UM ALGORITMO GENÉTICO APLICADO AO PROBLEMA DE EMPACOTAMENTO DE BINS TRIDIMENSIONAIS

José Lassance de Castro Silva

Nei Yoshihiro Soma

Departamento de Computação, Instituto Tecnológico de Aeronáutica
12228-900, São José dos Campos-SP, e-mail: {lassance, nysoma}@comp.ita.cta.br

Abstract

The objective of the present work is to present one technique to solver combinatorial optimization problems, that involves permutation, in an acceptable computational time based on genetic algorithm. This technique partitions the set of viable solutions in small regions, distinct between itself, in order to diversify the local search. We apply the technique to the Three-Dimensional Bin Packing Problem. The problem is NP-hard in the strong sense and extremely difficult to be solved in the practice. Extensive computational experiments are reported for instances with up to 90 items, and the results are compared with those obtained from the literature.

Key words: *Combinatorial Optimization Problem, Heuristic, Permutation.*

1 – Introdução

Neste trabalho, nosso principal objetivo é apresentar uma técnica que encontre boas soluções para o problema de empacotamento de bins tridimensionais, denotado na literatura como Three-Dimensional Bin Packing Problem(*3D-BPP*), através da meta-heurística algoritmo genético.

No 3D-BPP são dados:

- um conjunto de n itens (caixas retangulares), cada um desses itens sendo caracterizado pela largura w_j, altura h_j e comprimento d_j, referenciado por uma tripla ordenada $\{w_j,h_j,d_j\}$, ou seja, *e.g.* $\{w_j,h_j,d_j\} \neq \{h_j,w_j,d_j\}$ (rotações implicam em itens distintos), para todo $j \in J = \{1, 2, ..., n\}$; e
- um número limitado de bins (conteineres ou caixas retangulares maiores), onde inicialmente há uma disponibilidade de pelo menos n bins idênticos e tridimensionais tendo como dimensões comuns: largura W, altura H e comprimento D.

Também, supomos, sem perda de generalidade, que todos os dados de entrada são números inteiros positivos, satisfazendo às relações: $w_j \leq W$, $h_j \leq H$ e $d_j \leq D$, para todo $j \in J$. O 3D-BPP consiste em se empacotar ortogonalmente todos os n itens utilizando o menor número possível de bins.

Uma motivação para o estudo do 3D-BPP vem do fato que ele modela uma grande quantidade de aplicações industriais, as quais variam do carregamento de cargas em aviões,

ao seqüenciamento (escalonamento) de tarefas em ambientes multi-processados. Sob o ponto de vista teórico, o problema é uma variação do bem conhecido e já clássico problema *Bin Packing* (BPP), *cf*. Garey e Johnson [1], o que implica, *a fortiori*, ser ele, NP-difícil no sentido forte, *cf*. Martello *et al.* [2]. Ressalve-se que, contrariamente ao *Bin Packing* usual (uni-dimensional), não se conhece heurística alguma, *s.m.j.*, para o 3D-BPP com garantia de desempenho, *e. g.* para o BPP, a bem conhecida heurística FFD implica em um erro assintótico de 11/9 e um tempo de processamento limitado por O(n.*log* n).

Verificamos que o 3D-BPP começa a ser pesquisado, mais intensivamente, há pouco mais de dez anos. Como especificado por Dowsland [3], há um número restrito de trabalhos publicados relatando tais problemas de empacotamentos tridimensionais. A primeira heurística apresentada para o 3D-BPP aparece em Scheithauer [4], enquanto que em Chen *et al.* [5] tem-se uma generalização do problema, onde os bins podem ter diversos tamanhos, tendo sido o problema, modelado *via* Programação Linear Inteira Mista.

Ainda em termos heurísticos, Pisinger *et al.* [6], adaptaram para o 3D-BPP uma nova metodologia de aproximação para problemas da otimização combinatória, denominada Busca Local Direcionada GLS (Guided Local Search). Em Martello *et al.* [2] tem-se o primeiro algoritmo exato para o problema (3D-BPP), assim como para o do Carregamento de Conteineres (Container Loading Problem), para os quais uma abordagem branch-and-bound foi utilizada. Naquele mesmo artigo, duas heurísticas são propostas (H_1 e H_2), as quais serão comentadas com mais detalhes adiante, já que uma delas (H_1), é base de comparação para a heurística aqui sugerida. Em todos esses artigos não há menção alguma à estabilidade dos itens empacotados dentro dos bins.

O ataque ao 3D-BPP sugerido aqui baseia-se em uma evolução do algoritmo genético tradicional. Desenvolvemos um procedimento distinto daqueles utilizados nos algoritmos genéticos, onde a população é construída de modo a diversificar a busca no conjunto de soluções viáveis.

A organização deste trabalho segue à: Na Seção 2, apresentaremos as idéias básicas do nosso *Algoritmo Genético* utilizada para a resolução do 3D-BPP. Na seção 3, apresentaremos os resultados obtidos nos experimentos computacionais. As conclusões estão apresentadas na Seção 4, enquanto a bibliografia será apresentada na Seção 5.

2 – Algoritmo Genético (AG)

Os algoritmos genéticos, criados por Holland [7] baseiam-se nos processos observados na evolução natural das espécies. Assim, um algoritmo genético parte de uma população de indivíduos (configuração inicial de um problema), faz a avaliação de cada um (aplicação da função objetivo), seleciona os melhores e promove manipulações genéticas, como cruzamento e mutação (correspondente às perturbações ou movimentos) a fim de criar uma nova população.

O nosso *algoritmo genético*, denominado aqui por AG, é baseado na teoria dos algoritmos genéticos tradicionais. Porém, AG trabalha com uma população, inicial e posterior, que é gerada pela heurística permutacional HP (utilizada para resolver problemas de otimização combinatória permutacional, inclusive o 3D-BPP), desenvolvida por Silva e Soma [8], que tem a finalidade de diversificar a procura por boas soluções dentro do espaço de soluções viáveis. Além disso, AG também usa a heurística gulosa,

denominada HV, desenvolvida por Silva *et al.* [9], que funciona como se fosse o operador mutação, ela altera a ordem dos itens na solução, podendo melhorá-la ou não, intensificando a busca por uma boa solução. A heurística HV foi desenvolvida para ser usada exclusivamente na resolução do 3D-BPP, podendo também servir de base para a construção do operador mutação para outros problemas de otimização combinatória permutacional. O objetivo básico de usarmos estes dois procedimentos é evitar uma convergência prematura do algoritmo.

Para o preenchimento de um bin é necessário que haja uma política que determine a ocupação do espaço interno dos bins. Neste sentido, diremos que uma dada disposição espacial formada por um conjunto de itens já empacotados (E) no bin, definirá duas regiões no mesmo: uma região chamada de "*envelope*", associado ao empacotamento daqueles itens de E, e outra região "*vazia*", na qual os itens ainda não alocados (Φ) podem, eventualmente, ser fisicamente alocados naquele bin. Tanto a região "vazia" dentro de um bin, quanto seu complemento, podem ser determinados como funções de alguns pontos extremos nos quais itens podem ser inseridos, *3D-Corner*, (*ponto de canto*); sendo esta definição, uma extensão do caso bi-dimensional. Um *ponto de canto* (PC) de um dado bin, seja ele p, indica que nenhum item $j \in \Phi$ poderá ter alguma interseção à direita de p, ou acima de p, ou defronte a p, em relação ao conjunto E no bin em questão, caso j houvesse que ser inserido em p. A representação de p é feita usando-se o sistema de coordenadas cartesianas (x_p, y_p, z_p), com a origem sendo definida no canto inferior esquerdo mais ao fundo do bin. Note-se que caso um item $j \in \Phi$ não possa ser alocado fisicamente em p, pelo menos uma das condições, $x_p+w_j \leq W$ ou $y_p+h_j \leq H$, ou $z_p+d_j \leq D$ deve ser violada.

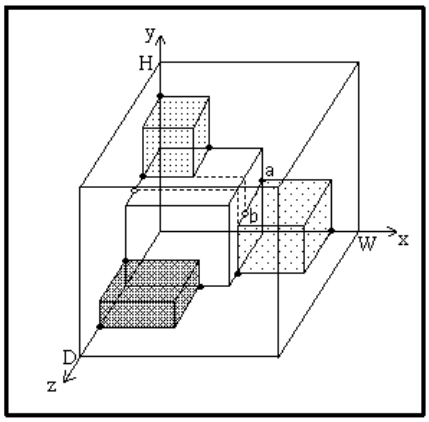

Figura 1 – Itens já empacotados com seus PC's (círculos negros maiores)

Na Figura 1 tem-se um conjunto de itens já alocados e os respectivos pontos de cantos (círculos pretos maiores). No processo de determinação dos PC's alguns pontos de cantos "falsos" (pontos brancos, Figura 1) podem ser gerados, os quais são facilmente detectados e removidos utilizando-se a dominância de pontos. Por exemplo, os pontos a e b, representados na mesma figura e, respectivamente, com coordenadas (x_a, y_a, z_a) e (x_b, y_b, z_b) mostram que somente a é um PC, pois a está "*escondido*" atrás de b. Note-se que $x_a=x_b$, $y_a=y_b$, $z_b>z_a$ (critério de dominância). Tanto HP quanto HV utilizam o procedimento 3D-Corner para alocar um item dentro de um bin.

A Figura 2, descreve o algoritmo genético tradicional, onde o *processo de adaptação* significa que os indivíduos evoluem independentemente, enquanto que o *processo de cooperação* implica uma troca de informações entre os indivíduos. Muitos algoritmos genéticos podem ser descritos dentro dessa estrutura. A *seleção* e o *cruzamento* de

indivíduos podem ser vistos como o processo de cooperação, enquanto o *operador mutação* é parte do processo de adaptação.

Generation an *initial population* of individuals;
While (*stop_condition*≠ *TRUE*) **do**
    Cooperation;
    Adaptation;
End_While.

Figura 2 – Esquema geral de um algoritmo genético tradicional.

A seguir descrevemos as etapas e dicas que podem ser seguidas para a construção de um algoritmo genético:

- Escolher a forma de *representação dos cromossomo* (indivíduos). Reeves [10] dá um estudo detalhado para esta representação.
- *População inicial*. Goldberg[11] sugere o tamanho e a construção da população;
- *Avaliação da população*, aplicação da função objetivo do problema;
- *Reprodução* dos melhores indivíduos (*seleção*). Viana[12] descreve algumas técnicas para a seleção dos indivíduos que farão o cruzamento afim de gerar a nova população.
- *Cruzamento (crossover)*, é a operação que possibilita a recombinação das estruturas genéticas da população. O operador crossover troca parte dos padrões genéticos de dois ou mais indivíduos da população, selecionados na reprodução, para formar novos indivíduos. Existem muitos trabalhos, na literatura, que fazem alusão ao operador crossover, podemos citar: Reeves [13] e Davis [14], entre outros;
- *Mutação*, esta operação corresponde a uma pequena perturbação numa configuração que tem como objetivo tentar regenerar (viabilizar) ou melhorar algum indivíduo da população.

Para a construção de AG, aplicado no 3D-BPP, definimos:

- O *cromossomo* (indivíduo da população) como sendo uma permutação dos *n* itens, onde cada item representa um gene do cromossomo. A ordem em que os itens se encontram no cromossomo será a mesma em que eles serão empacotados;
- Os indivíduos da população serão avaliados pelo procedimento 3D-Corner;
- A população de AG tem tamanho fixo igual a 52 indivíduos, dividida em quatro grupos, de 13 indivíduos cada. AG usa uma população diferente em cada iteração do algoritmo, sendo no total *n* populações distintas P_1, P_2, ..., P_n. Todos os indivíduos da população P_j , com $1 \le j \le n$, têm o item j como primeiro elemento do cromossomo. Sejam P_{j1}, P_{j2}, P_{j3} e P_{j4} os grupos da população j. Realizamos alguns testes para determinar o tamanho ideal da nossa população e concluímos que para valores menores que 52, múltiplos de quatro, o desempenho na qualidade das soluções apresentadas não foi muito bom, enquanto que para valores acima de 52 não houve melhoras significativas, comprometendo os recursos computacionais disponíveis, tanto em tempo quanto em memória;
- A reprodução será feita com o cruzamento de cada indivíduo do grupo P_{j1} com todos os indivíduos do grupo P_{j2}, idem para P_{j3} com P_{j4};
- O nosso operador crossover trabalha da seguinte forma. Sejam *P1* e *P2* dois vetores com *n* componentes cada, representando os cromossomos de dois indivíduos da população. Primeiro, determinamos o valor absoluto da diferença dos vetores *P1* com *P2*, e colocamos nos vetores *O1* e *O2* (*O1*=*O2*=|*P1* – *P2*|). Substituímos por *n* os valores das componentes de *O1* e *O2* que são iguais a zero. Depois, definimos mais dois vetores *R1* e *R2* com *n* componentes cada um tais que *R1*(j) e *R2*(j), $1 \le j \le n$,

representam a quantidade de vezes que o item j aparece nos vetores *O1* e *O2*, respectivamente. Para cada j=1, 2, ..., *n*, tal que *R1*(j)≥2, substituímos o primeiro item j encontrado em *O1* pela primeira componente (item) i de *R1* tal que *R1*(i)=0, i=1, 2, ..., n. Consequentemente atualizamos o vetor *R1* fazendo *R1*(i)=1 e *R1*(j)=*R1*(j)-1. Este mesmo procedimento também se aplica a *O2* e *R2*, porém a substituição de j por i dar-se-á no último item j encontrado em *O2* . Finalmente, fazemos com que o item k+1 ocupe a primeira posição tanto de *O1* quanto de *O2*, onde k representa o índice da população a que pertencem os indivíduos *P1* e *P2* (k=*P1*(1)=*P2*(1)), se k=*n* substitua k+1 por 1. Assim, *O1* e *O2* representam os cromossomos dos dois indivíduos gerados pelo cruzamento de *P1* e *P2*, e que ambos não pertencem a população de seus pais. Um exemplo da aplicação do operador crossover está ilustrado pela Figura 3, com *n*=7;

s = 2
P1 = [2 1 3 5 4 6 7]
P2 = [2 5 7 6 3 1 4]
O1 = O2 = Abs(P1 – P2) = [7 4 4 1 1 5 3]
R1 = R2 = [2 0 1 2 1 0 1]

⇒

O1 = [3 6 4 2 1 5 7]
O2 = [3 4 6 1 2 5 7]

Figura 3 – Aplicação do operador crossover, para n=7.

- Após realizarmos todos os cruzamento, para uma determinada população, selecionamos o cromossomo do indivíduo que apresentar o menor valor na avaliação, afim de fazermos uma perturbação na ordem em que se encontram os genes (itens) desse cromossomo, através da heurística do volume HV, com o intuito de melhorarmos sua avaliação. Este procedimento representa o operador *mutação* de AG, ele intensifica a busca por soluções melhores partindo de uma solução dada. Aplicamos no máximo *n* vezes este operador na execução do programa.

O algoritmo pára o processo evolutivo e consequentemente a busca, quando algum dos seguintes critérios é satisfeito:

- A solução do problema é igual ao Limite Inferior L_2, desenvolvido por Martello *et al.* [2], para o problema; ou
- Quando todas as *n* populações, com seus devidos cruzamentos, foram avaliadas;

Alguns passos de AG são notavelmente diferentes do algoritmo genético tradicional, tais como:

- A população não é gerada aleatoriamente (evitando a repetição de indivíduos) e a cada iteração uma nova população é gerada totalmente distinta das demais;
- O operador crossover não leva muito em consideração a preservação das características dos pais. O intuito disto, é diversificar a busca no conjunto de soluções viáveis com base na avaliação realizada através dos experimentos genéticos, na prática, tais como os que foram descritos nos trabalhos de Ingvarsson [15], Bentota *et al.* [16], Kahi *et al.* [17], Hancock [18], entre outros, que comprovaram que os melhores resultados (indivíduos com características geneticamente perfeita) não foram alcançados com a mutação e sim com bastante cruzamento de diferentes tipos de indivíduos (diversificação). Segundo esses experimentos, a mutação tem um custo operacional muito alto para ser realizada, ao contrário da diversificação no cruzamento, que tem um custo baixo. Além disso, a diversidade dos filhos obtidos nos cruzamentos feitos com diferentes tipos de experimentos tais como ovelhas, ratos e arrozes, proporcionou a genética um estudo mais detalhado sobre o comportamento e as características que os diferentes indivíduos apresentaram, durante o experimento;

- O operador mutação intensifica a busca por uma solução de boa qualidade a partir de uma solução dada .

Considerando η o número de gerações, μ o tamanho da população, n o número de genes em cada cromossomo (instância do problema) e $O(n^4)$ a ordem de complexidade do operador mutação; temos que a ordem de complexidade do algoritmo AG será $O(\eta.\mu.n.O(n^4)) = O(n^5)$, no pior caso, pois μ=52 e η=2×2×13×13=679.

3. Experimentos Computacionais

Um grande número de experimentos computacionais foram realizados, para uma melhor e mais fidedigna avaliação da técnica proposta. Comparamos AG com as heurísticas H_1, HP e HV. H_1 é baseada no princípio da construção de camadas, conhecida como o método das prateleiras bi-ortogonais.

As classes de testes foram definidas como em Berkey e Wang[19] e Martello *et al.* [2]. As seguintes instâncias foram consideradas para problemas com uma quantidade de itens variando de 10 à 90.

- **Classe 1:** A maioria dos itens são muitos altos e profundos.
- **Classe 2:** A maioria dos itens são muitos largos e profundos.
- **Classe 3:** A maioria dos itens são muitos altos e largos.
- **Classe 4:** A maioria dos itens têm grandes dimensões.
- **Classe 5:** A maioria dos itens têm pequenas dimensões.
- **Classe 6:** Instâncias geradas aleatoriamente em um intervalo pequeno.
- **Classe 7:** Instâncias geradas aleatoriamente em um intervalo médio.
- **Classe 8:** Instâncias geradas aleatoriamente em um intervalo grande.
- **Classe 9:** Instâncias tendo uma solução conhecida com três bins.

A amostra considerada para cada uma das classes tem tamanho 10 (para cada um dos valores de n). Os algoritmos: H_1, AG, HV e HP, foram executados em um Pentium III (600 MHz com 128 Mb de RAM) e o código foi implementado em Ansi C.

A Tabela 1 apresenta o desvio médio em relação ao limite inferior L_2. O desvio é definido como $(Z - L_2)/L_2$, onde Z é o número mínimo de bins encontrado pelos algoritmos para empacotar todos os itens, em cada uma das instâncias.

Verificamos pela Tabela 1, que AG e HP apresentaram os melhores resultados, tendo AG alcançado em média 47,3%, 44,5%, 48,1%, 16,9%, 49,3%, 51,7%, 48,2%, 52,5% e 52,5% dos resultados encontrados por H_1, para as classes de 1 à 9, respectivamente. Isto representa um índice médio de desvio de 45,6% dos resultados alcançados por AG em relação a H_1, que comprova que AG produz soluções bem melhores que H_1. AG utilizou menos da metade dos bins utilizados por H_1, para empacotar todos os itens, em média. Já HP e HV utilizaram menos de 65% dos bins utilizados por H_1.

Quanto ao fator tempo de execução constatamos que H_1 e HV são bastante rápidas. H_1 levou em média menos de 1 segundo para apresentar a solução, enquanto HV levou menos de 10 segundos. Já AG e HP levaram em média 24,4 e 43,6 segundos, respectivamente, porém o bom desempenho de AG está na qualidade das suas soluções apresentadas para o problema. Com isso, verificamos que o tempo computacional consumido por AG e HP é

bem maior que o tempo gasto por H_1 e HV, que já era esperado, pois ambas as técnicas são da ordem de $O(n^5)$, enquanto H_1 e HV são da ordem de $O(n^2)$ e $O(n^4)$, respectivamente.

Classe		10	15	20	25	30	35	40	45	50	60	70	80	90	TOTAL
	H1	0,24	0,33	0,32	0,30	0,31	0,32	0,37	0,30	0,33	0,36	0,30	0,25	0,22	3,95
	HP	0,08	0,24	0,19	0,15	0,17	0,16	0,22	0,20	0,22	0,23	0,21	0,20	0,17	2,44
1	AG	0,05	0,10	0,13	0,10	0,09	0,15	0,18	0,15	0,21	0,20	0,19	0,17	0,15	1,87
	HV	0,11	0,24	0,23	0,23	0,22	0,18	0,21	0,20	0,23	0,23	0,20	0,17	0,14	2,59
	H1	0,30	0,49	0,22	0,42	0,41	0,26	0,30	0,30	0,37	0,24	0,27	0,24	0,22	4,04
	HP	0,15	0,29	0,17	0,28	0,23	0,17	0,21	0,19	0,22	0,16	0,19	0,16	0,19	2,61
2	AG	0,05	0,13	0,08	0,14	0,14	0,12	0,17	0,16	0,19	0,14	0,17	0,14	0,17	1,80
	HV	0,15	0,31	0,19	0,28	0,25	0,18	0,19	0,19	0,23	0,16	0,18	0,14	0,15	2,60
	H1	0,41	0,53	0,39	0,28	0,35	0,35	0,33	0,29	0,35	0,27	0,28	0,23	0,22	4,28
	HP	0,36	0,38	0,27	0,17	0,26	0,22	0,21	0,19	0,24	0,18	0,21	0,18	0,17	3,04
3	AG	0,05	0,12	0,16	0,14	0,20	0,16	0,18	0,18	0,22	0,15	0,19	0,15	0,16	2,06
	HV	0,36	0,40	0,32	0,19	0,26	0,23	0,23	0,20	0,25	0,15	0,20	0,17	0,15	3,11
	H1	0,03	0,03	0,03	0,05	0,05	0,04	0,04	0,05	0,04	0,03	0,06	0,03	0,05	0,53
	HP	0,03	0,01	0,01	0,02	0,04	0,03	0,02	0,03	0,03	0,02	0,03	0,03	0,03	0,33
4	AG	0,00	0,00	0,00	0,00	0,01	0,01	0,00	0,01	0,01	0,01	0,02	0,01	0,01	0,09
	HV	0,03	0,01	0,01	0,02	0,05	0,03	0,02	0,03	0,03	0,02	0,03	0,03	0,03	0,34
	H1	0,45	0,71	0,51	0,56	0,47	0,44	0,68	0,67	0,65	0,67	0,47	0,48	0,37	7,13
	HP	0,30	0,42	0,27	0,36	0,30	0,26	0,43	0,42	0,40	0,43	0,34	0,37	0,30	4,60
5	AG	0,10	0,12	0,16	0,23	0,20	0,22	0,36	0,37	0,42	0,40	0,33	0,35	0,26	3,52
	HV	0,30	0,47	0,30	0,34	0,30	0,26	0,43	0,42	0,42	0,42	0,34	0,34	0,25	4,59
	H1	0,47	0,47	0,32	0,32	0,36	0,39	0,30	0,31	0,37	0,30	0,31	0,25	0,22	4,39
	HP	0,28	0,29	0,17	0,16	0,20	0,22	0,17	0,22	0,24	0,24	0,25	0,22	0,19	2,85
6	AG	0,13	0,16	0,05	0,10	0,15	0,20	0,15	0,23	0,23	0,21	0,24	0,22	0,20	2,27
	HV	0,28	0,32	0,17	0,20	0,20	0,22	0,15	0,18	0,21	0,17	0,17	0,16	0,13	2,56
	H1	0,83	0,61	0,62	0,69	1,07	0,78	0,81	0,82	0,60	0,71	0,79	0,55	0,52	9,40
	HP	0,43	0,14	0,20	0,28	0,47	0,42	0,44	0,46	0,43	0,46	0,56	0,43	0,41	5,13
7	AG	0,15	0,05	0,17	0,22	0,47	0,41	0,34	0,46	0,40	0,44	0,56	0,46	0,41	4,54
	HV	0,48	0,39	0,23	0,33	0,56	0,47	0,43	0,51	0,39	0,43	0,50	0,37	0,33	5,42
	H1	0,68	0,55	0,33	0,58	0,58	0,44	0,70	0,60	0,57	0,38	0,40	0,42	0,47	6,70
	HP	0,35	0,25	0,21	0,29	0,32	0,22	0,46	0,40	0,39	0,26	0,33	0,36	0,40	4,24
8	AG	0,15	0,18	0,07	0,20	0,27	0,22	0,38	0,39	0,39	0,25	0,32	0,33	0,37	3,52
	HV	0,35	0,25	0,21	0,29	0,30	0,26	0,44	0,34	0,39	0,19	0,28	0,29	0,35	3,94
	H1	0,23	0,67	0,50	1,07	0,82	1,42	1,25	1,18	1,60	2,40	1,70	1,90	2,70	17,44
	HP	0,05	0,22	0,28	0,53	0,40	0,67	0,53	0,57	0,67	1,68	0,72	0,80	1,67	8,79
9	AG	0,05	0,32	0,32	0,52	0,40	0,67	0,53	0,70	0,80	1,58	0,82	0,78	1,67	9,16
	HV	0,05	0,47	0,25	0,68	0,43	0,72	0,53	0,62	0,75	1,38	0,75	0,92	1,62	9,17

Tabela 1 – Desvio médio de L_2 com respeito ao valor Z da solução encontrada.

4. Conclusões

Nossos experimentos computacionais baseados em problemas da literatura, indicam que AG pode ser aplicada ao 3D-BPP com sucesso, gerando boas soluções. A presente meta-heurística faz parte de um trabalho maior, onde se está estudando o problema da estabilidade dos empacotamentos e, que *s.m.j.*, é inédito na literatura. AG também pode ser aplicada na resolução de outros problemas de otimização combinatória permutacional.

5. Bibliografia

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco,1979.

[2] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, march-april, 2000.

[3] W. B. Dowsland. Three-dimensional packing solution approaches and heuristics development. *International Journal of Production Research*, 29(8): 1673-1685, 1991.

[4] G. Scheithauer. A three-dimensional bin packing algorithm. *J. Inform. Process. Cybernet.*, 27:263-271, 1991.

[5] C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80:68-76, 1995.

[6] D. Pisinger, O. Faroe and M. Zachariasen. Guided Local Search for the three-dimensional bin packing problem. *Techinical Report*. University of Copenhagen, Denmark, 13, 1999.

[7] J. H. Holland. *Adaptation in natural artificial systems*. University of Michigan Press. 1975.

[8] J. L. C. Silva, N. Y. Soma e Nelson Maculan. Uma heurística para Problemas de Otimização Combinatória Permutacional. Anais do *XXXIII SBPO*, Campos do Jordão-SP, Brazil,2001.

[9] J. L. C. Silva e N. Y. Soma. Uma heurística para problemas de empacotamento de bins tridimensionais. *Anais do Encontro Nacional de Engenharia de Produção*, Salvador-BA, Brazil, 2001.

[10] C. Reeves (Ed.). *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, London, 1995.

[11] D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann, Los Altos, CA, 70-79, 1989.

[12] Valdisio Viana. Meta-heurísticas e Programação Paralela em Otimização Combinatória. Edições UFC, Fortaleza-CE, Brasil, 1998.

[13] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Ops. Res.*, (in review), 1992.

[14] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[15] Ingvarsson, P. K. Restoration of genetic variation lost - the genetic rescue hypothesis. *TRENDS in Ecology & Evolution*, Vol.16, 2: 62-63, 2001.

[16] Bentota, A. P. , Senadhira, D. , Lawrence, M. J. Quantitative genetics of rice: The potential of a pair of new plant type crosses. *Field Crops Research*, 55: 267-273, 1998.

[17] Kahi, A. K. *et al.* Economic evaluation of crossbreeding for dairy production in a pasture based production system in Kenya. *Livestock Production Science*, 65: 167-184, 2000.

[18] Hancock, J. F. Implications for plant evolutionary ecology. *TRENDS in Plant Science*, Vol. 6, 4: 185, 2001.

[19] J. O. Berkey and P. Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423-429, 1987.