CNMAI Congresso Nacional de Matemática Aplicada à Indústria 2014

18 a 21 de novembro de 2014, Caldas Novas - Goiás

O PROBLEMA DA MOCHILA TRIDIMENSIONAL RESOLVIDO POR UMA HEURÍSTICA

Mirella Augusta Sousa Moura, mirella.asm14@hotmail.com[1]
Thiago Alves de Queiroz, taq@.ufg.br[1]

[1] Departamento de Matemática – Universidade Federal de Goiás/Regional Catalão,
Av. Dr. Lamartine Pinto de Avelar, 1120, Setor Universitário, 75704-020, Catalão-GO, Brasil.

Resumo: *Problemas de empacotamento têm aparecido constantemente nas indústrias, com aplicações principalmente nas cadeias de suprimento e logística. Esta pesquisa lida com o problema da mochila ilimitado em sua versão tridimensional com o objetivo de maximizar a ocupação do recipiente. No problema em questão não há um limite quanto ao número de cópias de cada item que podem ser empacotados, porém os itens devem ser organizados sem sobreposição e respeitando as dimensões do recipiente. Apresenta-se uma heurística baseada na técnica de Recozimento Simulado, que busca diminuir o desperdício selecionando o item com a melhor aptidão para ser colocado em espaços vazios. A linguagem de programação C foi usada para a codificação dos algoritmos, em que testes computacionais usando instâncias da literatura mostraram resultados satisfatórios.*

Palavras-chave*: Problema da Mochila; Mochila Irrestrita; Empacotamento Tridimensional; Recozimento Simulado.*

1. INTRODUÇÃO

Organizações buscam, por meio do lucro financeiro, obter o maior retorno possível para seus investimentos. Por esse motivo, elas precisam tomar decisões para minimizar seus desperdícios e/ou aumentar seus rendimentos. A Pesquisa Operacional (PO) tem sido usada em diferentes áreas para resolver problemas que conduzam e coordenam atividades, como transportes, manufatura, construção, telecomunicações, planejamento financeiro, assistência médica, militar e serviços públicos (HILLIER; LIEBERMAN, 2010). A PO está diretamente ligada com a tomada de decisões em empresas e organizações (ARENALES *et al.*, 2007).

No âmbito da Pesquisa Operacional encontram-se os Problemas de Empacotamento, os quais consistem em colocar objetos menores, chamados de itens, dentro de objetos maiores, chamados de recipientes. Existe uma variedade de aplicações para problemas de empacotamento, conforme as diferentes necessidades: logística, cadeia de suprimentos, organização pessoal, investimento financeiro, alocação de recursos, etc. (QUEIROZ, 2010).

Dentro os problemas de empacotamento, o problema da Mochila é um dos problemas que tem chamado a atenção. Para este problema é desejado empacotar um subconjunto de itens que maximize o valor total empacotado. Na versão irrestrita não existe limite quanto ao número de cópias de um item que pode ser empacotado. O problema da mochila é computacionalmente difícil, ao mesmo tempo que possui muitas aplicações na indústria, como no carregamento de paletes e contêineres, corte de tecido, de placas de madeira e aço, de bobinas de papel e alumínio, etc.

Resolver o problema da mochila é uma tarefa computacionalmente difícil, pois uma instância com apenas 20 itens pode ter mais de 1 milhão de soluções. Claramente, deseja-se a melhor solução (a ótima) dentro deste conjunto. Então, enumerar cada possível solução, para depois obter a solução ótima, não parece ser uma forma viável de resolver esse problema. Vale mencionar que o problema da mochila na sua versão irrestrita está na classe *NP*-Difícil,

tal que, supondo $P \neq NP$, não se conhecem algoritmos exatos de tempo polinomial para resolvê-lo. A dificuldade aumenta ainda mais quando restrições práticas são adicionadas ao problema (QUEIROZ, 2010).

O presente trabalho considera o problema da mochila irrestrita (PMI) em sua versão tridimensional (3D). Parte-se das ideias propostas por Leung *et al.* (2012), que desenvolveram heurísticas simples e eficientes para o problema da mochila bidimensional (2D), para desenvolver heurísticas para a variante do PMI em estudo. Com isso, os algoritmos desenvolvidos são codificados na linguagem de programação C e vários testes executados em instâncias obtidas de repositórios da Internet. A próxima seção apresenta uma revisão da literatura, enquanto a Seção 3 define formalmente o problema e traz a heurística proposta. Os testes computacionais são apresentado na Seção 4, enquanto conclusões e direções para trabalhos futuros estão na Seção 5.

2. REVISÃO DA LITERATURA

Alguns algoritmos foram desenvolvidos no passado para resolver o PMI. Na versão 3D, há uma variante chamada de problema de carregamento de contêineres. Ele é formulado com um modelo de programação inteira mista por Chen *et al.* (1995). Os autores consideraram diferentes tipos de contêineres, vários tamanhos de caixas e orientações para cada caixa. O modelo foi estendido para formular alguns casos especiais também. Bischoff e Ratcliff (1995) apresentaram duas abordagens, enquanto a análise de desempenho é feita por meio de problemas teste publicados e outros gerados aleatoriamente. Os testes mostraram que as duas abordagens, quando combinadas, fornecem uma ferramenta poderosa e versátil. Esse problema também foi investigado por Gehring e Bortfeldt (1997) por meio de um algoritmo genético (GA). As principais ideias do GA está em gerar um conjunto de torres de caixas e depois organizar tais torres no chão do recipiente de acordo com um determinado critério de otimização. Neste contexto, o problema poderia incluir diferentes restrições práticas.

É proposta uma heurística para a versão 3D por Pisinger (2002) com base em uma abordagem de construção de paredes que decompõe o problema em várias camadas. As camadas são divididas em um número de tiras, de forma que cada tira é formulada e resolvida como um problema da mochila com capacidade igual a largura ou a altura do recipiente. Várias regras de classificação para a seleção da camada mais profunda e mais promissora da faixa são apresentadas e o desempenho dos algoritmos correspondentes é comparado experimentalmente para instâncias homogêneas e heterogêneas. A melhor regra de classificação é então utilizada em um estudo computacional abrangente envolvendo instâncias de grande porte. Uma heurística gulosa foi apresentada por Eley (2002). As soluções fornecidas pela heurística gulosa são melhoradas através de uma busca em árvore. Aspectos adicionais, tais como estabilidade de carga e distribuição de peso dentro do recipiente também são levados em conta. Bortfeldt *et al.* (2003) apresentaram um algoritmo de busca tabu para uma variante do problema de carregamento do contêineres. Buscas são realizadas e organizadas por um algoritmo de busca tabu paralelizado. A eficácia do algoritmo foi demonstrada por um teste comparativo em instâncias da literatura.

Abordagens mais recentes trabalham sobre formulações já existentes, como feito por Junqueira (2009) para o caso 3D, que manipula a formulação de Beasley (1985) do caso 2D, que diz que as possíveis posições onde os itens podem ser colocados dentro do recipiente podem ser definidas por conjuntos distintos ao longo das dimensões do recipiente. Junqueira *et al*. (2010) desenvolveram modelos de programação inteira 0-1 para a versão tridimensional do problema da mochila restrita, considerando restrições práticas de estabilidade e de empilhamento de itens.

Modelos matemáticos para o problema da mochila e suas variantes 2D e 3D têm sido propostos na literatura, no entanto, há uma carência de estudos que consideram restrições realistas que surgem na prática. Junqueira *et al.* (2012) apresentaram modelos de programação inteira mista para o problema de carregamento do contêineres que considera a estabilidade vertical e horizontal da carga e a força da carga, incluindo fragilidade. Baldi *et al.* (2012) resolveram o problema da mochila tridimensional com restrições de balanceamento de carga, que pede por restrições adicionais relacionadas com o centro de massa dos itens. Os itens não devem se sobrepor, mas podem ser rotacionados e os testes mostraram que a heurística supera a qualidade das soluções disponíveis na literatura.

A heurística apresentada por Andrew e Hong (2013) foi desenvolvida para o problema de carregamento em um único contêiner com restrições práticas que atendam os requisitos legais estipulados no Código de Estradas da Califórnia. Esses requisitos estão relacionados com limite de peso permitido por eixo de caminhão. A abordagem heurística combina um algoritmo de construção de paredes com modelos de programação linear inteira. Experimentos, realizados utilizando dados gerados a partir de casos reais mostraram a eficácia da abordagem. O trabalho foi desenvolvido em um componente de software para facilitar as tomadas de decisões na indústria. Um apanhado geral sobre problemas de corte e empacotamento pode ser obtido em Wäscher *et al.* (2007) e Bortfeldt e Wäscher (2013).

3. DEFINIÇÃO DO PROBLEMA

Como comentado inicialmente, este trabalho considera a versão 3D do PMI. Tal problema busca por um empacotamento de itens tridimensionais (caixas) em um único recipiente (contêiner) de tal forma que o valor total empacotado é maximizado.

Cada item a ser empacotado tem uma orientação fixa e deve ser empacotado de forma ortogonal aos lados do recipiente. Para um item i existe uma largura w_i, uma altura h_i; a profundidade p_i; e, um valor v_i associado. O recipiente (contêiner) possui uma largura W, altura H e profundidade P. O item é empacotado pelo seu canto mais baixo, mais a

esquerda e mais a frente, nesta ordem. O valor do item corresponde ao seu volume, de forma que o objetivo do problema passa a ser minimizar o desperdício de ocupação do contêiner.

3.1. Heurística para o PMI

O desenvolvimento da heurística parte do trabalho de Leung *et al.* (2012). Em linhas gerais, dado um conjunto de itens e um espaço vazio no recipiente, o algoritmo seleciona um item conforme a ocupação do contêiner, observando o espaço vazio. Em outras palavras, é calculada a aptidão de cada item do conjunto dado o espaço vazio, selecionando o item com melhor aptidão. Ao empacotar um item, novos espaços vazios são criados e a análise do algoritmo continua observando tais espaços e o conjunto de itens da instância.

O espaço vazio é determinado a partir dos atributos: canto inferior esquerdo do espaço (x, y, z), largura do espaço (w), altura do espaço (h), a profundidade do espaço (p), a altura (l) da parede do lado esquerdo e a altura (r) da parede do lado direito.

O algoritmo *Empacota*, obtido de Leung *et al.* (2012), descreve a rotina usada para empacotar, dado um conjunto de itens, um subconjunto destes. As variáveis presentes no algoritmo são: *vpn* denota o número de espaços vazios disponíveis; f_i indica o valor de aptidão do item *i*; *minw* denota a menor largura dentre o conjunto de itens; *s* indica o espaço vazio em análise. Ao final, o algoritmo retorna o desperdício em porcentagem, calculado como: $\frac{volume\ não\ usado\ do\ recipiente}{volume\ total\ do\ recipiente}\ X\ 100.$

Empacota(conjunto de itens X, recipiente com dimensões WxHxP):

```
1   vpn ← 1; s.x ← 0; s.y ← 0; s.w ← W; s.l ← L; s.r ← L;
2   Enquanto s.y < L e vpn > 0 faça
3     Se s.w ≥ minw então
4       Para cada item não empacotado i faça
5           f_i ← APT-3D(i, s);
6       Selecione o item r com o maior valor de fitness: f_r = max{f_i};
7       Se f_r ≥ 0 então
8         Se s.l ≥ s.r então
9           O item r é empacotado do lado esquerdo;
10          Atualizar o vetor S, minw e vpn;
11        Senão
12          O item é empacotado do lado direito;
13          Atualizar o vetor S, minw e vpn;
14      Senão
15        Atualizar o vetor S, minw e vpn;
16    Senão
17      Atualizar o vetor S, minw e vpn;
18    Junte os espaços em S que não cabem qualquer item do conjunto de itens.
19    Encontrar em S o espaço s mais abaixo, mais a esquerda e mais a frente.
20  Retornar o desperdício;
```

No algoritmo *Empacota*, a linha 1 inicializa com o espaço vazio *s* sendo todo o contêiner, enquanto que a linha 2 verifica se existe espaço vazio disponível. Assim, enquanto houver espaço vazio disponível, verifica-se se o mesmo tem largura suficiente para empacotar algum item (linha 3). Se não for possível, o algoritmo segue para a linha 16, de forma que o espaço atual *s* é inviável, e o vetor de espaços vazios *S* e a variável *vpn* são atualizadas na linha 17. Nas linhas 4 até 13, o algoritmo busca, no conjunto de itens, pelo item de melhor valor de aptidão. Este item é selecionado para, então, ser empacotado no espaço s atual.

A aptidão dos itens é obtida através do algoritmo *APT-3D*. Este algoritmo analisa e ranqueia os itens de acordo com o espaço vazio atual. Essa função gera melhores valores de aptidão para os itens que melhor ocupam o dado espaço vazio.

APT-3D (item i, espaço vazio s):

```
1 Se s.l ≥ s.r então
2   Se s.w = i.largura e s.h = i.altura e s.p = i.profundidade então
3     fitness ← 9;
4     Senão se s.w = i.largura e s.h = i.altura e s.p > i.profundidade então
5       fitness ← 8;
6       Senão se s.w = i.largura e s.h > i.altura e s.p = i.profundidade então
7         fitness ← 7;
8         Senão se s.w > i.largura e s.h = i.altura e s.p = i.profundidade então
```

```
9                fitness ← 6;
10                 Senão se s.w = i.largura e s.l = i.altura e s.h ≥ i.altura e s.p =  i.profundidade
                   então
11                    fitness ← 5;
12                    Senão se s.w = i.largura e s.l > i.altura e s.h ≥ i.altura e s.p >
                      i.profundidade então
13                       fitness ← 4;
14                       Senão se s.w > i.largura e s.l = i.altura e s.p = i.profundidade
                         então
15                          fitness ← 3;
16                          Senão se s.w  > i.largura e s.l = i.altura e s.p > i.profundidade
                            então
17                             fitness ← 2;
18                             Senão se s.w > i.largura e s.h ≥ i.altura e s.p =
                               i.profundidade então
19                                fitness ← 1;
20                                Senão se s.w > i.largura e s.h ≥ i.altura e
                                  s.p > i.profundidade então
21                                   fitness ← 0;
22                                   Senão
23                                      fitness ← -1;
24 Senão Repetir os mesmos passos anteriores considerando agora s.r no lugar de s.l;
26 Retornar o valor em fitness;
```

Ao usar apenas o algoritmo *Empacota* não é possível obter soluções de qualidade. Por este motivo, usa-se um algoritmo geral obtido de Leung *et al*. (2012), baseado na meta-heurística Recozimento Simulado (KIRKPATRICK *et al*., 1983), para gerar vários empacotamentos diferentes. Em linhas gerais, realizando ordenações e trocas de itens de posição no conjunto de itens, chama-se a rotina *Empacota* na tentativa de obter uma solução diferente, possivelmente melhor. Além disso, soluções piores (empacotamentos com maior desperdício que o corrente) podem ser aceitas usando a função de probabilidade do recozimento simulado na tentativa de diversificar o espaço de busca. Esse processo é repetido um número máximo de iterações. A melhor solução é retornada no final como solução para a instância de entrada. O algoritmo geral *RS* realiza esses passos.

RS (conjunto X com n itens, dimensões WxHxP do recipiente):

```
1     Faça melhor_desperdício ← 0;
2     Ordene o conjunto de itens em X de forma não crescente de perímetro;
3     melhor_desperdício ← Empacota(X, WxHxP);
4     Para i de 1 até n-1 faça
5          Para j de i + 1 até n faça
6               Fazer a troca dos itens nas posições i e j de X, resultando em X';
7               atual_desperdício ← Empacota(X', WxHxP);
8               Se atual_desperdício < melhor_desperdício então
9                    melhor_desperdício ← atual desperdício;
10                   X ← X';
11              Senão, desfazer a troca dos itens de i e j, e obter a ordenação inicial X;
12    Estabelecer uma temperatura inicial T0 e faça T ← T0;
13    Enquanto o número IT de iterações não for alcançado faça
14         Para i de 1 até MAX faça
15              Selecione aleatoriamente dois itens das posições j e k em X;
16              Obter X' trocando a ordem dos itens nas posições j e k de X;
17              atual_desperdício ← Empacota(X', WxHxP);
18              Se atual_desperdício < melhor_desperdício então
19                 melhor_desperdício ← atual_desperdício;
20                 X ← X';
21              Senão
22                 Se exp [(melhor_desperdício – atual_desperdício) / T] ≥ rand(0, 1) então
23                    X ← X';
24         T  ← α T;
25    Ordene os itens por ordem não crescente de: ou perímetro ou área ou altura ou largura. Escolha aleatoriamente
qual das ordenações fazer;
26    Retornar melhor_desperdício;
```

No algoritmo *RS*, com o intuito de escapar de soluções locais, a linha 22 busca aceitar soluções ruins de acordo com a função de probabilidade que analisa a solução atual e a melhor solução já encontrada. Neste contexto, a função *rand(0,1)* gera de forma aleatória uniforme um número real entre 0 e 1.

Diante das possibilidades para arranjar uma caixa dentro de um contêiner, os seguintes casos foram considerados:

(i) Caso em que a largura, a profundidade e a altura do item são menores que as do espaço, e o item é empacotado do lado esquerdo.

(ii) Caso em que a largura e a altura do item são menores que as do espaço, mas as profundidades são iguais, para um empacotamento do lado esquerdo.

(iii) Caso em que a largura e a profundidade do item são iguais a largura e a profundidade do espaço, respectivamente, mas a altura é menor.

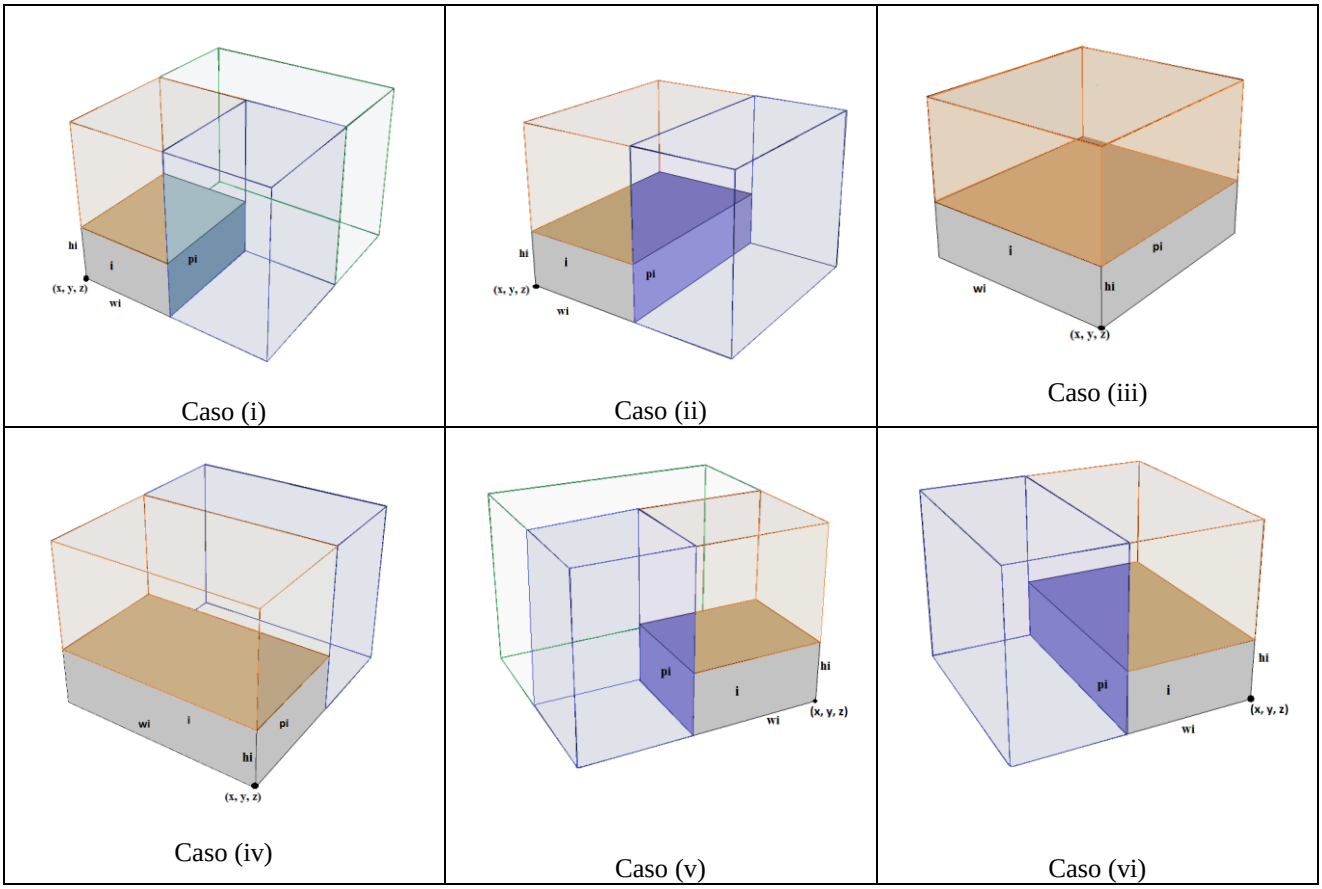
(iv) Caso em que a largura do item é igual a largura do espaço e a profundidade e a altura do item são menores que as do espaço.

(v) Caso em que a largura, a profundidade e a altura do item são menores que as do espaço, para um empacotamento do lado direito.

(vi) Caso em que a profundidade é igual a profundidade do espaço e a largura e altura são menores, para um empacotamento do lado direito.

Vale comentar que eles não cobrem todas as possíveis situações quanto aos espaços gerados, mas são um bom indicativo para um trabalho futuro mais geral. Além disso, para cada um dos casos acima, pode-se gerar três, dois, um ou nenhum novo espaço vazio. A Figura 1 ilustra cada um dos casos acima mostrando os espaços vazios gerados na cor laranja, azul ou verde, dado o item *i* em cinza empacotado de acordo com os casos (i)-(vi).

Figura 1: Representação dos casos (i)-(vi) que surgem durante o empacotamento das caixas.



4. TESTES COMPUTACIONAIS

Para realizar os testes, fez-se a devida codificação dos algoritmos na linguagem de programação C e usou um computador com processador Intel® Dual Core™ 2.53 GHz, 4 GB de memória RAM e sistema operacional Linux 12.04 32 bits.

4.1. Resultados

Os testes computacionais foram realizados com instâncias retiradas do trabalho de Egeblad e Pisinger (2009), totalizando 60 instâncias. Os parâmetros para os algoritmos foram obtidos a partir de testes de calibração. Para se chegar em tais valores, buscou-se balancear tempo de processamento com qualidade das soluções em termo de ocupação do recipiente. Neste sentido, teve-se: IT = 500; MAX = 2 *x* (número de itens); $\alpha = 0{,}96$; e, T0 = IT *x* (número de itens) *x* MAX.

Em cada linha da Tabela 1, tem-se o nome da instância, as dimensões do recipiente, o número de itens, o desperdício em porcentagem e o tempo computacional gasto em segundos.

Tabela 1: Resultados para o caso 3D do PMI obtidos com o algoritmo *RS*.

Instância	**Dimensões do recipiente**	**Número de itens**	**Resultados do algoritmo *RS***	
			Desperdício (%)	*Tempo (s)*
EP3-20-C-C-50	120x120x242	20	24,32%	40,46
EP3-20-C-C-90	146x146x293	20	20,86%	96,49
EP3-20-C-R-50	116x116x232	20	26,97%	42,05
EP3-20-C-R-90	141x141x283	20	27,33%	41,04
EP3-20-D-C-50	59x59x119	20	47,38%	0,55
EP3-20-D-C-90	72x72x144	20	38,65%	9,59
EP3-20-D-R-50	52x52x104	20	16,16%	5,58
EP3-20-D-R-90	63x63x126	20	15,24%	23,92
EP3-20-F-C-50	106x106x213	20	19,02%	0,51
EP3-20-F-C-90	129x129x259	20	43,50%	0,78
EP3-20-F-R-50	104x104x208	20	14,31%	0,59
EP3-20-F-R-90	127x127x254	20	23,92%	1,47
EP3-20-L-C-50	80x80x160	20	34,31%	0,40
EP3-20-L-C-90	97x97x194	20	24,34%	1,28
EP3-20-L-R-50	77x77x154	20	18,52%	0,55
EP3-20-L-R-90	94x94x188	20	13,05%	1,60
EP3-20-U-C-50	125x125x250	20	38,02%	0,52
EP3-20-U-C-90	152x152x305	20	34,52%	0,58
EP3-20-U-R-50	129x129x258	20	19,16%	0,30
EP3-20-U-R-90	157x157x314	20	8,98%	0,54
EP3-40-C-C-50	151x151x303	40	23,24%	143,27
EP3-40-C-C-90	184x184x369	40	32,98%	123,88
EP3-40-C-R-50	141x141x282	40	12,87%	56,89
EP3-40-C-R-90	172x172x344	40	15,91%	65,09
EP3-40-D-C-50	75x75x150	40	37,82%	34,06
EP3-40-D-C-90	91x91x182	40	28,51%	106,52
EP3-40-D-R-50	60x60x121	40	6,72%	58,22
EP3-40-D-R-90	74x74x148	40	6,88%	176,67
EP3-40-F-C-50	134x134x268	40	29,40%	2,31
EP3-40-F-C-90	163x163x326	40	23,77%	11,53
EP3-40-F-R-50	131x131x263	40	12,70%	4,32
EP3-40-F-R-90	160x160x320	40	14,59%	12,07
EP3-40-L-C-50	100x100x201	40	31,29%	3,04
EP3-40-L-C-90	122x122x245	40	20,83%	6,41
EP3-40-L-R-50	94x94x188	40	9,52%	5,10
EP3-40-L-R-90	114x114x229	40	5,03%	11,09
EP3-40-U-C-50	158x158x316	40	41,05%	1,70
EP3-40-U-C-90	192x192x384	40	12,07%	3,14
EP3-40-U-R-50	164x164x329	40	10,79%	4,60
EP3-40-U-R-90	200x200x400	40	12,07%	5,47
EP3-60-C-C-50	173x173x347	60	15,44%	94,32
EP3-60-C-C-90	211x211x422	60	24,48%	33,48
EP3-60-C-R-50	161x161x322	60	25,80%	183,49
EP3-60-C-R-90	196x196x392	60	21,64%	84,48
EP3-60-D-C-50	85x85x171	60	19,38%	123,08
EP3-60-D-C-90	104x104x209	60	26,16%	193,53

EP3-60-D-R-50	67x67x135	60	4,34%	152,00
EP3-60-D-R-90	82x82x164	60	12,43%	124,35
EP3-60-F-C-50	153x153x307	60	15,94%	16,96
EP3-60-F-C-90	186x186x373	60	29,63%	24,39
EP3-60-F-R-50	150x150x301	60	9,19%	15,19
EP3-60-F-R-90	183x183x367	60	7,24%	32,73
EP3-60-L-C-50	115x115x230	60	17,37%	10,25
EP3-60-L-C-90	140x140x280	60	30,81%	12,68
EP3-60-L-R-50	104x104x209	60	15,92%	17,98
EP3-60-L-R-90	127x127x255	60	14,61%	28,61
EP3-60-U-C-50	180x180x361	60	8,93%	8,66
EP3-60-U-C-90	220x220x440	60	34,01%	11,78
EP3-60-U-R-50	184x184x369	60	14,55%	9,26
EP3-60-U-R-90	224x224x449	60	19,71%	17,81

Na média, o desperdício do recipiente foi de 21,07% e o tempo computacional de 38,32 segundos, de acordo com os resultados na Tabela 1. A instância mais demorada gastou cerca de 193,53 segundos. O pior resultado em termos de desperdício ocorreu para a instância EP3-20-D-C-50 com desperdício de quase metade do recipiente, isto é, de 47,38%. Apesar disso, o número de instâncias com desperdício inferior a 15% é de 19, sendo considerado um resultado mediano para aplicações práticas, uma vez que muitas empresas ainda usam procedimentos manuais para determinar o empacotamento.

Acredita-se que melhores resultados poderiam ter sido obtidos caso se considerasse todas as possibilidades na hora de gerar os espaços vazios. Por outro lado, o tempo computacional e o uso de memória para armazenamento poderiam ter crescido bastante, tornando o ganho em termos de ocupação não necessariamente satisfatório.

Como trabalhos futuros, pretende-se estudar o efeito das restrições práticas para o caso 3D, uma vez que o carregamento de contêineres aparece largamente em aplicações práticas (JUNQUEIRA, 2009). Outra linha de investigação consiste na consideração de todas as possibilidades para se gerar espaços ao empacotar caixas.

5. CONCLUSÃO

Neste trabalho foi estudado o problema da Mochila Ilimitado 3D. Partiu-se da heurística proposta por Leung *et al*. (2012) com o objetivo de desenvolver uma heurística para o problema em questão.

Para os resultados encontrados, notou-se que apenas 30% dos resultados apontaram para uma taxa de ocupação inferior a 15% do recipiente. Por outro lado, algumas instâncias chegaram a apresentar até 50% de desperdício, certamente não sendo um resultado interessante para empresas ou indústrias.

Propostas para trabalhos futuros estão focadas em considerar novos critérios para a função de aptidão e lidar com diferentes formas para empacotar as caixas. Observou-se que a forma como os espaços são gerados e ocupados interferem drasticamente na ocupação do contêiner. Em vários casos, ocorreu a geração de "buracos" que impossibilitava uma melhor ocupação do contêiner.

REFERÊNCIAS

ANDREW L.; HONG M. The single container loading problem with axle weight constraints. *International Jornal of Production Economics*, 144(6): 358– 69, 2013.

ARENALES M.; ARMENTANO V.; MORABITO R.; YANASSE H. *Pesquisa Operacional*. Rio de Janeiro: Elsevier, 2007.

BALDI M. M.; PERBOLI G.; TADEI R. The three-dimensional knapsack problem with balancing constraints. *Applied Mathematics and Computation*, 218(19): 9802-9818, 2012.

BEASLEY J. E. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36: 297-306, 1985.

BISCHOFF E. E.; RATCLIFF M. S. W. Issues in the development of approaches to container loading. OMEGA. *International Journal of Management Science*, 23(4): 377–390, 1995.

BORTFELDT A.; GEHRING H.; MACK D. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5):641-62, 2003.

CHEN C. S.; LEE S. M.; SHEN Q.S. An analytical model for the container loading problem. *European Journal of Operational Research*, 80: 68–76, 1995.

EGEBLAD J.; PISINGER D. Heuristic approaches for the two-and three- dimensional knapsack packing problem. *Computers and Operations Researc*h; 36(4):1026-1049, 2009.

ELEY M. Solving container loading problems by block arrangement. *European Journal of Operational Research*, 141(2): 393-409, 2002.

GEHRING H.; BORTFELDT A. A genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 401–18, 1997.

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à Pesquisa Operacional.* 8 ed. Porto Alegre: McGraw-Hill, 2007e.
JUNQUEIRA L. *Modelos de programação matemática para problemas de carregamento de caixas dentro de contêineres*. Dissertação de Mestrado, Departamento de Engenharia de Produção, Universidade Federal de São Carlos, São Carlos - SP, Brasil, 2009.
JUNQUEIRA L.; MORABITO R.; YAMASHITA D. S. Modelos de otimização para problemas de carregamento de contêineres com considerações de estabilidade e de empilhamento. *Pesquisa Operacional*, 30: 73-98, 2010.
JUNQUEIRA L.; MOROBITO R.; YAMASHITA D. S. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers and Operations Research*, 39(1): 74-85, 2012.
KIRKPATRICK S.; GELLAT JR C. D.; VECCHI M. P. Optimizing by simulated annealing. *Science*, 220(4598): 671-680, 1983.
LEUNG S. C. H.; ZHANG D.; ZHOU C.; WU T. A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. *Computers and Operations Research*, 39: 64-73, 2012.
PISINGER D. Heuristics for the container loading problem. *European Journal of Operational Research*, 141: 143–53, 2002.
QUEIROZ T. A. *Algoritmos para Problemas de Corte e Empacotamento*. Tese de doutorado, Instituto de Computação, Universidade Estadual de Campinas, Campinas - SP, Brasil, 2010.
WÄSCHER G.; HAUSSNER H.; SCHUMANN H. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3): 1109-1130, 2007.

RESPONSABILIDADE AUTORAL

Os autores são os únicos responsáveis pelo conteúdo deste trabalho.

THE THREE DIMENSIONAL KNAPSACK PROBLEM SOLVED WITH A HEURISTIC APPROACH

Mirella Augusta Sousa Moura, mirella.asm14@hotmail.com[1]
Thiago Alves de Queiroz, taq@ufg.br [1]

[1]Department of Mathematics – Federal University of Goiás/Campus Catalão,
Av. Dr. Lamartine Pinto de Avelar, 1120, Setor Universitário, 75704-020, Catalão-GO, Brazil.

Abstract: *Packing problems have been used in industries constantly, mainly in supply chain and logistic organizations. This research deals with the knapsack problem in its three-dimensional version with the objective of maximize the occupied volume for a single container. We study the version of the problem in which there is no limit on the number of copies of each item that may be packed, but items must be arranged with no overlapping and respecting the container dimensions. We present a heuristic based on the Simulated Annealing metaheuristic that aims for minimize the trim loss, and that selects items with the best fitness value, considering empty spaces. The C programming language was used to code the algorithms, in which computational experiments using instances from the literature shown that the heuristic is reasonable for solve medium-sized instances, whereas it fails in some cases.*

Keywords*: Knapsack Problem; Unconstrained Knapsack; Three-dimensional Packing; Simulated Annealing.*