UMA ABORDAGEM HÍBRIDA BASEADA EM CONSTRUÇÃO DE BLOCOS PARA O PROBLEMA DE CARREGAMENTO DE CONTÊINER

Rommel D. Saraiva, Napoleão V. Nepomuceno, Plácido R. Pinheiro
Universidade de Fortaleza, Mestrado em Informática Aplicada
Av. Washington Soares 1321, CEP: 60811-905, Fortaleza, Ceará, Brasil
rommelds@edu.unifor.br, {napoleaovn, placido}@unifor.br

RESUMO

Este artigo apresenta um algoritmo híbrido baseado em construção de blocos para resolver o Problema de Carregamento de Contêiner. A ideia central do algoritmo é decompor esse clássico Problema de Corte e Empacotamento em dois subproblemas: um problema de empacotamento tridimensional que objetiva gerar blocos de caixas; e um problema de posicionamento bidimensional no piso do contêiner que busca maximizar o volume ocupado por um subconjunto desses blocos. Por um lado, a fase de geração de blocos é completamente determinística. Algoritmos construtivos que realizam essa tarefa têm sido recentemente propostos. Por outro lado, a fase de posicionamento de blocos é não-determinística. Esta compreende a metodologia híbrida Gerar-e-Resolver, que combina uma metaheurística com um modelo exato. Experimentos computacionais são realizados em bibliotecas de testes da literatura, e os resultados encontrados são comparados com os de outros autores.

PALAVARAS CHAVE. Corte e Empacotamento, Carregamento de Contêiner, Metaheurísticas Híbridas.

Logística e Transportes, Metaheurísticas, Programação Matemática.

ABSTRACT

This paper presents a hybrid block building based approach to solve the Container Loading Problem. The general idea behind the proposed algorithm is to decompose this classic Cutting and Packing Problem into two subproblems, namely, a three-dimensional problem of generating a set of blocks of boxes and a two-dimensional problem of positioning a subset of these blocks on the floor of the container. On the one hand, the block generation phase is completely deterministic. Constructive algorithms to accomplish with this task have been recently proposed. On the other hand, the positioning phase is non-deterministic. It comprises the Generate-and-Solve methodology, a hybrid optimization framework that combines a metaheuristic engine with an exact solver. Computational experiments are performed on benchmark problem instances and the achieved results are compared with those found by other authors.

KEYWORDS. Cutting and Packing, Container Loading, Hybrid Metaheuristics.

Logistics and Transport, Metaheuristics, Mathematical Programming.

1. Introdução

No âmbito da Pesquisa Operacional, Problemas de Corte e Empacotamento são problemas clássicos de Otimização Combinatória que surgem em diversos segmentos industriais durantes os processos de produção. Embora teoricamente distintos, ambos os problemas, na prática, são caracterizados por possuírem uma mesma estrutura lógica: o problema de corte consiste em atender a uma demanda de itens a partir do corte de material (em geral, matéria-prima), enquanto o problema de empacotamento visa determinar a melhor forma de arranjar itens dentro de um ou mais objetos. Devido à vasta aplicabilidade, esses problemas têm sido muito estudados durante as últimas décadas.

1.1. O Problema de Carregamento de Contêiner

O foco deste artigo é um caso particular dos Problemas de Corte e Empacotamento que está relacionado ao empacotamento de caixas no interior de um único contêiner, sendo popularmente conhecido na literatura como Problema de Carregamento de Contêiner. Em suma, o problema envolve dois grupos de dados: de um lado, tem-se um único objeto tridimensional (o contêiner), a princípio, vazio; do outro, tem-se um conjunto de itens retangulares tridimensionais (as caixas) que podem ser divididos em vários tipos, cada qual associado a um valor de lucro (usualmente relacionado ao seu volume). O objetivo final do problema é encontrar uma configuração de carga que maximize o lucro/volume ocupado no interior do contêiner de maneira que as caixas carregadas estejam ortogonalmente e inteiramente posicionadas no interior do contêiner, não sobrepostas umas às outras. Além disso, grande parte dos trabalhos que englobam carregamentos de contêineres têm destacado duas particularidades que frequentemente aparecem em situações reais: a restrição de *orientação* dita que certos tipos de caixas não podem ser rotacionados. Caso não seja levada em consideração, até seis orientações diferentes são permitidas para cada tipo de caixa; já a restrição de *estabilidade* exige que a superfície inferior das caixas não posicionadas diretamente no piso do contêiner deve ser inteiramente suportada pela superfície superior de uma ou mais caixas previamente alocadas.

Presente principalmente em indústrias de logística e manufatura, o Problema de Carregamento de Contêiner vem à tona quando se deseja minimizar o custo do transporte de mercadorias, já que o valor dessa operação é diretamente associado à quantidade de contêineres alugados para armazená-las. Assim, quanto mais contêineres forem utilizados, maior será o custo de exportação da carga. Portanto, a vantagem de aproveitar o volume do contêiner ao máximo se tornou um fator relevante para reduzir custos operacionais.

1.2. Heurísticas de Resolução e Contribuição

De acordo com Scheithauer (1992), o Problema de Carregamento de Contêiner pertence à classe NP-*Hard*. Como esperado, não há qualquer algoritmo determinístico apto a resolver o problema em tempo polinomial à otimalidade, já que métodos exatos são efetivamente aplicados apenas em instâncias de tamanho moderado. Diante dessa dificuldade, maior atenção é devotada aos trabalhos que usam heurísticas. Estas, conforme Fanslau e Bortfeldt (2010), são catalogadas dentro dos seguintes grupos (não necessariamente disjuntos):

(1) *Heurísticas convencionais* são aquelas na qual um padrão de carregamento é construído elemento a elemento, seguindo algum critério heurístico de otimização, até que se tenha um padrão final de carregamento. Os procedimentos de George e Robinson (1980), Bischoff e Ratcliff (1995) e Zhu e Lim (2012) são clássicos exemplos de heurísticas convencionais.

(2) *Metaheurísticas* são ferramentas de busca inteligente que têm sido extensivamente aplicadas para a resolução do Problema de Carregamento de Contêiner. Nestas, estão incluídas, dentre outras, técnicas de computação evolucionária, como os Algoritmos Genéticos utilizados por Gehring e Bortfeldt (1997) e Gonçalves e Resende (2012); e

métodos de busca em trajetória, como o algoritmo que adota a Têmpera Simulada desenvolvido por Mack *et al* (2003), a Busca Tabu empregada por Liu *et al* (2011), e o procedimento de Moura e Oliveira (2005) baseado em *Greedy Randomized Adaptive Search Procedure* (GRASP).

(3) *Buscas inteligentes em árvore ou grafo* têm produzido resultados de notável relevância. Os trabalhos concebidos por Morabito e Arenales (1994), Pisinger (2002) e Fanslau e Bortfeldt (2010) apresentam algoritmos que utilizam essa técnica de resolução.

Este artigo utiliza uma abordagem de construção de blocos fundamentada na ideia geral apresentada por Haessler e Talbot (1990) e Gehring e Bortfeldt (1997): em um primeiro momento, blocos estáveis de caixas são construídos por meio dos algoritmos determinísticos recém-propostos por Fanslau e Bortfeldt (2010); após isso, um subconjunto desses blocos é posicionado no piso do contêiner através de uma variante da metodologia híbrida Gerar-e-Resolver, introduzida por Nepomuceno *et al* (2007, 2008). Bibliotecas de testes largamente utilizadas para validação de diversos trabalhos sobre Problemas de Corte e Empacotamento, bem como análises comparativas com os resultados de outros algoritmos propostos, são tomadas como base para fins de avaliação de desempenho.

1.3. Organização do Trabalho

Para uma melhor compreensão, o presente artigo é estruturado da seguinte forma: a Seção 2 revisa detalhadamente a metodologia híbrida Gerar-e-Resolver; a Seção 3 descreve a abordagem de construção de blocos proposta; a Seção 4 apresenta uma análise dos resultados computacionais obtidos em bibliotecas de testes utilizadas na literatura; por fim, a Seção 5 finaliza o artigo com algumas conclusões e perspectivas futuras.

2. Gerar-e-Resolver

Introduzido por Nepomuceno *et al* (2007, 2008), o *framework* híbrido Gerar-e-Resolver é caracterizado pela combinação de dois componentes conceituais distintos, como ilustrado na Figura 1: o Gerador de Instâncias Reduzidas e o Decodificador de Instâncias Reduzidas. Um método exato encapsulado no Decodificador de Instâncias Reduzidas é encarregado de resolver instâncias reduzidas do problema original (ou seja, subproblemas) que ainda preservam a sua estrutura conceitual legítima. Assim, uma solução viável para um dado subproblema será também uma solução viável para o problema original. Em um nível superior, uma metaheurística (implementada no Gerador de Instâncias Reduzidas) é responsável por gerar instâncias reduzidas que, quando submetidas ao Decodificador de Instâncias Reduzidas, são associadas a um valor de aptidão (*fitness*) utilizado para guiar o processo de busca da metaheurística. Nos pioneiros estudos de Nepomuceno *et al* (2007, 2008), a metaheurística escolhida para implementar o Gerador de Instâncias Reduzidas foi um Algoritmo Genético padrão, enquanto o LINGO DLL se ocupou da função do Decodificador de Instâncias Reduzidas.

Denominadas de *metaheurísticas híbridas* por Dumitrescu e Stützle (2003), metodologias que combinam métodos exatos e heurísticos, como a Gerar-e-Resolver, são vistas como poderosas ferramentas de resolução de problemas de natureza combinatória, já que as melhores características de cada uma das abordagens são potencializadas no intuito de obter soluções de boa qualidade para os problemas tratados.

No que segue, são descritos os componentes da metodologia híbrida quando o problema sob investigação é atacado diretamente pelo *framework* Gerar-e-Resolver.

2.1. Decodificador de Instâncias Reduzidas

Para o Decodificador de Instâncias Reduzidas, codifica-se, em um *solver*, uma extensão do modelo de Programação Linear Inteira Binária—proposto por Beasley (1985)—para resolver o Problema de Carregamento de Contêiner. Considere um conjunto de caixas agrupadas em *m* tipos. Para cada tipo *i*, caracterizado pelo comprimento, largura e altura (l_i, w_i, h_i) e pelo volume v_i, tem-se uma quantidade b_i de caixas. Considere também

um contêiner de posse das dimensões (L, W, H). As caixas devem ser ortogonalmente carregadas dentro do contêiner de maneira que o aproveitamento do seu espaço interno seja maximizado.

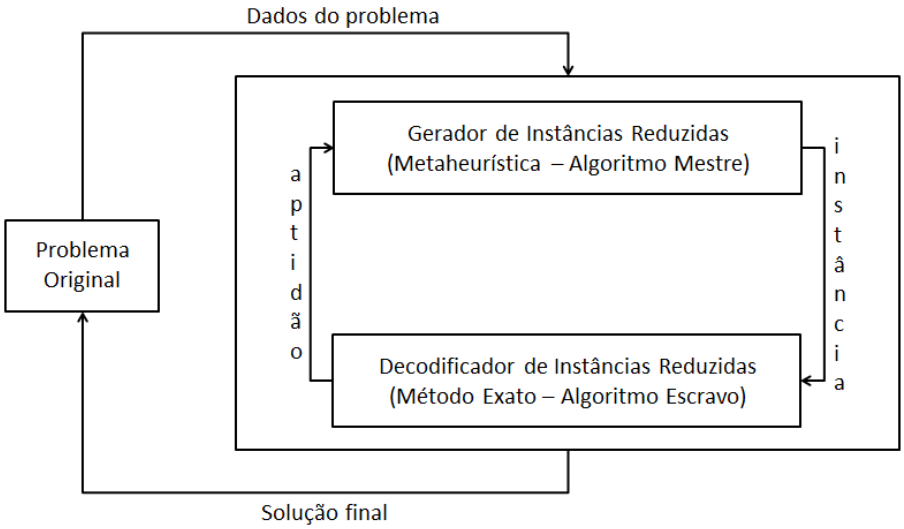


Figura 1: Metodologia híbrida Gerar-e-Resolver.

Cada variável binária do programa representa a decisão de colocar ou não uma caixa do tipo i na coordenada (x, y, z) do contêiner. Sem perda de generalidade, pode-se afirmar que x, y e z pertencem, respectivamente, aos seguintes conjuntos de discretização:

$$X = \{x | x = \sum_{i=1}^{m} \alpha_i l_i, x \leq L - \min_{i=1,\dots,m}\{l_i\}, \alpha_i \geq 0, \alpha_i \in \mathbb{Z}^+\} \quad (1.1)$$

$$Y = \{y | y = \sum_{i=1}^{m} \beta_i w_i, y \leq W - \min_{i=1,\dots,m}\{w_i\}, \beta_i \geq 0, \beta_i \in \mathbb{Z}^+\} \quad (1.2)$$

$$Z = \{z | z = \sum_{i=1}^{m} \gamma_i h_i, z \leq H - \min_{i=1,\dots,m}\{h_i\}, \gamma_i \geq 0, \gamma_i \in \mathbb{Z}^+\} \quad (1.3)$$

Dados x_j o j-ésimo elemento do conjunto X, y_k o k-ésimo elemento do conjunto Y e z_l o l-ésimo elemento do conjunto Z. Se uma caixa do tipo i for colocada na posição (x_j, y_k, z_l), então não se pode ter qualquer outra caixa nas posições (x_p, y_q, z_r) satisfazendo $x_j \leq x_p \leq x_j + l_i - 1$, $y_k \leq y_q \leq y_k + w_i - 1$ e $z_l \leq z_r \leq z_l + h_i - 1$, com $p = 1, \dots, |X|$, $q = 1, \dots, |Y|$ e $r = 1, \dots, |Z|$. Assim, para evitar a sobreposição de caixas, define-se a matriz de incidência $g_{ijklpqr}$ como:

$$g_{ijklpqr} = \begin{cases} 1, \text{se } x_j \leq x_p \leq x_j + l_i - 1, \\ \quad y_k \leq y_q \leq y_k + w_i - 1, \\ \quad z_l \leq z_r \leq z_l + h_i - 1; \\ 0, \text{caso contrário.} \end{cases} \quad (2)$$

que deve ser computada, a priori, para cada caixa do tipo i $(i = 1, \dots, m)$, para cada posição (x_j, y_k, z_l) $(j = 1, \dots, |X|, k = 1, \dots, |Y|, l = 1, \dots, |Z|)$, e para cada posição (x_p, y_q, z_r) $(p = 1, \dots, |X|, q = 1, \dots, |Y|, r = 1, \dots, |Z|)$. Considere $J_i = \max\{j | x_j \leq L - l_i\}$, $K_i = \max\{j | y_j \leq$

$W - w_i\}$ e $L_i = \max\{j | z_j \leq H - h_i\}$ e as variáveis de decisão a_{ijkl} definidas como:

$$a_{ijkl} = \begin{cases} 1, \text{se uma caixa do tipo } i \text{ for colocada na posição } (x_j, y_k, z_l); \\ 0, \text{caso contrário.} \end{cases} \quad (3)$$

Finalmente, o Problema de Carregamento de Contêiner pode ser formulado como:

$$\text{maximizar} \quad \sum_{i=1}^{m} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} \sum_{l=1}^{L_i} v_i a_{ijkl} \quad (4.1)$$

$$\text{sujeito à:} \quad \sum_{i=1}^{m} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} \sum_{l=1}^{L_i} g_{ijklpqr} a_{ijkl} \leq 1, \forall p \in X, \forall q \in Y, \forall r \in Z \quad (4.2)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} \sum_{l=1}^{L_i} a_{ijkl} \leq b_i, i = 1, \dots, m \quad (4.3)$$

$$a_{ijkl} \in \{0,1\}, i = 1, \dots, m, \forall j \in X, \forall k \in Y, \forall l \in Z \quad (4.4)$$

2.2. Gerador de Instâncias Reduzidas

Partindo do fato de que o modelo matemático exposto anteriormente contém $O(m|X||Y||Z|)$ variáveis e $O(|X||Y||Z|)$ restrições, implementa-se um Algoritmo Genético padrão para gerar instâncias reduzidas do problema original, sendo cada indivíduo representado por um cromossomo binário cujo tamanho é designado pela soma das cardinalidades dos conjuntos de discretização, isto é, $|X| + |Y| + |Z|$. Como ilustrado na Figura 2, os bits denotam os elementos dos conjuntos. Um valor 0 (zero) indica que determinado elemento não é utilizado no problema reduzido. Do contrário, um valor 1 (um) sinaliza que o elemento faz parte da instância reduzida a ser resolvida.

No que diz respeito à população inicial, são gerados P cromossomos da seguinte maneira: uma caixa é escolhida aleatoriamente e somente os elementos múltiplos de suas respectivas dimensões são considerados no problema reduzido. Cada subproblema gerado deve ser resolvido pelo modelo exato, cujo valor da função objetivo—que representa o volume utilizado do contêiner—é atribuído ao valor de aptidão do indivíduo associado. Nos casos em que não se consegue resolver ou nem mesmo gerar o modelo da instância reduzida diante da complexidade do subproblema, o indivíduo é descartado e um outro cromossomo é gerado. Na prática, diversos critérios de descarte podem ser utilizados. Alguns devido a retorno de erro do decodificador, como estouro de memória na geração do modelo ou na pilha de execução em sua resolução. Mas também critérios estabelecidos pelo usuário, por exemplo, limite de tempo de execução sem que se encontre ao menos uma solução viável para a instância.

Para obtenção das populações subsequentes, são aplicados os operadores genéticos convencionais (isto é, seleção, cruzamento e mutação). Como critérios de parada, são utilizados, em conjunto, um número máximo de gerações n, uma geração de convergência C e um limite de tempo T.

3. Aplicação Proposta

A aplicação proposta é segmentada em duas etapas ordenadas: a primeira delas abrange a construção de blocos de caixas estáveis. Encarregam-se desse ofício os algoritmos determinísticos recém-propostos por Fanslau e Bortfeldt (2010); já a segunda etapa compreende a instalação de um subconjunto desses blocos no piso do contêiner, tarefa esta

conduzida por uma nova instanciação da metodologia híbrida Gerar-e-Resolver que adota um Algoritmo Genético de Chaves Aleatórias para implementar o Gerador de Instâncias Reduzidas.

Conjuntos de discretização:

X = {0, 3, 5, 6, 8, 9}	→	x1 x2 x3 x4 x5 x6
Y = {0, 4, 7, 8}	→	y1 y2 y3 y4
Z = {0, 2, 4, 6, 7, 8}	→	z1 z2 z3 z4 z5 z6

Cromossomo:

x1	x2	x3	x4	x5	x6	y1	y2	y3	y4	z1	z2	z3	z4	z5	z6
1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0

Instância reduzida:

X' = {0, 6, 9}
Y' = {0, 4, 8}
Z' = {0, 2, 4, 7}

Figura 2: Representação do cromossomo binário.

3.1. Geração de Blocos

Algumas metodologias que aparecem no estado da arte do Problema de Carregamento de Contêiner, como aquelas apresentadas por Parreño *et al* (2010), Fanslau e Bortfeldt (2010), Gonçalves e Resende (2012) e Zhu e Lim (2012), receberam notoriedade durante os últimos anos por alcançar soluções de qualidade bastante satisfatória. Essas metodologias compartilham uma peculiaridade ímpar: elas trabalham com o carregamento de blocos de caixas em vez de simples caixas unitárias, sendo apropriadamente rotuladas como *abordagens de construção de blocos*. Por definição, um bloco é uma coleção de caixas aglomeradas de forma compacta dentro de um cuboide delimitado, como mostra a Figura 3.

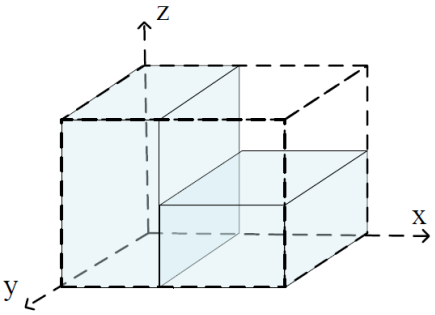
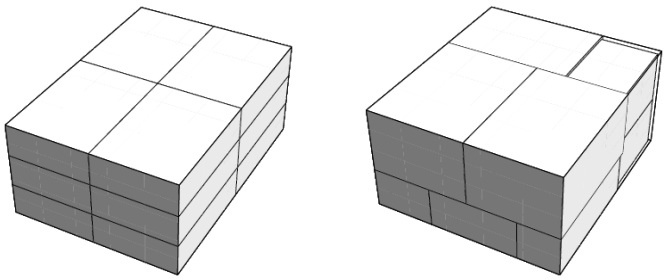


Figura 3: Bloco composto por duas caixas com seu cuboide em linhas tracejadas.

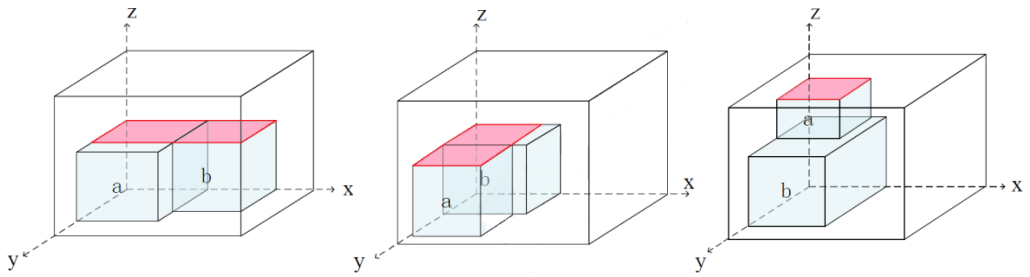
Acompanhando a classificação de Fanslau e Bortfeldt (2010), distinguem-se dois tipos de blocos na literatura: *blocos simples* (Figura 4a) resultam do agrupamento de caixas idênticas, isto é, do mesmo tipo e na mesma orientação espacial. Paredes e colunas são típicos exemplos de blocos simples, que sempre ocupam o volume total dos respectivos cuboides delimitados; já *blocos guilhotinados* (Figura 4b) resultam do agrupamento de múltiplos tipos de caixas que podem ser posicionadas em diferentes orientações espaciais (do exposto, é importante frisar que todo bloco simples é um bloco guilhotinado). Eles são gerados—de maneira recursiva—através da junção de dois blocos guilhotinados ao longo dos eixos *x*, *y* e *z*. Embora os detalhes de implementação para a construção de blocos guilhotinados não terem sido fornecidos pelos autores, Zhu *et al* (2012) conceberam um procedimento plausível para a realização de tal tarefa.



(a) (b)

Figura 4: Bloco (a) simples e (b) guilhotinado.

Com relação à restrição de estabilidade, um atributo singular chamado *área de empacotamento* deve ser introduzido e manipulado para gerar blocos estáveis, isto é, blocos que obedecem à restrição de estabilidade de carga. A área de empacotamento de um bloco é a região retangular localizada na sua face superior e que é completamente suportada por caixas presentes no interior do seu cuboide delimitado. Naturalmente, blocos simples são blocos estáveis. A área de empacotamento de qualquer bloco simples é representada por toda a sua superfície superior. No que concerne à geração de blocos guilhotinados estáveis, algumas normas devem ser obedecidas para validar a combinação de blocos ao longo dos eixos x, y e z. Sejam a e b dois blocos estáveis. Para combiná-los ao longo do eixo x (ou y), os blocos devem possuir a mesma altura; e o comprimento (ou largura) da área de empacotamento resultante da união dos blocos deve ser equivalente ao comprimento (ou largura) do possível bloco gerado. Como consequência, a área de empacotamento do bloco final, como observada na Figura 5a (ou 5b), possui comprimento $a.x + b.x$ (ou $\min\{a.x, b.x\}$) e largura $\min\{a.y, b.y\}$ (ou $a.y + b.y$); para combinar os mesmos blocos ao longo do eixo z, isto é, posicionar o bloco a acima do bloco b, basta que a base do primeiro possa ser inteiramente cercada pela área de empacotamento do segundo. Assim, a área de empacotamento do bloco resultante é a mesma área de empacotamento do bloco a, como mostra a Figura 5c.



(a) (b) (c)

Figura 5: Área de empacotamento resultante da combinação de dois blocos ao longo (a) do eixo x; (b) do eixo y; (c) do eixo z.

Feitas tais considerações, seja B o conjunto total de blocos estáveis obtidos após a execução dos algoritmos de geração de blocos simples e guilhotinados. Como a segunda fase da aplicação proposta tem por fim instalar blocos no piso do contêiner de maneira a maximizar o volume ocupado no seu interior, introduz-se uma função de avaliação para identificar *blocos desejáveis* em B. Um bloco $b(i)$, com $1 \leq i \leq |B|$, é dito desejável se, e somente se:

$$f(b(i)) = \frac{V_{CAIXAS}}{V_{COMPARTIMENTO}} \geq MinUtil \qquad (5)$$

onde V_{CAIXAS} é o volume das caixas aglomeradas em $b(i)$; $V_{COMPARTIMENTO}$ é o volume do espaço ocupado por $b(i)$, isto é, o produto da área da base do seu cuboide delimitado pela altura do contêiner; e *MinUtil* é um parâmetro no intervalo [0, 1] que indica a utilização mínima permitida de um bloco no seu respectivo compartimento. É importante notar que a inequação supracitada é de fundamental valia quando se deseja construir blocos e resolver o Problema de Carregamento de Contêiner sob uma ótica bidimensional, pois, uma vez que o valor de *MinUtil* é próximo de 1 (um), identificam-se blocos com elevado aproveitamento de volume ao passo que ignoram-se aqueles de média/pequena altura, que produzem grandes espaços residuais. Nas abordagens de construção de blocos de Fanslau e Bortfeldt (2010) e Zhu e Lim (2012), $V_{COMPARTIMENTO}$ é substituído por $V_{CUBOIDE}$, que representa o volume do cuboide delimitado $b(i)$. No entanto, em ambos os trabalhos, o problema é atacado sob uma perspectiva tridimensional, não sendo necessário incluir um dado do contêiner (no caso, o valor da altura) para medir a qualidade de um bloco. No que segue, referencia-se como $\bar{B}$ o conjunto de blocos desejáveis em B selecionados por meio do critério acima.

3.2. Algoritmos Genéticos de Chaves Aleatórias

Algoritmos Genéticos de Chaves Aleatórias têm lidado com uma variedade de problemas de Otimização Combinatória. Introduzidos por Bean (1994), essa classe de Algoritmos Genéticos implementa um cromossomo por meio de um vetor de chaves aleatórias, que nada mais são do que número reais representados no intervalo [0, 1]. Um decodificador é um algoritmo determinístico que toma como entrada um vetor de chaves aleatórias e, como saída, retorna uma solução viável associada a um valor de aptidão.

Tal como o padrão, o Algoritmo Genético de Chaves Aleatórias evolui uma população de cromossomos ao longo de um número pré-determinado de gerações. A população inicial é composta por P cromossomos, sendo cada alelo computado de forma randômica no intervalo [0, 1]. Após o *fitness* de cada indivíduo ser calculado por um decodificador, a população é particionada em dois grupos: um grupo P_E de indivíduos-elite, isto é, aqueles que detêm os melhores valores de aptidão, e um grupo de $P - P_E$ indivíduos-não-elite. Para evoluir a população, uma nova geração de indivíduos é produzida. Primeiramente, uma estratégia elitista permite a migração de todos os indivíduos-elite da população da geração K para a população da geração $K + 1$. Adiante, o operador de mutação do Algoritmo Genético de Chaves Aleatórias introduz P_M mutantes na nova população. Mutantes são indivíduos produzidos de maneira similar àqueles da população inicial. Por fim, descontando P_E indivíduos-elite e P_M mutantes, $P - P_E - P_M$ indivíduos são procriados pelo mecanismo de cruzamento para completar a população da geração $K + 1$. Esses cruzamentos ficam a cargo do operador introduzido por Spears e De Jong (1991), chamado *parametrized uniform crossover*, que, dados dois cromossomos-pais A e B, selecionados aleatoriamente da população corrente, e uma probabilidade p_A, procede da seguinte forma: para cada alelo $i = 1, \dots, N$, onde N é o número de genes do cromossomo, o cromossomo-filho herda do i-alelo de A com probabilidade p_A e o correspondente alelo de B com probabilidade $1 - p_A$.

3.2.1. Codificação da Solução

Na nova variante da metodologia híbrida Gerar-e-Resolver, os indivíduos são representados por cromossomos (de chaves aleatórias) cujos tamanhos são equivalentes à soma do número de blocos desejáveis com as cardinalidades dos conjuntos de discretização, isto é, $|\bar{B}| + |X| + |Y|$. Como observado na Figura 6, nesta nova forma de codificação, cada alelo [0, 1] denota a decisão de contar ou não com determinado elemento de discretização/bloco na instância reduzida a ser resolvida pelo decodificador, dada uma probabilidade de aceitação pré-calibrada relativa a cada conjunto particular (na mesma figura, apenas para fins de exemplificação, essa probabilidade de aceitação é fixa para todos os conjuntos). Neste caso, sejam $p_{\bar{B}}$, p_X e p_Y as respectivas probabilidades de aceitação

relativas aos conjuntos $\bar{B}$, X e Y. Se um alelo $\overline{b(i)} \leq p_{\bar{B}}$, com $1 \leq i \leq |\bar{B}|$, então o correspondente bloco será um bloco candidato a ser posicionado no piso do contêiner durante a fase de decodificação. O mesmo raciocínio é válido para os elementos dos conjuntos de discretização: se um alelo $x(j) \leq p_X$, com $1 \leq j \leq |X|$ (ou $y(k) \leq p_Y$, com $1 \leq k \leq |Y|$), então o correspondente elemento de X (ou Y) fará parte da instância reduzida gerada.

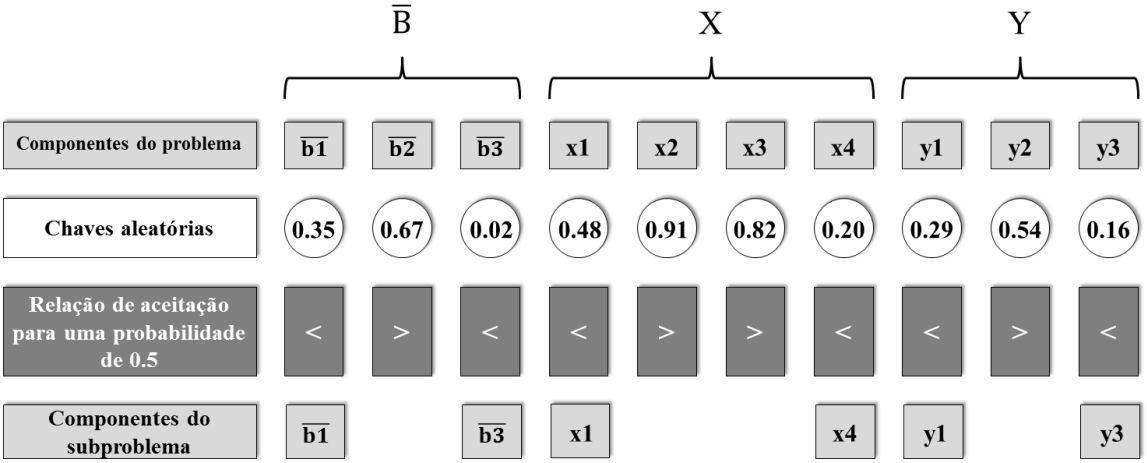


Figura 6: Representação do cromossomo de chaves aleatórias.

Posto isso, é de suma importância salientar que: (i) a escolha de se produzir subproblemas levando-se em conta um subconjunto de $\bar{B}$ não implica, necessariamente, a exclusão de tipos de caixas do carregamento obtido pelo decodificador, pois os algoritmos de construção de blocos geram blocos *não disjuntos*. Assim, para cada tipo de caixa t, a soma das quantidades de caixas desse tipo em $\bar{B}$ pode ser superior à demanda disponível, bem como no componente reduzido de $\bar{B}$. Além disso, a partir do momento em que um bloco é tido como componente de uma instância reduzida, este poderá ser utilizado inúmeras vezes em um mesmo carregamento, desde que a quantidade total de caixas disponíveis seja respeitada; e (ii) uma vez que se atribui valores próximos de 0 (zero) às probabilidades de aceitação, um número relativamente pequeno de blocos e de pontos de discretização é considerado no decodificador. Assim, mesmo com o passar das gerações do Algoritmo Genético de Chaves Aleatórias, espera-se amenizar o risco de se deparar com subproblemas cujos componentes são quase semelhantes àqueles do problema original, inconveniência esta encontrada na versão original da metodologia híbrida Gerar-e-Resolver.

3.2.2. Decodificação da Solução

Para interpretar o Decodificador de Instâncias Reduzidas, codifica-se em um *solver* um modelo de Programação Linear Inteira Binária quase similar àquele proposto por Beasley (1985) para resolver Problemas de Corte e Empacotamento bidimensionais. Descrita abaixo, a formulação usada para resolver instâncias reduzidas é bem semelhante àquela representada por (4.1)—(4.4), porém, com duas sutis diferenças: a ausência do conjunto de discretização Z, que armazena as posições relacionadas à altura do espaço interno do contêiner; e a presença de uma restrição *knapsack*, pois a aplicação trabalha com o carregamento de blocos de caixas em vez de simples caixas singulares.

$$\text{maximizar} \quad \sum_{i=1}^{|\bar{B}|} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} v_i a_{ijk} \tag{6.1}$$

sujeito à:
$$\sum_{i=1}^{|\bar{B}|}\sum_{j=1}^{J_i}\sum_{k=1}^{K_i} g_{ijkpq}a_{ijk} \leq 1, \forall p \in X, \forall q \in Y \quad (6.2)$$

$$\sum_{i=1}^{|\bar{B}|}\sum_{j=1}^{J_i}\sum_{k=1}^{K_i} q_i^t a_{ijk} \leq b_t, t = 1, \ldots, m \quad (6.3)$$

$$a_{ijk} \in \{0,1\}, i = 1, \ldots, m, \forall j \in X, \forall k \in Y \quad (6.4)$$

Nesse modelo, a função objetivo (6.1) maximiza o volume ocupado no interior do contêiner, dados como entrada os $|\bar{B}|$ tipos de blocos e as posições (j, k) do piso do contêiner, com $j \in X$ e $k \in Y$. Assim, as variáveis (binárias) de decisão a_{ijk} indicam se um bloco do tipo i, de volume v_i, é posicionado na posição (j, k); a já conhecida matriz de incidência g evita que blocos sejam sobrepostos uns aos outros em (6.2); a restrição *knapsack* (6.3) impõe que, para cada tipo de caixa t, a soma das quantidades de caixas desse tipo presentes no padrão de carregamento do contêiner não pode exceder a disponibilidade de caixas; por fim, a restrição (6.4) especifica o domínio das variáveis de decisão.

4. Experimentos Computacionais

O Gerador de Instâncias Reduzidas foi desenvolvido em *Java*. Para resolver os subproblemas de Programação Linear Inteira Binária gerados por esse componente, utilizou-se o otimizador IBM ILOG CPLEX 12.5 (com parâmetros *default*). Os testes foram executados em máquinas *Intel Pentium* i7, 3.60 GHz, e 8 GB de memória RAM.

No que concerne à identificação de blocos desejáveis, optou-se, após alguns experimentos preliminares, por configurar $MinUtil = 0.90$. Com relação ao Algoritmo Genético de Chaves Aleatórias, os parâmetros foram calibrados com $P = 300$ indivíduos, número máximo de gerações de 200, $p_E = 0.20 \times P$, $p_M = 0.20 \times P$ e $p_A = 0.7$. Além disso, um tempo $T = 6$ horas foi adotado como limite de tempo de execução. Já as probabilidades de aceitação foram configuradas com $p_X = 0.01$ e $p_Y = 0.015$; os valores de $p_{\bar{B}}$ variam de acordo com a quantidade de blocos desejáveis obtidos, como mostra a Tabela 1.

Tabela 1: Valores de $p_{\bar{B}}$

Quantidade de blocos desejáveis	$p_{\bar{B}}$
$\|\bar{B}\| \leq 1000$	0.4
$1000 < \|\bar{B}\| \leq 2000$	0.2
$2000 < \|\bar{B}\| \leq 3000$	0.1
$1000 < \|\bar{B}\| \leq 4000$	0.05
$\|\bar{B}\| > 4000$	0.025

A abordagem híbrida baseada em construção de blocos (AHCB) apresentada foi aplicada em problemas de uma biblioteca de testes proposta por Bischoff e Ratcliff (1995), na qual estão contidas instâncias de natureza fracamente heterogênea (poucos tipos de caixas, com muitas unidades disponíveis para cada tipo) até instâncias de natureza fortemente heterogênea (muitos tipos de caixas, com poucas unidades disponíveis para cada tipo). Por se tratar de um trabalho ainda em desenvolvimento, são exibidos os resultados das dez primeiras instâncias das bibliotecas BR1 e BR2, que contêm três e cinco tipos de caixas, respectivamente. Os resultados encontrados, bem como uma análise comparativa com os algoritmos propostos *Iterative-Doubling Greedy—Lookahead Tree Search* (ID-GLTS) de Zhu e Lim (2012) e *Container Loading by Tree Search* (CLTRS) de Fanslau e Bortfeldt (2010), que são aqueles que detêm as melhores soluções encontradas até o momento para o Problema de Carregamento de Contêiner levando em consideração a restrição de estabilidade de carregamento, estão presentes na Tabela 2. Nesta, Problema,

Melhor solução, Tempo, Média ± Desvio (solução/tempo), ID-GLTS e CLTRS denotam, nessa ordem: a instância analisada; a melhor solução (em termos de utilização do contêiner) encontrada dentre sete execuções da aplicação; o tempo decorrido para concluir a execução que obteve a melhor solução encontrada; a média e o desvio padrão das soluções e dos tempos de execução; e os resultados encontrados pelos métodos ID-GLTS e CLTRS, conforme dados individuais disponibilizados pelos respectivos autores.

Tabela 2: Resultados computacionais e análise comparativa.

Problema	Melhor solução (utilização)	Tempo (segundos)	Média±Desvio (solução)	Média±Desvio (tempo)	ID-GLTS vS[1]/vM[2]	CLTRS
BR1_01	**95.83**	5071.03	**95.40** ± 0.28	4538.37 ± 67.36	93.79/92.97	93.83
BR1_02	**97.07**	3196.72	**95.21** ± 0.37	3004.07 ± 54.80	95.27/94.52	95.45
BR1_03	**92.67**	2784.30	**92.67** ± 0.11	2637.66 ± 51.35	91.03/91.03	91.03
BR1_04	**95.26**	5313.58	**94.28** ± 0.46	3842.52 ± 61.98	91.48/91.84	91.84
BR1_05	**94.88**	5291.40	**94.51** ± 0.20	4597.45 ± 67.80	95.14/94.78	95.14
BR1_06	**94.27**	5010.34	**93.90** ± 0.33	4681.44 ± 68.42	96.12/94.62	95.72
BR1_07	**95.03**	4554.62	**94.69** ± 0.17	3375.72 ± 58.10	93.85/92.84	94.14
BR1_08	95.32	3214.90	93.75 ± 0.21	2782.35 ± 52.74	97.05/96.53	97.05
BR1_09	87.61	1972.33	87.61 ± 0.11	1738.37 ± 41.69	90.62/90.53	91.57
BR1_10	93.23	2879.50	93.18 ± 0.16	2954.84 ± 54.35	94.67/94.67	94.81
Média (1-10)	**94.11**	3928.87	**93.52**	3415.27	93.90/93.43	94.05
BR2_01	93.00	7655.64	92.89 ± 0.17	6704.82 ± 81.88	95.58/95.45	95.50
BR2_02	**94.83**	11685.10	**94.15** ± 0.17	10942.35 ± 104.60	95.69/95.00	95.50
BR2_03	**93.30**	13413.37	92.58 ± 0.22	14993.97 ± 122.44	95.09/95.52	94.14
BR2_04	**96.68**	6592.77	**94.03** ± 0.39	5359.12 ± 73.20	93.68/93.30	93.68
BR2_05	**95.08**	7912.42	**95.08** ± 0.10	7913.83 ± 88.95	95.58/96.23	95.89
BR2_06	94.72	11485.90	93.94 ± 0.39	8931.28 ± 94.50	96.23/95.99	96.33
BR2_07	**95.74**	11494.20	**95.14** ± 0.25	12510.62 ± 111.85	95.70/95.37	96.19
BR2_08	**93.62**	14294.69	92.73 ± 0.43	10762.45 ± 103.74	94.13/94.13	94.12
BR2_09	**93.96**	18160.53	**93.66** ± 0.23	14081.25 ± 118.66	94.38/94.03	95.23
BR2_10	**95.74**	11401.59	**95.28** ± 0.17	11038.67 ± 105.06	96.14/95.95	96.69
Média (1-10)	**94.66**	11409.62	93.94	10323.83	95.22/95.09	95.32

[1]versão *Single-Best-Space*, [2]versão *Multi-Best-Space*

Por um lado, tomando como referência os dados apresentados em negrito preto na Tabela 2, nota-se que o algoritmo proposto superou, na melhor solução (ou na média), os resultados encontrados pelos métodos ID-GLTS e/ou CLTRS em até oito (ou seis) das dez primeiras instâncias dos grupos BR1 e BR2. Por outro lado, considerando os dados apresentados em negrito vermelho, observa-se que a utilização do volume do contêiner alcançado pela AHCB é, no máximo, 1% menor quando comparada àquelas encontradas pelos algoritmos ID-GLTS e/ou CLTRS. Com relação ao tempo de execução, é importante ressaltar que as metodologias de Fanslau e Bortfeldt (2010) e Zhu e Lim (2012) são determinísticas e possuem tempo de execução inferior à AHCB.

5. Conclusão

Este trabalho apresentou um algoritmo híbrido baseado em construção de blocos para resolver o Problema de Carregamento de Contêiner. A ideia geral do algoritmo é decompor o problema em dois subproblemas: um de empacotamento tridimensional responsável por gerar blocos de caixas, que envolve algoritmos determinísticos recém-propostos; e outro de posicionamento bidimensional responsável por posicionar um subconjunto desses blocos no piso do contêiner, que envolve *o framework* híbrido Gerar-e-Resolver. Experimentos computacionais realizados em bibliotecas de testes da literatura mostraram que o algoritmo proposto superou, em diversas instâncias, os resultados encontrados pelas duas melhores metodologias do estado da arte.

Agradecimentos

Os autores agradecem a Wenbin Zhu e Andreas Bortfeldt por disponibilizarem os resultados individuais das instâncias. O primeiro autor agradece o apoio financeiro dado pela Capes. E o terceiro autor agradece o apoio do CNPq.

Referências

Bean, J. C. (1994), Genetic algorithms and random keys for sequencing and optimization, *ORSA J. Comput.*, 6, 154-160.

Beasley, J. (1985), An exact two-dimensional non-guillotine cutting tree search procedure, *Oper. Res.*, 33, 49-64.

Bischoff, E. E. e Ratcliff, M. (1995), Issues in the development of approaches to container loading, *Omega*, 23, 377-390.

Dumitrescu, I. e Stützle, T., Combinations of local search and exact algorithms, em Raidl, G. R. et al. (Eds.), *Applications of evolutionary computing*, Springer, 2611, 211-223, 2003.

Fanslau, T. e Bortfeldt, A. (2010), A tree search algorithm for solving the container loading problem, *INFORMS J. Comput.*, 22, 222-235.

Gehring, H. e Bortfeldt, A. (1997), A genetic algorithm for solving the container loading problem, *Int. Trans. Oper. Res.*, 4, 401-418.

George, J. A. e Robinson, D. F. (1980), A heuristic for packing boxes into a container, *Comput. Oper. Res.*, 7, 147-156.

Gonçalves, J. F. e Resende, M. G. (2012), A parallel multi-population biased random-key genetic algorithm for a container loading problem, *Comput. Oper. Res.*, 39, 179-190.

Haessler, R. W. e Talbot, F. B. (1990), Load planning for shipments of low density products, *Eur. J. Oper. Res.*, 44, 289-299.

Liu, J., Yue, Y., Dong, Z., Maple, C. e Keech, M. (2011), A novel hybrid tabu search approach to container loading, *Comput. Oper. Res.*, 38, 797-807.

Mack, D., Bortfeldt, A. e Gehring, H. (2003), A parallel hybrid local search algorithm for the container loading problem, *Int. Trans. Oper. Res.*, 11, 511-533.

Morabito, R. e Arenales, M. (1994), An AND/OR-graph approach to the container loading problem, *Int. Trans. Oper. Res.*, 1, 59-73.

Moura, A. e Oliveira, J. F. (2005), A GRASP approach to the container-loading problem, *IEEE Intell. Syst.*, 20, 50-57.

Nepomuceno, N., Pinheiro, P. e Coelho, A., Tackling the container loading problem: A hybrid approach based on integer linear programming and genetic algorithms, em Cotta, C. e van Hemert, J. I. (Eds.), *EvoCOP*, Springer, 154-165, 2007.

Nepomuceno, N., Pinheiro, P. e Coelho, A., A hybrid optimization framework for cutting and packing problems: Case study on constrained 2D non-guillotine cutting, em Cotta, C. e van Hemert, J. I. (Eds.), *Recent Advances in Evolutionary Computation for Combinatorial Optimization of Studies in Computational Intelligence*, Springer, 153, 87-99, 2008.

Parreño, F., Alvarez-Valdés, R., Oliveira, J. e Tamarit, J. M. (2010), Neighborhood structures for the container loading problem: a VNS implementation, *J. Heuristics*, 16, 1-22.

Pisinger, D. (2002), Heuristics for the container loading problem, *Eur. J. Oper. Res.*, 141, 383-392.

Scheithauer, G., Algorithms for the container loading problem, *Oper. Res. Proc. 1991*, Springer, 445-452, 1992.

Spears, W. M. e De Jong, K. A. (1991), On the virtues of parametrized uniform crossover, *In: Proc. of the 4th Int. Conf. on Genetic Algorithms*, 230-236.

Zhu, W. e Lim, A. (2012), A new iterative-doubling greedy—lookahead algorithm for the single container loading problem, *Eur. J. Oper. Res.*, 222, 408-417.

Zhu, W., Oon, W. –C., Lim, A. e Weng, Y. (2012), The six elements to block-building approaches for the single container loading problem, *Appl. Intell.*, 37, 431-445.