., Spielberg, K. Set covering by single branch enumeration with linear programming subproblems. **Operations Research**, 19: 998-1022, 1971.

[35] Levine, D.M, A genetic algorithm for the set partitioning problem. **Proceedings of the 5th International Conference on GAs**, 481-487, 1993.

[36] Liepins, F.E.; Hilliard, M.R.; Richardson, J.; Palmer, M. Genetic algorithms applications to set covering and traveling salesman problems. **Operations research and artificial intelligence: The integration of problem solving strategies**. D.E. Brown & C.C. White Ed. 29-57, 1990.

[37] Lopes, F.B. Uma nova heurística para problemas de cobertura de conjuntos. **Dissertação de Mestrado**, **INPE-5471-TDI/502**, 1992.

[38] Lorena, L.A.N., Plateau, G. A monotone decreasing algorithm for the 0-1 multiknapsack dual problem. (Raport de recherche L.I.P.N. 89-1) Paris, Université Paris-Nord, 1988

[39] Marsten, R.F. Muller, M.R., Killion, C. L. Crew planning at Flying Tiger: a sucessfull application of integer programming. **Management Science**, 25(12): 1175-1183, 1979.

[40] Marsten, R.F.; Shepardson, F. Exact solution of crew scheduling problems using the set partition model: recent successful applications. **Networks**, 11(2): 167-179, 1981.

[41] Quine, W.V. A way to simplify truth functions. **American Mathematical Monthly**, 62: 627-631, 1955.

[42] Revelle, C.D.; Marks, D; Liebman, J.C. An analysis of private and public sector facilities location models. **Management Science** 16(12): 692-707, 1970.

[43] Salverson, M.E. The assembly line balancing problem. **Journal of Industrial Engineering**, 6:18-25, 1955.

[44] Salkin, H.M., Koncal, R.D. Set covering by an all-integer algorithm computational experience. **Journal of Association of Computational Machinery**, 20: 189-193, 1973.

[45] Schafer, J.D, Caruana, R.A., Eshelman, K.,J. e Das, R. A study of control parameter affecting online performance of genetic algorithms for function optimization. **Proceedings of the 3th International Conference on Gas**, 51-60, 1989.

[46] Tam, K,Y. Genetic algorithm, function optimization, and facility layout design. **European Journal of Operational Research**, 63: 322-340, 1992.

[47] Torregas, C.; Revelle, C., Bergman, L. The location of emergency service facilities. **Operations Research**, 19: 1363-1373, 1970.

[48] Vasko, F.J., Wilson, G.R. An efficient heuristic for large set covering problems. **Naval Research Logistics Quaterly**, 31: 163-171, 1984.

[49] Vasko, F.J., Wilson, G R. Hybrid heuristics for Minimum Cardinality Set Covering Problems. **Naval Research Logistics Quaterly**, 33, 1986.

[50] Vignaux, G.A., Michalewicz, Z. A genetic algorithm for the linear transportation problem. **IEEE Transactions on System, Man, and Cybernetics**, 21(2): 445-452, 1991.

[51] Walker, W. Application of the set covering problems to the assignment of ladder trucks to fire houses. **Operations Research**, 22: 275-277, 1974.