UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

TESE DE DOUTORADO

Heurísticas Especializadas Aplicadas ao Problema de Carregamento de Contêiner

Mariza Akiko Utida

Orientador: Prof. Dr. Rubén Augusto Romero Lázaro

Ilha Solteira - SP

Maio de 2012

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Heurísticas Especializadas Aplicadas ao Problema de Carregamento de Contêiner

Aluna: Mariza Akiko Utida

Orientador: Prof. Dr. Rubén Augusto Romero Lázaro

Tese apresentada à Faculdade de Engenharia - UNESP - Campus de Ilha Solteira, como parte dos requisitos para a obtenção do título de Doutora em Engenharia Elétrica.

Área de Conhecimento: Automação.

Ilha Solteira - SP, maio de 2012

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

Utida, Mariza Akiko.
U89h Heurísticas especializadas aplicadas ao problema de carregamento de contêiner / Mariza Akiko Utida. – Ilha Solteira: [s.n.], 2012.
180 f. : il.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2012

Orientador: Rubén Augusto Romero Lázaro
Inclui bibliografia

1. Carregamento de contêiner. 2. Heurística. 3. Metaheurísticas. 4. GRASP (Sistema operacional de computador).

unesp

UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE ILHA SOLTEIRA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: Heurísticas Especializadas Aplicadas ao Problema de Carregamento de Contêiner

AUTORA: MARIZA AKIKO UTIDA
ORIENTADOR: Prof. Dr. RUBEN AUGUSTO ROMERO LAZARO

Aprovada como parte das exigências para obtenção do Título de DOUTOR EM ENGENHARIA ELÉTRICA , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Prof. Dr. RUBEN AUGUSTO ROMERO LAZARO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. JOSÉ ROBERTO SANCHES MANTOVANI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. SILVIO ALEXANDRE DE ARAUJO
Departamento de Ciência de Computação e Estatística / Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto

Prof. Dr. JOSÉ ELIAS CLAUDIO ARROYO
Departamento de Informática / Universidade Federal de Vicosa

Prof. Dr. EDUARDO NOBUHIRO ASADA
Departamento de Engenharia Elétrica / Escola de Engenharia de São Carlos-Usp

Data da realização: 04 de maio de 2012.

A **Deus** e a **minha família** sem os quais
não seria possível chegar até aqui.

Agradecimentos

Este trabalho é uma longa viagem, com muitos percalços pelo caminho. Eu gostaria de expressar meus profundos agradecimentos a todos aqueles que contribuíram direta ou indiretamente, ao longo do desenvolvimento deste doutorado.

Primeiramente a Nossa Senhora Aparecida por ter me concedido força e perseverança para concluir este trabalho, com muita responsabilidade e dignidade.

Em especial à minha família aos meus queridos pais (Mituo e Akie) e a minha irmã (Keilla), que incondicionalmente fazem das minhas conquistas, as suas metas, pelas suas orações, conselhos e estímulos para a realização desse trabalho e pelo grande amor em todos os momentos bons e ruins da minha vida. Especialmente minha mãe, pelo amor incondicional.

Agradeço sinceramente pela orientação ao Prof. Dr. Rubén, pela disposição, profissionalismo e empenho como orientador durante o doutorado, e ao Robinson Alves Lemos, a forma como ajudou na programação, disponibilizando seu tempo e empenho, o meu muito obrigada.

Agradeço também aos membros da banca examinadora Prof. Dr. Jose Roberto Sanches Mantovani, Prof. Dr. Silvio Alexandre de Araújo, Prof. Dr. Eduardo Nobushiro Asada e ao Prof. Dr. Jose Elias Cláudio Arroyo, pela grande colaboração.

Ao Prof. Dr Marcos Julio Rider Flores, pelas sugestões, ajuda com os gráficos para a conclusão deste trabalho.

Aos funcionários do Departamento de Engenharia Elétrica: Deoclécio Mitsuiti Kosaka, Luzinete e Cristina, pela ajuda e amizade convivida durante esse período em Ilha Solteira.

Aos colegas do curso de pós-graduação do LaPSEE e DEE, em especial, aos meus amigos Luisin, Marcia Regina, Marcia, Regiane, Cleide, Vitória, Will e Rosangela.

A todos os professores do Departamento de Engenharia Elétrica, pelos bons momentos compartilhados e ajuda.

Enfim, a todos que de certa forma contribuíram para a realização desta tese, o meu muito obrigada e a minha eterna gratidão.

 Este trabalho teve o suporte financeiro do **Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).**

Resumo

O problema de carregamento de contêiner consiste em carregar um número conhecido de caixas de tipos diferentes e dimensões conhecidas dentro de um contêiner, de modo a utilizar o espaço do contêiner da melhor maneira possível, ou seja, de modo a diminuir os espaços não ocupados. O problema de carregamento de contêiner é um problema NP-difícil e, portanto, muito complexo de ser resolvido de forma exata usando um modelo matemático de programação inteira e pacotes comerciais de otimização baseados em técnicas tipo *branch and bound*. Neste trabalho é realizada a elaboração, assim como a implementação computacional, de dois algoritmos de otimização. A primeira proposta de otimização consiste de um algoritmo heurístico construtivo. A segunda proposta de otimização é um algoritmo GRASP que usa, na fase construtiva, uma generalização do algoritmo heurístico construtivo desenvolvido neste trabalho. A metaheurística GRASP desenvolvida neste trabalho apresentou resultados promissores quando foram realizados testes usando 13 instâncias conhecidas na literatura especializada.

Palavras-chave: Carregamento de Contêiner. Heurística. Metaheurística. GRASP.

Abstract

The container loading problem is to load a known number boxes of different dimensions and known within a container, in order to use the space of the container the best way possible, ie, in order to reduce the empty spaces. The problem of container loading is NP-hard and therefore too complex to be solved using an exact mathematical model integer programming and commercial packages based on optimization techniques *branch and bound* type. In this work the preparation, as well as the computational implementation of two optimization algorithms. The first proposal consists of an optimization constructive heuristic algorithm. The second proposal is an optimization GRASP that uses in construction phase, a generalization of the algorithm constructive heuristic developed in this work. The GRASP developed in this work showed promising results when tests were performed using 13 known instances in the literature specialized.

Palavras-chave: Container loading. Heuristic. Metaheuristic. GRASP.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

INTRODUÇÃO

Com a alta competitividade no mercado, uma empresa deve estar preparada para realizar as atividades de forma apropriada e com o menor custo possível, proporcionando uma considerável vantagem competitiva. Em toda essa operação produtiva utiliza-se muitas tecnologias que deve ser continuamente melhorada.

As empresas de logística vincula a organização de seus clientes e fornecedores, em um esforço integrado de planejamento, implementação e controle do fluxo eficiente e economicamente eficaz dos produtos a serem transportados. O objetivo é proporcionar o menor custo possível para os clientes, satisfazendo as suas necessidades e facilitando as operações relevantes da carga (CECILIO, 2003).

Nos últimos anos as empresas de logística passaram a exercer um papel importante na integração e coordenação dos serviços relativos ao transporte e a armazenagem de mercadorias, sendo responsáveis por deslocar a mercadoria do ponto de origem ao ponto de consumo. Com o avanço das indústrias e a globalização, o transporte de mercadorias teve grande importância no crescimento do comércio e nenhuma invenção teve mais impacto do que o contêiner. Há mais de cinquenta anos atrás para preencher um navio, por exemplo, exigiam-se centenas de trabalhadores que organizavam as cargas para embarque e desembarque. Graças aos contêineres um único operador faz o trabalho, com isso o custo do frete, roubos e avarias das cargas caíram significativamente. Essa necessidade de reduzir os custos de transportes, de manuseio e aumentar a segurança e a rapidez nos intercâmbios comerciais, fez surgir uma caixa padronizada, uma simples caixa que devido a suas inegáveis virtudes, se tornou a espinha dorsal do transporte internacional.

O contêiner é definido como uma grande caixa retangular, feita de metal ou madeira de grande dimensão, usado para colocar pequenas caixas. Conhecidas as dimensões dos contêineres e das caixas a serem carregadas, a questão é encontrar a melhor disposição da carga de maneira a minimizar as

perdas de espaços ociosos dentro dos contêineres. O contêiner tem também a vantagem de facilitar a movimentação, armazenagem e transporte da carga, reduzir o número de volume a ser manipulado, minimizar o tempo de operação de embarque e desembarque, reduzir os custos com embalagem e a diminuição de avarias e roubos de mercadorias. Normalmente as empresas de transportes utilizam o sistema em que aluga o contêiner por dia, mesmo para aluguéis de contêineres a longo prazo. Porém não se consegue evitar as perdas de espaços dentro dos contêineres.

Assim, surge o Problema de Carregamento de Contêiner (PCC). O PCC consiste então em carregar um número conhecido de caixas de tipos distintos e dimensões conhecidas dentro de contêineres, de modo a utilizar o espaço do contêiner da melhor maneira possível, ou seja, de modo a diminuir os espaços não ocupados. As caixas podem ser homogêneas (somente um tipo de caixa), heterogêneas fraca (poucos tipos de caixas, com muitas caixas de cada tipo) e fortemente heterogêneas (muitos tipos de caixas e poucas caixas de cada tipo) (VENDRAMINI, 2007).

A movimentação dos embarques e desembarques dos contêineres é realizada por empilhadeiras e guindastes. O deslocamento e transporte dos contêineres são feitos por equipamentos especiais para acomodar os contêineres da melhor forma possível.

Este problema é considerado NP-Complete (RAIDL, 1999), NP-difícil (SILVA; SOMA, 2003) e as restrições dimensionais, pesos, centro de gravidade, valores, orientações do posicionamento das caixas e outros, dificultam a obtenção da solução ótima. Portanto, justifica-se o emprego de técnicas heurísticas e metaheurísticas para resolução do problema, com o objetivo de maximizar o volume total de caixas utilizadas em relação ao volume disponível do contêiner, ou seja, ocupar o máximo volume do contêiner.

Neste trabalho apresentamos duas propostas para resolver de forma eficiente o PCC. Consideramos neste trabalho caixas heterogêneas, apenas um único contêiner e sem restrição quanto ao rotacionamento das dimensões das caixas.

Uma proposta é uma heurística diferenciada, onde apresentamos o conceito de carregamento por arranjos e em cada arranjo, propomos a formação de cuboides. Para facilitar o carregamento, inicialmente com o arranjo principal, espaço residual de altura, espaço residual de largura e espaço residual frontal.

Outra proposta deste trabalho para a resolução do PCC é uma metaheurística GRASP, utilizando como base o algoritmo heurístico proposto e a formulação de uma fase de melhoria local com o algoritmo GRASP, onde apresentamos pequenas mudanças.

Este trabalho está estruturado em 6 capítulos. No capítulo 1 apresentamos uma introdução que

consiste do PCC, mostrando a modelagem matemática existente na literatura.

No capítulo 2 apresentamos as técnicas de otimização existentes na literatura, ou seja, as heurísticas e metaheurísticas. Neste capítulo detalha-se o algoritmo heurístico construtivo e a metaheurística GRASP generalizado.

O capítulo 3 apresenta uma revisão bibliográfica das aplicações dos métodos de otimização da literatura, especificamente, o método heurístico de George e Robinson (1980) e as principais heurísticas e metaheurísticas utilizadas.

O capítulo 4 mostramos a primeira técnica de otimização proposta, um algoritmo heurístico construtivo proposto e os testes realizados com caixas heterogêneas. Neste capítulo detalha-se, passo a passo, o algoritmo heurístico proposto que é comparado com outros algoritmos apresentados na literatura.

No capítulo 5 mostramos, o algoritmo GRASP para o PCC, com todos os testes realizados para a heurística proposta neste trabalho. Os dados dos testes utilizados neste trabalho são apresentados no apêndice. No capítulo 6 apresentamos as conclusões do trabalho. No final deste trabalho apresenta as referências bibliográficas pesquisadas e finalmente um apêndice com os dados de todas as instâncias testadas.

1.1 O Problema de Carregamento de Contêiner

O PCC é um problema altamente complexo no campo da pesquisa operacional e que encontra muitas aplicações práticas, como o clássico problema da mochila. Se o carregamento for feito de forma a maximizar o espaço ocupado pelas caixas, pode-se ter um grande impacto econômico e ecológico, como facilitar a movimentação, a armazenagem , o transporte da carga, reduzir o número de volume a ser manipulado, minimizar o tempo de operação de embarque e desembarque, reduzir os custos e diminuir as avarias e os roubos das mercadorias a serem transportadas.

O PCC consiste em carregar um número conhecido de caixas de tipos distintos e dimensões conhecidas dentro de contêineres, de modo a utilizar o espaço do contêiner da melhor maneira possível, ou seja, de modo a diminuir os espaços não ocupados.

As caixas podem ser homogêneas (somente um tipo de caixa), fracamente heterogêneas (poucos tipos de caixas e com muitas caixas de cada tipo) e fortemente heterogêneas (muitos tipos de caixas

e poucas caixas de cada tipo).

Ao se modelar o PCC, pode-se levar ou não em consideração a estabilidade do carregamento, a resistência ou fragilidade das caixas, limitação de peso da carga, múltiplos destinos da carga, etc. Todas estas considerações dificultam a solução do PCC e têm motivado o estudo e o desenvolvimento de diversos métodos aproximados para resolver este problema.

Atualmente os tamanhos dos contêineres obedecem a padrões internacionais, onde os de 40 pés (padronizada em: 12,04m de comprimento, 2,32m de largura e 2,38m de altura) são os mais utilizados, possibilitando assim o desenvolvimento tecnológico e logístico. Foram desenvolvidos equipamentos específicos para o manuseio de cargas, como guindastes, empilhadeiras e diversos tipos de contêineres para cada carga específica. Entre os inúmeros tipos de contêineres existentes os mais utilizados são:

- Dry Box: É um tipo de contêiner muito usado, totalmente fechado com portas somente nos fundos e sendo adequado para transportar cargas secas, como roupas, móveis, calçados, etc.
- Open Top: É um tipo de contêiner totalmente fechado, com aberturas no teto, usado para transporte de cargas como produtos agrícolas.
- Flat Rack: Contêiner sem as paredes laterais e sem teto. É adequado para cargas pesadas e grandes.
- Reefer: É totalmente fechado com portas no fundo. É apropriado para cargas que necessitam de controle de temperatura.
- Contêineres Modulares: São os tipos de contêineres que servem de escritórios, dormitórios, guaritas, ambulatórios médicos, sanitários, depósitos, estandes para feiras e eventos, etc. Os tamanhos variam de 2,00m a 6,00m de comprimento por 2,25m de largura e 2,50m de altura.

O mesmo desenvolvimento ocorreu com o transporte de carregamento de contêineres, podendo ser por via terrestre, ferroviário, aéreo e marítimo, possibilitando assim a redução dos custos e permitindo um maior controle do fluxo da carga.

A organização das cargas dentro dos contêineres pode ser através de paletes, em que as caixas são empilhadas em uma base de metal ou madeira para depois serem colocadas dentro dos contêineres, ou podem simplesmente serem distribuídas diretamente dentro do contêiner através de um fator logístico

importante e de alto custo. Em geral os métodos manuais utilizados para o carregamento de contêiner não apresentam um bom desempenho.

Com uma melhor utilização do espaço nos contêineres é possível obter uma redução nos custos e tempo de carregamento e descarregamento das caixas. Antes de escolher um contêiner é necessário verificar qual o melhor tamanho e a melhor capacidade a ser utilizada, que varia de acordo com o produto. Normalmente os contêineres de 20 pés (padronizada em: 5,94m de comprimento, 2,37m de largura e 2,28m de altura) é utilizado para cargas mais pesadas e menos volumosas e os contêineres de 40 pés (padronizada em: 12,04m de comprimento, 2,32m de largura e 2,38m de altura) é utilizada para cargas mais leves e mais volumosas (VENDRAMINI, 2007).

1.2 Modelagem Matemática

1.2.1 *Introdução*

Nesta seção apresentamos uma formulação matemática e a aplicação para o PCC, mostrando um modelo analítico, que utiliza variáveis inteiras e binárias. O objetivo é mostrar a complexidade do problema em termos de programação matemática, justificando a aplicação dos métodos heurísticos e metaheurísticos, para encontrar uma solução que maximiza o volume ocupado dentro do contêiner.

1.2.2 *Modelagem Matemática de Chen, Lee e Shen (1995)*

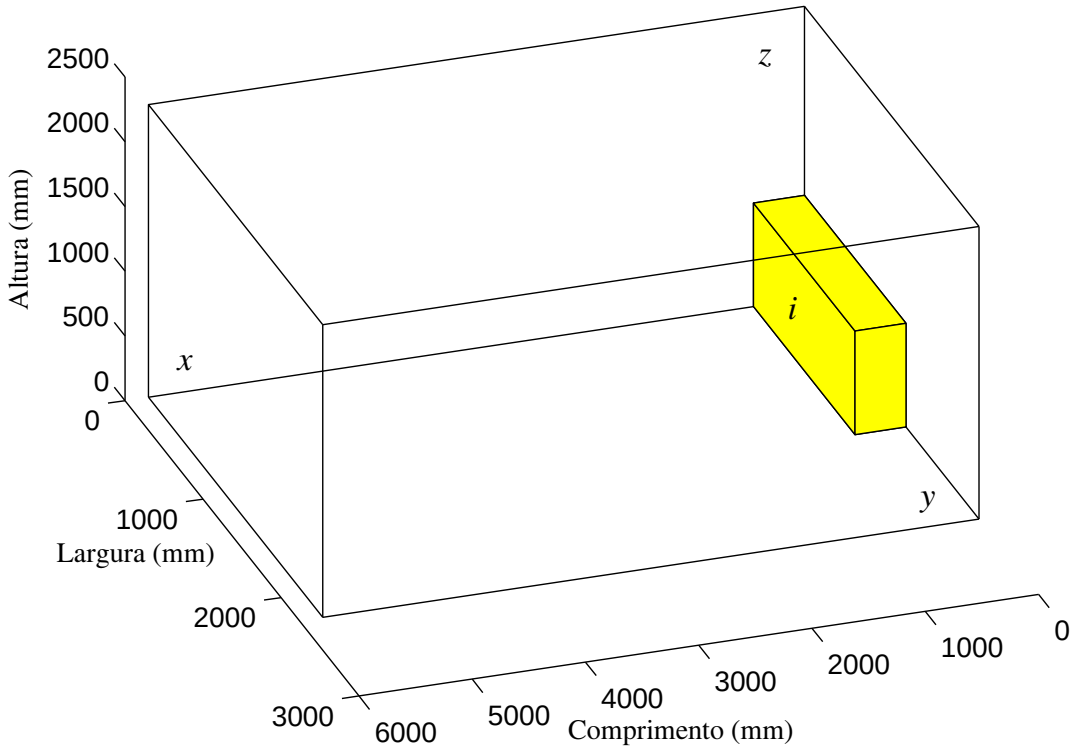
Chen, Lee e Shen (1995) desenvolveram um modelo analítico para o PCC utilizando variáveis inteiras, onde são consideradas caixas retangulares e de tamanhos não uniformes. Nesse modelo matemático, o objetivo é encontrar uma solução que minimize o espaço desperdiçado do contêiner e minimize o número de contêineres necessários para carregar todas as caixas.

Para entender melhor o modelo matemático proposto por Chen, Lee e Shen (1995), consideramos o carregamento de apenas um contêiner e permissão para rotacionar as caixas em duas de suas dimensões, isto é, em todos os eixos exceto a altura da caixa.

Na Figura 1, mostra que a largura do contêiner será o eixo (y) e tem a dimensão de maior valor da

caixa, enquanto o comprimento do contêiner (x) tem a dimensão de menor valor da caixa e por fim a altura do container será o eixo (z).

Figura 1: Carregamento da caixa i na origem do contêiner.



Fonte: Informações da pesquisa da autora.

As dimensões das caixas e do contêiner são conhecidas, bem como a quantidade de caixas a serem carregadas. As caixas são independentes e são posicionadas ortogonalmente no contêiner, ou seja, são posicionadas paralelas ou perpendicularmente aos eixos. As caixas podem sofrer rotações no posicionamento, de tal forma que a largura e o comprimento da caixa não sejam posicionados obrigatoriamente em paralelo à largura e ao comprimento do contêiner, respectivamente, contudo a altura da caixa sempre será posicionada em paralelo ao eixo z da altura do contêiner.

As notações listadas abaixo são necessárias para o entendimento do modelo matemático:

N - número de caixas disponíveis para o carregamento;

S_i - Variável binária que indica se a caixa foi colocada no contêiner. Quando isso ocorre, $S_i = 1$, caso contrário, $S_i = 0$;

(L, W, H) - Vetor que indica comprimento, largura e altura do contêiner, respectivamente;

(p_i, q_i, r_i) - Vetor que indica comprimento, largura e altura da caixa, respectivamente;

(x_i, y_i, z_i) - Vetor que indica alocação da caixa pelo canto inferior esquerdo traseiro;

(l_{xi}, l_{yi}, l_{zi}) - Vetor binário que indica qual eixo está em paralelo com o comprimento da caixa. Como a altura da caixa sempre está em paralelo com a altura do contêiner, podemos trabalhar com o vetor $(l_{xi}, l_{yi}, 0)$;

(w_{xi}, w_{yi}, w_{zi}) - Vetor binário que indica qual eixo está em paralelo com a largura da caixa. Como a altura da caixa sempre está em paralelo com a altura do contêiner, esse vetor pode ser $(w_{xi}, w_{yi}, 0)$;

(h_{xi}, h_{yi}, h_{zi}) - Vetor binário que indica qual eixo está em paralelo com a altura da caixa. Porém como a altura da caixa é fixa sempre em paralelo com a altura do contêiner, então esse vetor será fixo $(0, 0, 1)$;

Ainda existem outras variáveis que são usadas para indicar o posicionamento das caixas em relação a outras caixas:

a_{ik} - Caso seja 1, indica que a caixa i está à esquerda da caixa k;

b_{ik} - Caso seja 1, indica que a caixa i está à direita da caixa k;

c_{ik} - Caso seja 1, indica que a caixa i está atrás da caixa k;

d_{ik} - Caso seja 1, indica que a caixa i está à frente da caixa k;

e_{ik} - Caso seja 1, indica que a caixa i está abaixo da caixa k;

f_{ik} - Caso seja 1, indica que a caixa i está acima da caixa k;

O problema é então formulado conforme abaixo, onde se observa que o objetivo é minimizar o espaço não preenchido do contêiner:

Função objetivo:

$$min \;\; v = \qquad L.W.H - \sum_{i=1}^{N} p_i.q_i.r_i.s_i \tag{1}$$

Sujeito a:

Restrições de desigualdade:

(Evitar sobreposições de caixas no contêiner)

$$x_i + p_i.l_{xi} + q_i.w_{xi} + r_i.h_{xi} \leq x_i + (1 - c_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{2}$$

$$x_k + p_k.l_{xk} + q_k.w_{xk} + r_k.h_{xk} \leq x_i + (1 - d_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{3}$$

$$y_i + p_i.l_{yi} + q_i.w_{yi} + r_i.h_{yi} \leq y_k + (1 - a_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{4}$$

$$y_k + p_k.l_{yk} + q_k.w_{yk} + r_k.h_{yk} \leq y_i + (1 - b_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{5}$$

$$z_i + p_i.l_{zi} + q_i.w_{zi} + r_i.h_{zi} \leq z_k + (1 - e_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{6}$$

$$z_k + p_k.l_{zk} + q_k.w_{zk} + r_k.h_{zk} \leq z_i + (1 - f_{ik}).M, \quad \forall i, k, \quad \text{onde } i < k \tag{7}$$

(Garantir que o par de caixas avaliados nas equações acima esteja dentro do contêiner).

$$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \geq s_i + s_k - 1, \quad \forall i, k, \quad \text{onde } i < k \tag{8}$$

(Garantir que o posicionamento das caixas obedeça às limitações físicas dimensionais do contêiner).

$$x_i + p_i.l_{xi} + q_i.w_{xi} + r_i.h_{xi} \leq L + (1 - s_i), \quad \forall i \tag{9}$$

$$y_i + p_i.l_{yi} + q_i.w_{yi} + r_i.h_{yi} \leq W + (1 - s_i), \quad \forall i \tag{10}$$

$$z_i + p_i.l_{zi} + q_i.w_{zi} + r_i.h_{zi} \leq H + (1 - s_i), \quad \forall i \tag{11}$$

M é um número inteiro arbitrário e grande.

Restrições de igualdade:

Os vetores binários que identificam o posicionamento das caixas em relação aos eixos são variáveis dependentes que se relacionam conforme abaixo:

$$l_{xi} + l_{yi} + l_{zi} = 1 \tag{12}$$

$$w_{xi} + w_{yi} + w_{zi} = 1 \tag{13}$$

$$z_{xi} + z_{yi} + z_{zi} = 1 \tag{14}$$

$$l_{xi} + w_{xi} + h_{xi} = 1 \tag{15}$$

$$l_{yi} + w_{yi} + h_{yi} = 1 \tag{16}$$

$$l_{zi} + w_{zi} + h_{zi} = 1 \tag{17}$$

Ainda que o problema possa ser simplificado com manipulações algébricas, continua sendo um problema complexo de solução não trivial, sendo classificado como um problema de classe NP-Complete (NP completo).

Em Chen, Lee e Shen (1995) descreveram que a formulação matemática leva a uma solução ótima para o PCC utilizando um modelo de programação linear inteiro mista, mas esta não se apresenta como uma solução eficiente para um problema de grande porte, com um número elevado de caixas, já que os números de variáveis e de restrições tornam-se muito grandes.

Li, Tsai e Hu (2003) apresentaram um refinamento na modelagem matemática, onde adapta o método formulado por Chen, Lee e Shen (1995) para resolver o problema utilizando-se das mesmas variáveis, caracterizadas pela mistura de variáveis contínuas e binárias. Este método tem a vantagem de garantir a solução ótima global, um método computacional mais eficiente, adaptando as variáveis de sobreposição (equação 2) a (equação 7) de Chen, Lee e Shen (1995) e com o objetivo de minimizar o volume ocioso desperdiçado do contêiner.

A inclusão de novas restrições, tais como o peso de carga, valores e outros, aumentaria a dificuldade de se encontrar uma solução. Devido à complexidade da formulação mostrada acima, Rodrigues (2005) afirma que as heurísticas e metaheurísticas são mais eficientes, simples e possíveis de serem aplicados para o problema de carregamento de contêiner.

Capítulo 2

ALGORITMOS HEURÍSTICOS E AS METAHEURÍSTICAS

2.1 Introdução sobre as Heurísticas

As heurísticas são técnicas de otimização que geralmente encontram soluções de boa qualidade de problemas complexos. Deve-se observar que entre as décadas de 1960 e 1970, as heurísticas foram as técnicas de otimização mais usadas e com maior sucesso para resolver problemas complexos no campo da otimização matemática, especialmente para aqueles problemas não lineares, discretos e não convexos.

A maioria das heurísticas encontra soluções de boa qualidade de problemas altamente complexos em tempos computacionais relativamente rápidos. Adicionalmente, a maioria das heurísticas é simples de entender e também de implementar computacionalmente. Entretanto, as técnicas heurísticas renunciam, pelo menos do ponto de vista teórico, de encontrar a solução ótima global de um problema complexo. Em problemas de grande porte e complexos, as técnicas heurísticas raramente encontram as soluções ótimas. Neste trabalho consideramos um problema como sendo complexo, quando existe grande dificuldade para encontrar a solução ótima global, devido principalmente a que o problema apresenta a característica da explosão combinatória, quando o tamanho do problema cresce e/ou, porque o problema apresenta uma modelagem matemática complexa (variáveis inteiras ou discretas, função objetivo não linear e não diferenciável, restrições não lineares, região factível não convexa, etc.).

Uma técnica heurística pode ser muito simples como, por exemplo, o uso de bom senso ou a experiência de um especialista ou pode ser muito sofisticada, geralmente envolvendo a solução de

modelos matemáticos relaxados em relação ao modelo original.

É interessante usar técnicas heurísticas de otimização nos seguintes casos (GLOVER; KOCHENBERGER, 1989):

1. Quando não existe um método exato de otimização para resolver o problema sob análise.

2. Quando a solução ótima não é muito importante do ponto de vista prático por diferentes motivos como, por exemplo, a existência de muitas soluções ótimas locais de qualidade muito próxima da solução ótima global.

3. Quando os dados usados apresentam incerteza elevada como acontece em muitos problemas relacionados com o planejamento.

4. Quando existem limitações de tempo de processamento para encontrar a solução procurada e quando existem problemas de memória para armazenamento de dados. Problemas desse tipo podem aparecer em problemas de aplicação em tempo real.

5. Quando se pretende encontrar uma boa solução inicial para ser usada como ponto de partida na aplicação de uma técnica de otimização mais sofisticada como, por exemplo, quando pretendemos usar um algoritmo branch and bound.

Não é uma tarefa simples classificar as técnicas heurísticas de otimização. Uma proposta de classificar as técnicas heurísticas é a seguinte (GLOVER; KOCHENBERGER, 1989): algoritmos heurísticos construtivos, algoritmos de decomposição, algoritmos de divisão, algoritmos de redução, algoritmos de manipulação do modelo matemático e algoritmos de busca através de vizinhança (*steepest descent heuristic*). Neste trabalho, não pretendemos realizar uma apresentação detalhada dos diferentes tipos de algoritmos ou técnicas heurísticas de otimização. Assim, apresentamos de forma mais detalhada apenas dois desses algoritmos ou técnicas heurísticas de otimização e que são muito importantes no desenvolvimento das propostas de otimização desenvolvidos neste trabalho, assim como na compreensão das metaheurísticas. Essas heurísticas são as seguintes: (1) o algoritmo heurístico construtivo e, (2) o algoritmo heurístico de busca através de vizinhança.

2.1.1 *O Algoritmo Heurístico Construtivo*

O algoritmo heurístico construtivo (AHC) é uma das técnicas heurísticas de otimização mais usadas para resolver problemas complexos e ainda é muito usado isoladamente ou integrado a metaheurísticas mais sofisticadas. O mais popular dos algoritmos heurísticos construtivos é o tipo guloso (*greedy*). O AHC do tipo guloso é o único tipo de AHC que é analisado neste trabalho e chamaremos a esse algoritmo simplesmente de AHC.

O AHC é uma técnica de otimização que, em um processo passo a passo, gera uma solução geralmente de boa qualidade de um problema complexo. Em cada passo o AHC escolhe um elemento ou componente da solução que está sendo construída e no último passo termina de gerar uma solução factível. O elemento ou componente da solução que é escolhido em cada passo do AHC é identificado usando um indicador de sensibilidade que identifica a componente mais interessante a ser incorporado na solução em construção. Assim, a diferença fundamental entre os AHCs usados para resolver um mesmo problema está no indicador de sensibilidade usado.

Um AHC pode assumir a seguinte forma genérica:

1. Armazenar os dados do problema e escolher o indicador de sensibilidade a ser usado. Escolher os componentes que podem ser incorporados na solução em construção (geralmente o processo é iniciado sem componentes).

2. Verificar se a solução em construção já representa uma solução factível. Caso seja factível então pare o processo porque foi encontrada a solução factível procurada. Em caso contrário ir ao passo 3.

3. Usando a solução em construção, resolver o problema que permite identificar o indicador de sensibilidade de todos os componentes do problema que ainda não foram incorporados na solução em construção.

4. Usando a informação dos indicadores de sensibilidade encontrados no passo anterior identificar o componente que deve ser incorporado na solução em construção. Adicionar o componente identificado na solução em construção e voltar ao passo 2.

A ilustração de um AHC que iremos mostrar é para o problema do caixeiro viajante (*traveling salesman problem*). Nesse problema, o caixeiro viajante deve fazer um tour partindo de uma cidade

origem, passando por cada uma das cidades uma única vez e voltando para a cidade de origem. Pretende-se encontrar o tour de distância mínima. Os dados do problema são os pontos cartesianos das cidades e existem distâncias euclidianas ligando todas as cidades. O modelo matemático desse problema é um problema de programação linear binária com um número muito elevado de variáveis e restrições. A variável de decisão binária é x_{ij} em que $x_{ij} = 1$ se o caixeiro viajante escolhe o percurso da cidade i para a cidade j e $x_{ij} = 0$ se o caixeiro viajante não escolhe esse percurso. Nesse contexto, dois AHCs com níveis de sofisticação muito diferentes, para o problema do caixeiro viajante são os seguintes:

1. AHC simples:

    - No passo 1 escolhemos a cidade inicial de forma aleatória.
    - No passo 3 o indicador de sensibilidade usado é a distância mais próxima. Assim, se o caixeiro viajante se encontra na cidade p então se calcula a distância da cidade p para todas as cidades ainda não visitadas. Finalmente, a próxima cidade a ser visitada é aquela que se encontra mais perto de p e que ainda não foi visitada. Portanto, em cada passo se escolhe uma aresta do grafo usando o critério de menor distância e que no final do processo deve gerar um tour.

2. AHC sofisticado:

    - No passo 3 resolvemos o modelo matemático do problema do caixeiro viajante após usar a estratégia de relaxação. O modelo original do problema é de programação linear binária. Entretanto, se relaxamos as variáveis binárias para assumir a forma $0 \leq x_{ij} \leq 1$ então o problema transformado (relaxado) é um problema de programação linear (PL). Assim, após resolver o problema de PL podemos escolher o arco a ser incorporado na solução em construção como sendo aquele arco que apresenta o maior valor de x_{ij} na solução do PL e, obviamente, escolhido entre aqueles arcos que não geram subtour na solução em construção. Também, em cada passo do AHC um conjunto de variáveis x_{ij} já se encontram com valores conhecidos ($x_{ij} = 1$ para os arcos já incorporados na solução em construção). Também algumas variáveis podem ser fixados em $x_{ij} = 0$ (aquelas relacionadas com os arcos que geram subtour com arcos já incorporados na solução em construção). Outro nível de relaxação adicional pode ser encontrado eliminando as restrições que evi-

tam a geração de subtours. Neste caso o problema de PL a ser resolvido é muito mais simples, mas a qualidade do indicador de sensibilidade pode ficar comprometida.

Em resumo, os algoritmos heurísticos construtivos do tipo guloso apresenta as seguintes características:

- É um processo iterativo, onde em cada passo escolhe-se uma componente da solução em construção. O indicador de sensibilidade pode ser muito simples (a menor distância no caso do caixeiro viajante e cuja informação já se encontra na matriz de distâncias) ou muito sofisticado (resolver um problema de PL no caso do caixeiro viajante).
- Apenas no último passo se encontra uma solução factível. Antes disso não existe nada útil disponível em relação com o problema a ser resolvido. Esta característica nos lembra o algoritmo dual simplex em PL, onde apenas na última iteração encontramos um ponto extremo que é adicionalmente ótimo e antes disso existe apenas uma sequência de pontos infactíveis.

2.1.2 *O Algoritmo Heurístico de Busca Através de Vizinhança*

O Algoritmo heurístico de busca através de vizinhança (*steepest descent heuristic*) é significativamente diferente do algoritmo heurístico construtivo do tipo guloso. No AHC se gera apenas uma solução factível através de uma sequência de passos e usando um indicador de sensibilidade. No algoritmo heurístico de busca através de vizinhança, que chamaremos neste trabalho apenas como algoritmo SDH (do inglês *Steepest Descent Heuristic* para o problema de minimização), o processo é geralmente iniciado a partir de uma solução factível e na sequência são encontradas novas soluções factíveis percorrendo o espaço de busca e passando sempre para a melhor solução vizinha.

A terminologia usada na heurística SDH é diferente da terminologia usada na otimização clássica (programação linear, programação não linear, programação inteira, etc.) e às vezes pode ser significativamente diferente. Essa mesma terminologia é usada nas metaheurísticas. Assim, antes de analisar com detalhes a heurística SDH, devemos revisar parte da terminologia usada na heurística SDH e também nas metaheurísticas.

Terminologia Usada na Heurística de Busca Através de Vizinhança

A seguir apresentamos os principais conceitos usados na heurística SDH e, na medida do possível, fazemos uma análise paralela com os conceitos usados na otimização clássica. Um detalhe fundamental a ser observado é que na otimização clássica se usa necessariamente uma modelagem matemática para resolver um problema de otimização. Assim, dado um problema como, por exemplo, o problema do caixeiro viajante, a primeira decisão na otimização clássica consiste em encontrar o modelo matemático desse problema. A seguir se resolve o modelo matemático usando uma técnica de otimização clássica. Para encontrar o modelo matemático a decisão mais importante é a escolha das variáveis de decisão.

A estratégia mais popular usada por uma técnica de otimização clássica consiste em resolver o modelo matemático do problema a partir de um ponto inicial (que pode ser factível ou infactível) o que significa em escolher valores específicos para as variáveis de decisão. A partir desse ponto inicial se gera uma sequência de outros pontos (factíveis ou infactíveis) até atingir a convergência para um ponto factível e ótimo (local ou global). Nesse tipo de análise o conceito de região factível é fundamental. A estratégia fundamental da heurística SDH pode ser resumida da seguinte forma:

- O processo de otimização é iniciado através de uma solução inicial (factível ou infactível) que passa a ser chamada de solução corrente.
- Deve-se definir uma estrutura de vizinhança. Assim, deve existir uma forma de identificar as soluções que são consideradas vizinhas da solução corrente. As soluções vizinhas podem ser factíveis ou infactíveis.
- Na heurística SDH se passa da solução corrente para a melhor solução vizinha.
- O processo termina quando todas as soluções vizinhas são de pior qualidade que a solução corrente.

Deve-se observar que para implementar a heurística SDH não necessariamente estamos obrigados a usar o modelo matemático do problema em análise. Na verdade a heurística SDH pode resolver problemas de otimização que não têm modelagem matemática e essa característica torna a heurística SDH, assim como as metaheurísticas, uma técnica de otimização relativamente distante da lógica de otimização usada na otimização clássica. Para uma adequada apresentação do funcionamento da

heurística SDH devemos definir alguns conceitos que apresentamos a seguir e que nem sempre são apresentados na otimização clássica ou podem ter significados diferentes.

A **codificação** ou **representação** de uma proposta de solução de um problema complexo representa a estratégia fundamental para entender o funcionamento da heurística SDH. Assim, a codificação de uma proposta de solução representa de forma inequívoca um elemento do espaço de busca do problema de otimização. O espaço de busca é formado por todas as propostas de solução que podem ser identificados usando a codificação escolhida para um determinado problema de otimização e usando as estratégias de vizinhança usadas para resolver o problema. Devemos lembrar que um aspecto crucial na otimização clássica é a escolha adequada das variáveis de decisão. Na heurística SDH um aspecto crucial e fundamental é a escolha de uma proposta de codificação eficiente.

A codificação na heurística SDH substitui as variáveis de decisão no problema de otimização clássica. Em determinados problemas o vetor de codificação de uma proposta de solução pode ter a mesma dimesão e a mesma estrutura que o vetor de variáveis de decisão (por exemplo, no problema da mochila), mas em outros casos pode ter forma e dimensões diferentes. Para mostrar a grande diferença que pode existir entre vetor de variáveis de decisão e vetor de codificação mostramos um exemplo do problema do caixeiro viajante para $n = 10$ cidades. Nesse caso, a proposta de codificação mais usada para mostrar a proposta de solução p_1 é a seguinte:

$p_1 =$

1	9	10	7	4	2	3	6	8	5

o que significa, que o caixeiro parte da cidade 1 e na sequência passa pelas cidades 9, 10, 7, 4, 2, 3, 6, 8, 5 e volta para a cidade de origem 1. Portanto, uma proposta de solução para o problema do caixeiro viajante pode ser codificado em um vetor de dimensão n. Os elementos do vetor de codificação mostram a sequência em que são visitadas as cidades.

A proposta de codificação mostrada para o problema do caixeiro viajante pode ser considerado extremamente eficiente pelos seguintes motivos:

- Qualquer proposta de solução em que aparecem os números de 1 a n no vetor de codificação representa uma solução factível (um tour). Particularmente, se os elementos do vetor de codificação de uma proposta de solução estão formados pelos números de 1 a n então não existe a possibilidade de formar subtours e, portanto, todas essas propostas de solução satisfazem trivialmente as restrições relacionadas com subtours na modelagem matemática tradicional (essas restrições são as mais numerosas).

- Qualquer proposta de solução em que aparecem os números de 1 a n, no vetor de codificação também, satisfazem as restrições na modelagem matemática tradicional exigindo que o caixeiro deve chegar e sair de uma cidade apenas uma vez. Finalmente, as restrições binárias também são satisfeitas de forma trivial.

- Se fossem gerados todos os vetores que representam propostas de solução em que cada vetor tem os elementos de 1 a n, em qualquer ordem então podemos gerar todos os tours possíveis. Em outras palavras, cada combinação dos números de 1 a n arranjados em um vetor, representa um tour no problema do caixeiro viajante. Assim, podemos concluir que se usamos a codificação apresentada anteriormente para o problema do caixeiro viajante e determinadas estruturas de vizinhança, então o espaço de busca é exatamente o mesmo que a região factível.

- O vetor de codificação de uma proposta de solução é pequeno, isto é, de dimensão n e cresce pouco com o tamanho do problema. O vetor de variáveis de decisão x na modelagem matemática tradicional do problema do caixeiro viajante é de dimensão $n(n-1)$ e cresce mais rapidamente que que o tamanho do vetor de codificação. Portanto, a forma e estrutura do vetor de codificação é muito diferente do vetor de variáveis de decisão. Finalmente, se pretendemos resolver o problema do caixeiro viajante usando uma heurística SDH ou uma metaheurística podemos descartar o uso da modelagem matemática usada na otimização clássica ou fingir que não conhecemos essa modelagem matemática já que ela não é usada no processo de solução.

Uma proposta de solução está *adequadamente codificada* quando, a partir dessa informação, é possível encontrar o valor da função objetivo (ou seu equivalente) do problema e determinar se a proposta é factível ou infactível. Uma proposta de solução está *eficientemente codificada*, para sua utilização em uma heurística SDH ou em uma metaheurística quando os operadores desse algoritmo podem ser eficientemente implementados. Deve-se observar que podem existir várias formas de codificar uma proposta de solução, assim como podem existir várias formas de definir as variáveis de decisão na formulação da modelagem matemática tradicional. Assim, por exemplo, no problema do caixeiro viajante podemos escolher um vetor de codificação com a mesma forma e estrutura do vetor de variáveis de decisão. Entretanto, uma análise rápida nos permite concluir, que esse tipo de proposta de codificação é menos eficiente, que a codificação proposta anteriormente através de um vetor de dimensão n.

Para algumas metaheurísticas como, por exemplo, nos algoritmos genéticos e evolutivos em geral a proposta de codificação deve permitir uma implementação eficiente dos operadores existentes na

metaheurística. No caso do algoritmo genético, uma proposta de codificação eficiente, deve permitir implementar de forma eficiente os operadores de seleção, recombinação e mutação.

Resumindo, a proposta de codificação escolhida para resolver um problema complexo usando a heurística SDH ou uma metaheurística deve permitir identificar de maneira única um elemento do espaço de busca, deve permitir encontrar o valor da função objetivo ou equivalente e deve permitir verificar se a proposta de solução é factível ou infactível. Adicionalmente, a proposta de codificação deve permitir definir e implementar de forma adequada as estruturas de vizinhança e/ou implementar de forma eficiente os operadores existentes na metaheurística.

O **espaço de busca** está estreitamente relacionado com a proposta de codificação. Os elementos do espaço de busca (propostas de solução) são todas as propostas de soluções que podem ser encontradas implementando as estruturas de vizinhanças ou os operadores existentes na heurística SDH ou nas metaheurísticas. O espaço de busca pode compreender apenas uma parcela da região factível, exatamente a região factível ou parcelas da região factível e infactível do problema que está sendo resolvido. Os elementos do espaço de busca estão implicitamente especificados quando escolhemos a codificação de uma proposta de solução e a forma de realizar as transições a partir de uma solução inicial (ou conjunto de soluções iniciais). Assim, as estruturas de vizinhança ou operadores da metaheurística junto com a proposta de codificação definem os elementos do espaço de busca. Para de ilustração, podemos mencionar que no problema do caixeiro viajante o espaço de busca é exatamente igual à região factível se escolhemos a proposta de codificação mencionada anteriormente através de um vetor de dimensão n e se realizamos as transições através de vizinhança tipo $k - opt$ ou operadores genéticos que permitem gerar descendentes representados por um vetor de dimensão n e com os números de 1 a n armazenados nesse vetor de codificação.

A heurística SDH realiza a busca através de vizinhança, isto é, a partir da solução corrente, passa-se para a melhor solução vizinha. Assim, precisamos definir a vizinhança da solução corrente.

Seja p_k a codificação da solução corrente de um problema complexo que está sendo resolvido pela heurística SDH. Uma *solução vizinha* de p_k é qualquer proposta de solução que pode ser encontrada com a implementação de um mecanismo de perturbação da solução corrente. As soluções vizinhas são armazenadas no conjunto $N_k(p)$. O número de soluções vizinhas de p_k é conhecido como a cardinalidade de $N_k(p)$ e denotada através da notação $|N_k(p)|$. Assim, para definir a vizinhança da solução corrente apenas precisamos definir o mecanismo de perturbação.

Supor que temos disponível a solução corrente p_k e conhecemos o mecanismo de perturbação para encontrar os elementos de $N_k(p)$. Nesse contexto, os elementos de $N_k(p)$ podem representar

soluções factíveis ou infactíveis para o problema que está sendo resolvido. A ideia central relacionada com a definição de $N_k(p)$ deveria ser, evitar que os elementos de $N_k(p)$ sejam factíveis e infactíveis. Assim, na medida do possível devemos usar estruturas de vizinhança em que os elementos de $N_k(p)$ sejam apenas propostas de solução factíveis. Entretanto, esse tipo de proposta nem sempre é possível e depende de muitos fatores tais como a forma de codificação proposta, o tipo de problema que está sendo resolvido e a forma de definição de vizinhança. Quando não existe possibilidade de gerar apenas soluções vizinhas, então a heurística SDH deve decidir a forma de tratamento das soluções vizinhas. Na maioria das aplicações as propostas de solução infactíveis são descartadas, mas em outras aplicações podem ser consideradas para análise e avaliadas considerando, por exemplo, uma penalização dessas soluções na função objetivo. Em resumo, as diferentes formas de definir a vizinhança da solução corrente diferem apenas no mecanismo de perturbação usado.

Na ilustração a seguir, mostramos a vizinhança mais usada para resolver o problema do caixeiro viajante, isto é, a vizinhança $2-opt$. O mecanismo de perturbação que permite definir a vizinhança $2-opt$ é a seguinte: é vizinho da solução corrente toda proposta de solução encontrada rotacionando uma parcela dos elementos do vetor de codificação da solução corrente. Assim, escolhemos duas cidades do vetor de codificação e rotacionamos a parcela do vetor de codificação que se encontra entre as duas cidades selecionadas. Assim, seja p_1 a seguinte solução corrente:

$p_1 =$

			●				●		
1	9	10	7	4	2	3	6	8	5

Na solução corrente mostrada anteriormente, se escolhemos as cidades 7 e 6, então podemos gerar a seguinte proposta de solução vizinha:

$p_2 =$

1	9	10	6	3	2	4	7	8	5

Em relação com a vizinhança $2-opt$ mostrada anteriormente para o problema do caixeiro viajante podemos fazer as seguintes observações:

- O número de soluções vizinhas da solução corrente é igual a $n(n-1)$. Assim, para o exemplo mostrado, a cardinalidade de $N_1(p)$ é igual a $N_1(p) = n(n-1) = 10(9) = 90$.
- É muito simples e rápido avaliar a qualidade de uma solução vizinha. Para a solução vizinha mostrada anteriormente, a variação da função objetivo Δv pode ser encontrada usando a

seguinte relação:

$$\Delta v = d_{10,6} + d_{7,8} - d_{10,7} - d_{6,8}$$

Assim, a qualidade da solução vizinha pode ser determinada após 4 somas algébricas. Se $\Delta v < 0$ significa que a solução vizinha é de melhor qualidade. Entretanto, na maioria dos problemas de otimização da engenharia elétrica, a determinação da qualidade de uma solução vizinha pode requerer a resolução de um problema de programação linear, um problema de programação não linear, um sistema de equações lineares ou um sistema de equações não lineares (o conhecido problema de fluxo de carga).

- Quando n for relativamente grande, por exemplo $n = 1000$, o tamanho da vizinhança pode ser muito grande e a heurística SDH usaria muito tempo de processamento para realizar uma iteração simples.

Para cada tipo de problema é possível definir vários tipos de vizinhanças. Assim, a definição da vizinhança, junto com a proposta de codificação de uma proposta de solução, é um dos tópicos cruciais na formulação de uma heurística SDH eficiente.

Foi analisado com certo detalhe o conceito de codificação de uma proposta de solução quando pretendemos resolver um problema complexo usando uma heurística SDH ou uma metaheurística, porque consideramos que esse tópico é um dos mais críticos quando pretendemos resolver o PCC usando as metaheurísticas.

Após apresentar a terminologia usada na heurística SDH retomamos a análise da heurística de busca através de vizinhança (SDH). Assim, a forma genérica da heurística SDH assume a seguinte forma:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Definir a estrutura de vizinhança a ser usada o que caracteriza o espaço de busca.

2. Encontrar uma solução inicial p_o que se transforma na solução corrente p_c.

3. Identificar e avaliar todas as soluções vizinhas da solução corrente p_c e identificar a melhor solução vizinha p^{best}.

4. Se $f(p^{best}) < f(p)$, então a solução vizinha é melhor que a solução corrente e, portanto, fazer $p_c = p^{best}$ e voltar ao passo 3. Em caso contrário, pare o processo e a solução encontrada pela heurística SDH é p_c (geralmente um ótimo local).

Em relação com a heurística SDH, devem-se realizar as seguintes observações:

- O passo 1 é muito mais importante do que normalmente é considerado por muitos pesquisadores. Uma codificação eficiente é crucial, assim como a caracterização da estrutura de vizinhança. Nesse passo também deve ser decidido como devem ser tratadas as propostas de solução vizinhas que são infactíveis. Assim, é de responsabilidade do pesquisador escolher ou definir a codificação a ser usada, do tipo de vizinhança escolhido e a forma de tratar as soluções vizinhas infactíveis.

- No passo 2 a heurística SDH exige apenas uma solução inicial. Assim, essa solução inicial pode ser encontrada de forma trivial escolhendo, por exemplo, uma solução gerada de forma aleatória ou pode ser gerada de forma sofisticada usando, por exemplo, um AHC conhecido e eficiente para o tipo de problema em análise (um ótimo local). Deve-se observar que em muitos problemas pode ser muito difícil encontrar uma solução inicial factível. Portanto, em geral, a solução inicial p_o pode ser factível ou infactível. Caso seja infactível, então devemos mudar a estratégia de qualidade para avaliar as soluções vizinhas. A estratégia mais popular consiste em penalizar a função objetivo das propostas de solução infactíveis. Também, no passo 2 é exigida apenas uma solução inicial.

- No passo 3, a heurística SDH exige avaliar a qualidade de todas as soluções vizinhas e identificar a melhor solução vizinha. Esse passo pode exigir muito tempo de processamento, especialmente em determinados problemas em que avaliar a qualidade de uma solução vizinha pode exigir resolver um problema relativamente complexo como, por exemplo, um problema de programação linear ou não linear. Mesmo que a avaliação da qualidade de uma solução vizinha seja relativamente rápida como, por exemplo, no problema do caixeiro viajante, o passo 3 pode exigir muito tempo de processamento. Por exemplo, no problema do caixeiro viajante para $n = 10000$ o passo 3 pode exigir um tempo de processamento muito elevado para decidir uma simples mudança em quatro arestas na solução corrente (duas arestas que entram e duas arestas que saem).

- No passo 4 a heurística SDH termina o processo se a melhor solução vizinha for de pior qualidade que a solução corrente. Assim, se a vizinhança for definida de forma inadequada então a heurística SDH pode terminar encontrando uma solução ótima local de pobre qualidade. Também, devemos observar que quando a heurística SDH converge então a solução corrente é também a incumbente. Essa característica não acontece na maioria das metaheurísticas, exceto na metaheurística de busca em vizinhança variável (VNS, do inglês *Variable Neighborhood Search*).

- Para fins de ilustração, uma heurística SDH para o problema do caixeiro viajante pode assumir a seguinte forma: (1) a codificação escolhida seria a apresentada neste trabalho, isto é, um vetor de dimensão n com os números de 1 a n na estrutura interna, a função objetivo seria a mesma da modelagem matemática tradicional, isto é, a soma de distâncias do tour, a vizinhança usada seria a $2 - opt$ e não apareceriam soluções vizinhas infactíveis, (2) a solução inicial pode ser gerada de forma aleatória ou usando o AHC mais simples mostrado anteriormente e, (3) a avaliação da qualidade das soluções vizinhas seria realizada calculando as quatro somas algébricas mostrada anteriormente.

O algoritmo heurístico construtivo e a heurística SDH foram analisados com certo detalhe porque a estrutura básica da maioria das metaheurísticas podem ser interpretadas apenas como uma generalização de um AHC, uma generalização da heurística SDH ou uma generalização conjunta do AHC e da heurística SDH.

Terminamos fazendo uma diferenciação clara entre o AHC de tipo guloso e a heurística SDH. Assim, devemos observar os seguintes assuntos:

- Um AHC gera apenas uma solução factível através de uma sequência de passos guiado por um indicador de sensibilidade e apenas no último passo é encontrada uma solução factível. Como mencionado anteriomente, essa estratégia é muito parecida com o algoritmo dual simplex em programação linear onde apenas na última iteração é encontrada uma solução factível e ótima. Nas iterações previas existem disponíveis apenas propostas de solução infactíveis.

- Uma heurística SDH gera um conjunto de soluções factíveis a partir de uma solução inicial (geralmente factível), realizando transições através de soluções factíveis, percorrendo o espaço de busca e sempre passando para uma solução de melhor qualidade. O processo de busca termina quando todas as propostas de solução vizinhas são de pior qualidade. Essa estratégia

é muito parecida com a estratégia fundamental do algoritmo primal simplex em programação linear.

- Uma heurística SDH geralmente é muito mais complexa, sofisticada e muito mais demorada que um AHC. Portanto, espera-se que a heurística SDH encontre soluções de melhor qualidade que o AHC na tentativa de resolver um problema complexo. Assim, quando analisamos publicações especializadas e verificamos que um AHC é superior a uma heurística SDH em termos de qualidade de solução final encontrada na resolução do mesmo problema complexo, então estamos em um caso anormal ou atípico e deveria ser analisado em detalhe os motivos desse comportamento.

2.2 Introdução sobre as Metaheurísticas

A definição de uma metaheurística apresentada por Glover e Kochenberger (2003) é a seguinte: Metaheurísticas são técnicas de solução que gerenciam uma interação entre as estratégias de busca local e as estratégias de nível superior para criar um processo de otimização com capacidade de sair de soluções ótimas locais e realizar uma busca robusta através de espaço de busca. Alternativamente, podemos definir uma metaheurística somo sendo um processo de otimização representado por uma generalização e/ou integração do algoritmo heurístico construtivo do tipo guloso e a heurística de busca através de vizinhança de forma que seja possível encontrar soluções de qualidade, percorrendo de forma eficiente o espaço de busca. Em outras, palavras, uma metaheurística pode ser interpretada como uma generalização e/ou integração do AHC e da heurística SDH mostrados no item anterior.

Apresentamos as principais metaheurísticas usadas no campo da pesquisa operacional. Nessa apresentação um tanto resumida, apresentamos a metaheurística preservando as origens e a interpretação dos autores e, paralelamente, apresentamos essas metaheurísticas apenas como uma generalização e/ou integração do AHC e da heurística SDH. Obviamente, a metaheurística GRASP que é usada neste trabalho é apresentada de forma mais detalhada em um subitem.

2.2.1 *Simulated Annealing*

Simulated Annealing (SA) é uma das primeiras metaheurísticas que foi usado com muito sucesso na otimização de problemas complexos na pesquisa operacional. SA foi inventando após verificar que existiam muitas semelhanças entre a técnica de construção de cristais perfeitos, usando a técnica de annealing e a otimização de um problema complexo no campo da pesquisa operacional.

Existem muitas técnicas usadas na construção de cristais perfeitos e uma delas é a técnica de annealing. A técnica de annealing consiste em esquentar um material até temperaturas elevadas na qual existe muita movimentação molecular do material esquentado e, portanto, um novo arranjamento dos atómos do material. A partir desse estado, se o material for esfriado lentamente então a movimentação molecular também diminui. Se esse processo de diminuição for adequadamente controlado preservando o chamado quase equilibrio termodinâmico no qual a temperatura deve ser diminuída lentamente então existe grande possibilidade de que o material se transforme em um cristal perfeito. Se esse processo não é realizado de forma adequada, então esse material pode ser transformado em um cristal imperfeito, isto é, em um vidro.

Resumindo, usando o paralelismo ou semelhanças que existem entre a técnica de annealing na construção de cristais perfeitos e na otimização de problemas complexos no campo da pesquisa operacional, foi desenvolvido o algoritmo de *Simulated Annealing* que na formulação básica assume a seguinte forma (problema de minimização):

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Definir a estrutura de vizinhança a ser usada o que caracteriza o espaço de busca. Escolher os parâmetros de SA tais como o parâmetro chamado de temperatura inicial T_o, a temperatura final T_f ou um critério de parada, o número de tentativas de transição no primeiro nível de temperatura N_o, o parâmetro ρ que controla o número de tentativas de transição em cada nível de temperatura e o parâmetro β que controla a diminuição do parâmetro temperatura.

2. Encontrar uma solução inicial p_o que se transforma na solução corrente p_c e fazer $N_k = N_o$. $s = 0$.

3. Identificar e avaliar uma solução vizinha p_v escolhida aleatoriamente.

4. Se $f(p_v) < f(p_c)$ então a solução vizinha é melhor que a solução corrente e, deve-se realizar a transição, isto é, $p_c = p_v$ e ir ao passo 4. Em caso contrário, gere um número aleatório entre 0 e 1, $P(0,1) = random[0,1]$, e seja $\Delta f(p) = f(p_v) - f(p_c)$. Assim, se $exp[\frac{-\Delta f(p)}{T_k} > P(0,1)]$ então, deve-se realizar a transição e $p_c = p_v$ e, em caso contrário, a solução corrente é preservada. Ir ao passo 4.

5. $s = s + 1$. Se $s < N_k$, então ir ao passo 3. Em caso contrário ir ao passo 6.

6. Se o critério de parada foi cumprido, então pare. Em caso contrário, fazer $T_{k+1} = \beta T_k$ e $N_{k+1} = \rho N_k$. $k = k + 1$. Voltar ao passo 3.

Deve-se observar também que a técnica de annealing é uma técnica mais geral e não é usada apenas para a construção de cristais perfeitos. Essa técnica também é usada, por exemplo, na construção de condutores elétricos para tornar o condutor mais uniforme e maleável. Em relação com o algoritmo SA, devem-se realizar as seguintes observações:

- A relação $exp[\frac{-\Delta f(p)}{T_k}]$ sempre é positiva e varia entre 0 e 1. Por esse motivo faz sentido a comparação com um número aleatório $P(0,1)$, que também varia entre 0 e 1. Adicionalmente, se T_k é muito grande ou $\Delta f(p)$ é muito pequeno, então $exp[\frac{-\Delta f(p)}{T_k}$ assume um valor próximo de 1. Por outro lado, se T_k é muito pequeno ou $\Delta f(p)$ é muito grande, então $exp[\frac{-\Delta f(p)}{T_k}]$ assume um valor próximo de zero.

- Da análise anterior podemos concluir que se a solução vizinha tem um valor da função objetivo, que é de pior qualidade, que a solução corrente, mas de valor muito próxima e/ou se o processo se encontra nas fases iniciais (o valor do parâmetro de temperatura elevada), então existe uma probabilidade elevada de aceitar uma solução vizinha de pior qualidade. Por outro lado, se o valor da função objetivo da solução vizinha é de pior qualidade e muito distante do valor da função objetivo da solução corrente e/ou o processo se encontra nas fases finais (valor do parâmetro temperatura baixa) então a probabilidade de aceitar uma solução vizinha de pior qualidade é pequena. Em resumo, nas fases iniciais do processo podem ser aceitas soluções vizinhas de pior qualidade com mais facilidade (priorizando um processo de diversificação) e nas fases finais do processo raramente são aceitas soluções vizinhas de pior qualidade (priorizando o processo de intensificação). Finalmente, em qualquer estágio do processo, soluções vizinhas ligeiramente piores que a solução corrente têm maior probabilidade de serem aceitas comparadas com soluções vizinhas piores e muito distantes da solução corrente.

- A relação $exp[\frac{-\Delta f(p)}{T_k}]$ pode ser substituída por qualquer outra, com a única condição de que desempenhe um papel equivalente, isto é, priorize as soluções vizinhas de pior qualidade da forma mostrada nos passos anteriores.

Deve-se observar que o algoritmo SA básico apresentado anteriormente pode ser obtido apenas como uma generalização da heurística SDH. Comparando a heurística SDH e o algoritmo SA podemos verificar as seguintes diferenças fundamentais:

- A heurística SDH avalia todas as soluções vizinhas para identificar o vizinho de melhor qualidade. Assim, o passo 3 da heurística SDH é muito demorado. Por outro lado, SA escolhe aleatoriamente uma solução vizinha e decide se realiza a transição sem conhecer nem avaliar as outras soluções. Nesse contexto, nas fases iniciais SA realiza transições com maior intensidade que a heurística SDH, mas nas fases finais do processo ambas técnicas de otimização devem apresentar comportamento muito próximos.

- SA pode sair de um ótimo local ao aceitar de forma probabilística transições para soluções vizinhas de pior qualidade. Essa é a principal diferença entre a heurística SDH e a metaheurística SA.

- Obviamente SA apresenta um maior tempo de processamento e deve encontrar soluções de melhor qualidade que a heurística SDH.

2.2.2 *Tabu Search - Busca Tabu*

Tabu Search (TS) é uma metaheurística inventada por Glover e Kochenberger (2003). Ao contrário da maioria das metaheurísticas que usaram comportamentos ou características existentes em ramos do conhecimento distantes da otimização matemática, Glover e Kochenberger (2003) usou apenas conhecimento existente no campo da otimização matemática. Antes de inventar o TS, Glover e Kochenberger (2003) já era um pesquisador destacado em otimização clássica, especialmente nas técnicas de otimização de problemas de programação inteira.

A ideia central de Glover e Kochenberger (2003) é mostrar de que é possível inventar uma estratégia de busca inteligente percorrendo o espaço de busca de forma eficiente e seletiva. Nesse processo é fundamental integrar o processo de busca as estratégias de intensificação e de diversificação.

Nesse contexto, intensificar significa realizar uma busca mais intensa em torno da solução corrente, por exemplo, aumentando o tamanho da vizinhança ou melhorando a qualidade da vizinhança. Por outro lado, diversificar significa sair de uma região do espaço de busca e, de forma proposital, atingir uma região distante para novamente realizar algum processo de intensificação.

A formulação básica de TS que está inspirado fundamentalmente no uso da estratégia de intensificação assume a seguinte forma:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Definir a estrutura de vizinhança a ser usada o que caracteriza o espaço de busca. Identificar os atributos que devem ser proibidos e o critério de aspiração. Escolher os parâmetros do algoritmo, tais como a duração da lista tabu. Escolher o critério de parada.

2. Encontrar uma solução inicial p_o que se transforma na solução corrente p_c.

3. Identificar e avaliar todas as soluções vizinhas da solução corrente p_c, ordenar essas soluções vizinhas por qualidade sendo que a primeira da lista é a melhor solução vizinha p^{best}.

4. Realizar a transição para a solução vizinha, melhor classificada que não tem o atributo proibido ou se tem o atributo proibido, então satisfaz o critério de aspiração. Seja p_e a melhor solução vizinha escolhida, então fazer $p_c = p_e$.

5. Atualizar a incumbente e a lista de atributos proibidos. Se o critério de parada for satisfeito, então pare. Em caso contrário, voltar ao passo 3.

O algoritmo TS anteriormente apresentado é chamado de algoritmo TS básico que fundamentalmente usa memória de curto prazo, uma lista de atributos proibidos e um critério de aspiração, isto é, é uma estratégia de otimização que prioriza a intensificação.

Técnicas de otimização tipo TS mais complexas podem ser implementadas onde o TS básico funciona como um módulo de otimização integrado em uma estrutura TS mais complexa. Essas estruturas mais complexas podem ser idealizadas usando outros operadores existentes em TS tais como a diversificação, a memória baseada em frequência, a lista de soluções de elite, o *path relinking*, entre outros. Para uma análise detalhada sobre a teoria e aplicação de TS ver Glover e Kochenberger (2003).

Em relação ao TS básico apresentado anteriormente, deve-se mencionar as seguintes observações:

- A única diferença do algoritmo TS básico em relação com a heurística SDH é que TS realiza a transição para a melhor solução vizinha, que não tem atributo proibido ou se tem o atributo proibido então cumpre com o critério de aspiração. Em outras palavras, se todas as soluções vizinhas da solução corrente são piores que a solução corrente, então o algoritmo TS passa para a menos pior, mas montando uma estratégia para tentar evitar visitar uma solução já visitada.

- Como o algoritmo TS pode passar para uma solução de pior qualidade, e ao mesmo tempo apresenta uma estratégia gulosa, então deve existir uma estratégia para evitar voltar a visitar uma solução já visitada. Essa estratégia pode ser montada de forma explícita armazenando todas as soluções já visitadas (ou um conjunto menor representado pelas últimas soluções já visitadas) e verificando se a solução vizinha candidata a ser escolhida já foi visitada anteriormente ou pode ser armazenada de forma implícita armazenando apenas uma característica da solução visitada e chamada de atributo.

- O algoritmo TS básico não recomenda o armazenamento de soluções explícitas, devido a problemas de memória para armazenamento da informação e pelo tempo de processamento necessário para verificar se uma solução vizinha já foi visitada anteriormente e, portanto, se encontra armazenada. Entretanto, essa premissa que era muito válida na década de 1980 pode não ter a mesma validade atualmente devido ao incremento elevado de disponibilidade de memória dos computadores modernos e também ao aumento da velocidade de processamento desses computadores. Assim, para determinadas aplicações pode ser oportuno avaliar a possibilidade do uso de memória explícita.

- Na estratégia implícita é usado o conceito de atributo. O atributo relacionado com uma proposta de solução é algum tipo de informação que permite identificar alguma característica da proposta de solução. O tipo de atributo mais usado é o próprio mecanismo usado na transição e que também permite a identificação de uma solução vizinha da solução corrente. Podemos mostrar um exemplo usando o problema do caixeiro viajante e a estrutura de vizinhança $2-opt$. Nesse contexto, uma solução vizinha da solução corrente é identificada escolhendo duas cidades no vetor de codificação e rotacionando as cidades localizadas, entre essas cidades no vetor de codificação. Assim, seja a solução corrente p_c:

$p_c =$

			●				●		
1	9	10	7	4	2	3	6	8	5

Uma solução vizinha pode ser identificada, por exemplo, escolhendo as cidades 7 e 6, para encontrar a seguinte solução vizinha p_v (usando a vizinhança $2 - opt$):

$p_v =$

1	9	10	6	3	2	4	7	8	5

Se o algoritmo TS básico realiza uma transição da solução corrente p_c para a solução vizinha p_v, então as cidades 6 e 7 não podem ser usadas para gerar novas soluções vizinhas durante um número de iterações (a duração da lista tabu). Nesse contexto, o atributo proibido é o movimento reverso de duas cidades (6 e 7). Deve-se observar que ao proibir escolher as cidades 6 e 7 para gerar novas soluções vizinhas estamos tentando evitar, voltar a visitar a solução p_c já visitada.

- A proibição através de atributos tenta evitar voltar a uma solução já visitada. Entretanto, como essa proibição termina após k_p transições previamente especificada, então é possível no futuro visitar uma solução já visitada e nesse caso, acontece o fenômeno chamado de ciclagem e uma forma de atenuar esse problema é aumentando o valor de k_p. Por outro lado, todas as soluções vizinhas que compartilham um atributo proibido com uma solução já visitada também não poderiam ser visitadas. Nesse contexto, se o valor de k_p for muito grande, então podem existir muitas soluções vizinhas que não poderiam ser visitadas, porque compartilham os mesmos atributos de soluções já visitadas e que ainda se encontram proibidas. Por esse motivo, o valor de k_p não pode ser muito grande. Adicionalmente, muitas soluções vizinhas que compartilham atributos proibidos com soluções já visitadas podem ser de excelente qualidade e, eventualmente, a própria solução ótima global pode estar proibida. Para contornar esse problema foi inventada a estratégia de critério de aspiração.

- A ideia central do critério de aspiração é eliminar a proibição do atributo de uma solução vizinha de excelente qualidade, porque compartilha o mesmo atributo de uma solução já visitada. Assim, se uma solução vizinha satisfaz um critério de qualidade que permite suspeitar que seja uma solução ainda não visitada então eliminamos a proibição e realizamos a transição para essa

solução vizinha. Esse critério de qualidade está relacionado com o valor da função objetivo. Um critério extremo, por exemplo, consiste em especificar que uma solução vizinha cumpre com o critério de aspiração, se for melhor que a incumbente. Nesse caso, existe a certeza absoluta de que essa solução vizinha nunca foi visitada anteriormente, mas apresenta a desvantagem de que raras vezes pode ser acionada o que pode ser quase equivalente a não ter critério de aspiração. Geralmente, pode ser mais oportuno escolher um critério de aspiração menos exigente. Um exemplo pode ser o seguinte: se a solução vizinha é melhor que as soluções visitadas nas últimas k_{ul} iterações então cumpre com o critério de aspiração.

2.2.3 *O Algoritmo Genético*

O algoritmo genético (AG) é uma metaheurística inventada por Goldberg (1989) na década de 70 sendo que apenas na década de 80 teve sua aplicação de forma intensa para resolver problemas complexos no campo da pesquisa operacional. Para inventar o AG Holland encontrou semelhanças entre a forma de resolver um problema de otimização matemática e o processo de seleção natural e evolução das espécies. Na verdade, na natureza o processo de seleção natural e a evolução das espécies é a consequência de um processo de otimização estocástico que acontece em um determinado ambiente e em tempo real.

O processo de seleção natural e a evolução das espécies é um problema muito complexo para que seja imitado de forma adequada por um processo de otimização de um problema complexo do campo da pesquisa operacional. Assim, pode-se afirmar que o AG imita apenas uma parcela dos componentes que fazem parte do processo de seleção natural e de evolução das espécies. Algumas implementações do AG que não acompanham de forma adequada o processo de seleção natural e a evolução das espécies são as seguintes:

- Um cromossomo na genética é considerado como equivalente a uma proposta de solução no problema de otimização. Na verdade uma proposta de solução deveria ser equivalente a um indivíduo. Adicionalmente, na genética, nas espécies mais complexas não existe um cromossomo, mas uma cadeia cromossômica.

- Um cromossomo na genética na verdade é um par cromossômico formado por duas cadeias de informação, um herdado do pai e outro herdado da mãe. Um elemento do cromossomo é

um gene que tem dois valores definidos (alelos) um em cada cadeia cromossômica. As duas informações representam uma unidade de informação genética. Na maioria dos casos existem apenas dois tipos de alelos (nessa posição genética e para todos os indivíduos da espécie) na posição correspondente a um gene. Assim, esse tipo de informação pode ser armazenado de forma binária (0 ou 1). Entretanto foi provado que para uma determinada espécie, existem vários tipos de alelos entre os elementos da espécie. Por exemplo, o gene que determina a cor dos olhos nos humanos. Portanto, a codificação binária usada nos AG não representa de forma adequada aquilo que acontece na natureza.

- A informação genética presente em um gene define o fenótipo, isto é, a característica externa do indivíduo. Assim, por exemplo, um tipo de alelo (informação genética) define o tipo de sangue humano (Rh^+ ou Rh^-). Assim, existiria uma relação biunívoca entre tipo de alelo e fenótipo, o que permite montar uma estratégia de codificação (tipo de alelo) e decodificação (fenótipo) no AG. Entretanto, foi provado que em muitos casos apenas um tipo de alelo define várias características fenotípicas (fenômeno conhecido como pleitropia), e também vários alelos correspondentes a vários genes definem apenas uma característica fenotípica (fenômeno conhecido como poligênia). Assim, para esses casos não existe uma relação biunívoca entre informação genética e o fenótipo. Adicionalmente, o fenômeno de dominância na genética em que um alelo domina o outro alelo é muito importante na genética e não se encontra representado no AG.

- O fenômeno de *crossing over* é fundamental para a existência da diversidade genética entre os individuos de uma espécie. A diversidade genética é a chave fundamental na seleção natural e na evolução das espécies. O fenômeno de *crossing over* acontece quando uma célula reprodutiva se duplica e se divide. Assim, uma célula reprodutiva (que é uma unidade celular com duas cadeias cromossômicas) após o crossing over gera quatro cadeias cromossômicas independentes. Acontece que na separação, uma cadeia cromossômica (por exemplo, um espermatozóide) é formado por parcelas da cadeia cromossômica herdado do pai e da mãe, formado por um processo de recombinação bastante complexo. Em outras palavras, um espermatozóide é formado por pequenas parcelas da cadeia cromossômica do pai e da mãe, misturados por parcelas pequenas de cada um. Sem crossing over, um espermatozóide estaria formado apenas pela cadeia cromossômica herdada pelo pai ou pela mãe. Assim, um espermatozóide é uma meia célula que posteriormente pode se juntar a um óvulo para formar uma nova unidade celular (zigoto) e, portanto, um novo indivíduo. Assim, deve-se esclarecer que o fenômeno de crossing over e

a reprodução sexuada acontecem em instantes muito diferentes.

- No AG o fenômeno de crossing over é imitado pelo operador de recombinação. Pode-se verificar que esse processo de imitação não é muito adequado.

- O fenômeno da mutação acontece na natureza quando a composição genética de um gene é alterado (no processo de duplicação e separação de uma célula reprodutiva). Geralmente, esse processo de mutação pode gerar um indivíduo de pior qualidade, mas às vezes pode gerar indivíduos de melhor qualidade. Esse fenômeno é adequadamente imitado no AG.

- Em resumo, podemos afirmar que um AG imita de forma não adequada o processo de seleção natural e de evolução das espécies. Esse processo de imitação é mais crítico no fenômeno de crossing over e relativamente significativo na codificação binária. Entretanto, apesar de todas as observações realizadas anteriormente, o AG apresenta excelente desempenho na resolução de muitos problemas do campo da pesquisa operacional. Assim, sempre existiu essa dúvida entre os pesquisadores sobre esse aparente paradoxo, o que de alguma maneira explica a busca intensa de encontrar algoritmos inspirados na seleção natural e da evolução das espécies que incorporem de forma adequada o que acontece na natureza. Essa pode ser uma explicação para a grande diversidade de algoritmos desse tipo, que existe na literatura especializada e que podem ser denominados apenas como algoritmos evolutivos.

Um algoritmo genético básico assume a seguinte forma:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Escolher os parâmetros do algoritmo tais como o tamanho da população n_p, a taxa de recombinação ρ_r, a taxa de mutação ρ_m e o tipo de seleção. Escolher um critério de parada.

2. Gerar a população inicial, isto é, gerar um conjunto de n_p propostas de solução que se transforma na população corrente.

3. Avaliar a qualidade $f(p)$ de todos os elementos da população e atualizar a incumbente, se possível.

4. Se o critério de parada for satisfeito, pare. Em caso contrário, ir ao passo 5.

5. Implementar o operador de seleção.

6. Implementar o operador de recombinação.

7. Implementar o operador de mutação, atualizar a população corrente e voltar ao passo 3.

Os primeiros algoritmos genéticos usavam a codificação binária. Entretanto, os algoritmos genéticos modernos seguem a proposta de Michalewicz (1994) sugerindo que a codificação deve ser realizada seguindo a natureza e as características do problema. Assim, por exemplo, no problema da mochila a codificação pode ser a binária, no problema do caixeiro viajante a codificação mais popular é um vetor que mostra a sequência em que as cidades são visitadas. Geralmente, em problemas que apresentam uma modelagem matemática e a proposta consiste em usar essa modelagem matemática e codificar as variáveis de decisão desse modelo, então a proposta mais popular consiste em codificar as variáveis de forma binária, as variáveis inteiras e discretas usando a codificação inteira e as variáveis contínuas usando uma codificação real. Entretanto, em problemas altamente restritos geralmente se codifica apenas as variáveis binárias e inteiras, sendo que o valores exatos das variáveis contínuas são encontradas de forma exata resolvendo um subproblema auxiliar (problema de programação linear, problema de programação não linear, sistema de equações algébricas lineares ou sistema de equações algébricas não lineares). Esse é o caso da maioria dos problemas de otimização relacionados com o planejamento e a operação de sistemas de energia elétrica.

A população inicial, isto é, o conjunto inicial de propostas de solução geralmente é montada de forma aleatória. Entretanto, em algoritmos genéticos mais sofisticados, o conjunto inicial de proposta de solução pode ser gerado usando algoritmos heurísticos construtivos eficientes de forma a gerar soluções de qualidade diversificada.

O operador de seleção determina o número de participações que deve ter cada elemento da população (proposta de solução) na formação da nova população. Obviamente o número de participações deve ser um número inteiro. O operador de seleção é um dos operadores que foi mais discutido e analisado nas fases iniciais de aplicação dos algoritmos genéticos. Na proposta inicial do algoritmo genético, o operador de seleção era implementado usando a seleção proporcional. A seleção proporcional determina o número de participações de cada proposta de solução usando a seguinte relação:

$$Nd_i = \frac{n_p \; f_i(p)}{\sum_{i=1}^{n_p} f_i(p)} \tag{18}$$

em que Nd_i é o número de participações (descendentes) da proposta de solução i, $f_i(p)$ é o valor da função objetivo ou equivalente da proposta de solução i e n_p é o número de propostas de solução (tamanho da população). Os valores dos Nd_i não são inteiros e, portanto, precisamos usar uma técnica de integralização desses valores. Na proposta original do AG, esse processo é realizado usando uma técnica estocástica parecido com a roleta nos jogos de azar.

Para que a relação (equação 18) seja aplicável, o problema de otimização deve estar padronizado. Na versão inicial do AG isso significava que o problema deveria ser de maximização e os valores da função objetivo deveriam ser maiores ou iguais a zero. Sem esses requisitos a relação (equação 18) não é aplicável. Portanto, se um determinado problema de otimização não cumpre esses requisitos então esse problema deve ser padronizado, isto é, deve ser transformado para um problema equivalente ou mais ou menos equivalente. Geralmente é relativamente fácil transformar um problema de minimização em um problema de maximização, mas a necessidade de satisfazer simultaneamente o requisito de que $f_i(p) \geq 0$ nem sempre é possível. Nesse contexto usamos o conceito de função objetivo ou equivalente. Assim, se não for possível transformar um problema de minimização em um problema de maximização e ao mesmo tempo cumprir o requisito de que $f_i(p) \geq 0$ então usamos uma função objetivo que não preserva as características originais, mas pode ser considerado equivalente sem ser exatamente equivalente.

Para ilustrar o problema anterior consideremos um problema de maximização, mas aparecem valores de $f_i(p) < 0$, então esse problema não se encontra padronizado. Nesse contexto, devemos aplicar a chamada técnica de janelamento, que significa em transformar os valores da função objetivo para que todos eles sejam não negativos, isto é, escolhemos uma constante K de forma que $f_i^n(p) = f_i(p) + K \geq 0$. Assim, o operador de seleção é aplicado usando os novos valores de função objetivo (nesse caso chamamos a esse valor transformado de função objetivo equivalente para fins de implementação do operador de seleção). Uma análise detalhada mostra que a relação (equação 18) também é aplicável, se o problema for de minimização e todos os valores dos $f_i(p) \leq 0$. Assim, de forma alternativa, se um problema for de minimização e todos os valores da função objetivo são menores ou iguais a zero, então esse problema também se encontra padronizado.

A seleção proporcional foi criticada por vários motivos tais como a necessidade de que o pro-

blema se encontre padronizado, o aparecimento das chamadas supersoluções com valores de função muito superiores aos outros e, portanto, com direito a muitos descendentes o que geralmente acontece nas fases iniciais do processo, a perda de diversidade que acontece quando as funções objetivos das propostas de solução que formam a população são muito próximas o que torna o processo de seleção em uma implementação quase aleatória e que tipicamente acontece nas fases finais do processo. Adicionalmente, devemos observar que os valores dos Nd_i não são inteiros e, portanto, precisamos de uma fase de integralização desses valores que na seleção proporcional é realizado pela roleta.

Na década de 90 apareceram muitas propostas alternativas para a seleção proporcional. As primeiras foram apenas modificações da seleção proporcional. Assim, pode-se realizar modificações de forma a mudar os resultados apresentados pela relação de proporcionalidade (equação 18) (perturbações nos valores da função objetivo para mudar a lógica de proporcionalidade, atribuição de limites aos valores de Nd_i, etc.) e/ou mudar ou eliminar o trabalho da roleta.

Aplicações recentes dos AG praticamente abandonaram a lógica de proporcionalidade, isto é, abandonaram o uso da relação (equação 18) e, portanto, a necessidade de padronização dos problemas de otimização quando pretendemos usar algoritmos genéticos. Uma dessas propostas é a seleção por torneio. Nessa proposta, os descendentes são escolhidos realizando n_p jogos. Em cada jogo são escolhidos aleatoriamente k propostas de solução (elementos da população) e a proposta de solução ganhadora e com direito a participar na formação da nova população é aquela que tem a função objetivo (ou equivalente) de melhor qualidade. O valor de k geralmente é pequeno e tipicamente escolhido do conjunto $k \in \{2, 3, 4, 5\}$. Após n_p jogos termina o processo de seleção.

Consideramos a seleção por torneio muito superior que a seleção proporcional porque apresenta vantagens tais como: o problema não precisa de padronização (pode ser de maximização ou de minimização), os valores da função objetivo não são muito determinantes (uma supersolução normalmente gera o mesmo número de descendentes que qualquer melhor solução), não apresenta perda de diversidade (precisa apenas que os elementos da população apresentem valores de função objetivo diferentes para realizar uma excelente seleção) e não precisa da roleta.

A recombinação escolhe (geralmente de forma aleatória) duas propostas de solução com direito a gerar novos descendentes e recombina essas soluções gerando duas candidatas ás novas soluções. Cada descendente é formado por parcelas de uma das duas soluções geradoras. Assim, usando a recombinação de um ponto, um descendente está formado por uma parcela contínua de um gerador e a seguir outra parcela contínua do outro gerador. A recombinação pode ser de um ponto, de dois pontos, vários pontos ou multiponto. Finalmente, a mutação significa uma pequena perturbação na

composição de um proposta de solução. Na codificação binária, uma posição do vetor de codificação que tem valor atual de 0 pode ser mudada para o valor 1. Em outros tipos de codificação, deve ser realizada uma pequena perturbação levando em conta o tipo de codificação usado e as características do problema. Uma análise detalhada dos operadores de recombinação e de mutação estão fora do escopo deste trabalho.

Deve-se observar que o algoritmo genético básico pode ser reformulado usando apenas conceitos existentes na pesquisa operacional e sem necessidade de procurar conceitos existentes na seleção natural e a evolução das espécies. Na verdade, o AGB pode ser interpretado como uma generalização da heurística SDH. Assim, uma generalização da heurística SDH usando a lógica do AGB seria a seguinte:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Definir a estrutura de vizinhança dinâmica definida pelos operadores de seleção, recombinação, mutação e os respectivos parâmetros.

2. Encontrar um conjunto n_p de soluções iniciais que se transforma no conjunto de soluções correntes (população corrente).

3. Avaliar a qualidade de todas as soluções existentes na população corrente, isto é, encontrar os valores de $f_i(p)$. Atualizar a incumbente, se possível.

4. Se for satisfeito um critério de parada, então pare. Em caso contrário, ir ao passo 4.

5. A partir do conjunto de soluções correntes gerar um novo conjunto de n_p^n propostas de soluções usando as estratégias de seleção, recombinação e mutação. O novo conjunto de soluções passa a ser o conjunto de soluções corrente e voltar ao passo 3.

No algoritmo apresentado anteriormente, a estratégia de seleção pode ser a seleção usando torneio. A estratégia de recombinação é simplesmente a mistura da informação codificada em duas soluções previamente selecionadas para gerar duas novas soluções candidatas, isto é, a partir de duas soluções de boa qualidade (selecionadas) e que se encontram estrategicamente localizadas no espaço de busca (região em que pode-se existir outras soluções melhores) gerar outras duas soluções que devem estar localizadas na região do espaço de busca entre as duas soluções geradoras e, portanto, existe grande probabilidade de que essas soluções novas sejam melhores de que as soluções geradoras. A estratégia

de mutação seria apenas uma pequena perturbação na solução encontrada após a recombinação, isto é, uma solução vizinha dessa solução no espaço de busca. Em outras palavras, a partir de n_p uma proposta de soluções gera n_p novas propostas de solução priorizando as melhores soluções e descartando as piores dentro de uma lógica estocástica e, portanto, encontrando soluções novas na vizinhança dessas soluções melhores no espaço de busca e, finalmente, incorporando um elemento de perturbação adicional chamado de mutação que consiste em escolher de forma estocástica uma solução vizinha.

2.3 A Metaheurística GRASP

O algoritmo GRASP, cujo nome vem do inglês *Greedy Randomized Adaptive Search Procedure*, é uma das metaheurísticas que juntamente com tabu search foram desenvolvidos usando apenas conceitos existentes no campo da pesquisa operacional. Lembremos que Glover e Kochenberger (2003), o inventor das metaheurísticas de tabu search e busca dispersa, já era um pesquisador conceituado no desenvolvimento de técnicas de otimização, especialmente no desenvolvimento de técnicas clássicas de otimização. Os inventores do GRASP Feo e Resende (1989) também já eram pesquisadores conceituados em temas de otimização no campo da pesquisa operacional. Assim, eles tinham pleno domínio da teoria e aplicação das técnicas heurísticas de otimização, assim como das técnicas exatas de otimização (as técnicas clássicas de otimização). Particularmente, Feo e Resende (1989) conheciam o algoritmo heurístico construtivo e a heurística de busca através de vizinhança (a heurística SDH, *steepest descent heuristic*). Assim, uma análise detalhada do GRASP permite verificar que esse algoritmo é apenas uma junção e uma generalização do AHC de tipo guloso e da heurística SDH. Na análise das heurísticas tradicionais (que apareceram muito antes que as metaheurísticas) verificamos que o AHC de tipo guloso e a heurística SDH eram os mais importantes pelo desempenho que apresentam e porque essas heurísticas podiam ser usadas para inventar metaheurísticas através de um processo de junção e de generalização. Portanto, o algoritmo GRASP vai ser apresentado e analisado usando os tópicos conceituais já desenvolvidos para o AHC de tipo guloso e da heurística SDH.

2.3.1 *Teoria Básica de GRASP*

O algoritmo GRASP, para um problema genérico, assume a seguinte forma:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Escolher uma estratégia heurística construtiva (um algoritmo heurístico de tipo guloso a ser usado na fase construtiva) e uma estratégia de busca local para ser usada na fase de busca local (implica escolher uma heurística tipo SDH ou mesmo outra metaheurística, juntamente com as estruturas de vizinhança e os correspondentes parâmetros).
2. Implementar a fase de pré-processamento.
3. Realizar a fase de busca construtiva.
4. Realizar a fase de pós-processamento de busca local. Atualizar a incumbente caso seja possível.
5. Se o critério de parada não for satisfeito, voltar ao passo 2. Caso contrário, pare. A resposta do algoritmo é a incumbente armazenada.

O pré-processamento tenta identificar determinadas subestruturas, isto é, atributos ou conjunto de atributos, que permitem iniciar o processo de busca construtivo de forma mais eficiente ou diminuir o espaço de busca do problema. Por exemplo, no problema do caixeiro viajante podemos descartar todos os arcos (distância entre duas cidades) que sejam maiores a uma distância adequadamente escolhida, porque existe quase certeza de que esses arcos não fazem parte da solução ótima. No outro extremo, por exemplo se existem duas cidades muito afastadas das outras cidades, então existe quase certeza que o arco entre essas duas cidades deve ser usado.

A ideia central na fase de pré-processamento é diminuir o espaço de busca incorporando componentes do problema na solução em construção, porque existe a certeza ou quase certeza de que esses componentes fazem parte da solução e descartando outros componentes, porque existe certeza ou quase certeza de que esses componentes não devem fazer parte da solução ótima ou quase ótima. Essa estratégia é muito comum e muito usada na otimização clássica, especialmente nas técnicas de solução de problemas de programação (linear) inteira mista. Nesse caso, existem várias estratégias de pré-processamento que permitem fixar variáveis inteiras ou binárias nos valores ótimos antes de iniciar o

processo de otimização (por exemplo combinando restrições em programação binária). Também existem estratégias de geração de restrições que eliminam uma parcela da região factível, porque existe a certeza de que a solução ótima não se encontra nessa parcela da região factível descartada (por exemplo, as restrições substitutas). Adicionalmente, essa estratégia de fixação de variáveis ou redução do espaço de busca, não somente faz parte de uma fase de pré-processamento. Essa estratégia, em alguns casos, faz parte da própria estratégia central de uma técnica de otimização. Assim, por exemplo, a estratégia central do algoritmo *branch and bound*, consiste em relaxar a região factível do problema original (ao relaxar a integralidade das variáveis inteiras) e depois a ideia central é gerar novas restrições para separar de forma disjuntiva a região factível do problema relaxado e diminuir sistematicamente o tamanho dessas regiões factíveis reduzidas até encontrar a solução ótima do problema original. A ideia central nos algoritmos de planos de cortes (GARFINKEL, NEMHAUSER, 1972) também é muito parecida do ponto de vista conceitual, isto é, inicialmente aumentamos a região factível do problema original relaxando a integralidade das variáveis inteiras ou binárias e depois reduzimos sistematicamente a região factível do problema relaxado, ao incorporar novos planos de cortes ou até que o ótimo do problema relaxado seja inteiro e, portanto, também ótimo do problema original. Finalmente outra estratégia de fixação de variáveis na otimização clássica é o Teste de Fixação de Variáveis Binárias de Geoffrion (1967) usado no algoritmo de Balas especializado para resolver problemas de programação binária.

A fase construtiva do GRASP é apenas um algoritmo heurístico construtivo (AHC) de tipo guloso generalizado. Portanto, na fase construtiva do GRASP a idéia central é gerar uma solução de boa qualidade, procurando contornar as limitações do AHC de tipo guloso. Adicionalmente, a proposta permite gerar um número elevado de soluções de qualidade e geralmente diferentes quando o algoritmo é processado repetidas vezes. Para entender a utilidade da fase construtiva do GRASP vamos analisar de forma resumida as limitações do AHC de tipo guloso.

Um AHC de tipo guloso gera uma solução, geralmente de boa qualidade, através de uma sequência de passos e em cada passo adicionamos uma componente da solução em construção, como mencionamos anteriormente. A componente que é incorporada na solução em construção é identificada usando um indicador de sensibilidade. Para ilustrar este fato vamos repetir a análise de um AHC simples para o problema do caixeiro viajante. Neste caso, o caixeiro inicia o tour de uma cidade escolhida aleatoriamente e em cada passo o caixeiro passa para a próxima cidade que se encontra mais próxima e ainda não foi visitada. Nesse AHC, a partir da cidade em que se encontra o caixeiro (a cidade corrente), em cada passo ordenamos as cidades ainda não visitadas, usando a informação da distância da

cidade corrente para cada uma das cidades ainda não visitadas. Nessa lista ordenada, a cidade mais próxima está em primeiro lugar na lista. Nesse contexto, o AHC escolhe a cidade mais próxima (a primeira da lista) e incorpora o arco entre essa cidade e a cidade corrente na solução em construção, e essa cidade se transforma na cidade corrente (onde se encontra o caixeiro). Um processo repetido gera um tour para o problema do caixeiro viajante. A questão central agora é identificar os motivos pelos quais o AHC mencionado anteriormente quase nunca encontra a solução ótima para o problema do caixeiro viajante de tamanho razoável. Para tentar responder essa questão podemos realizar a seguintes observações:

- Para que o AHC gere a solução ótima em cada passo devemos escolher o arco ótimo. Assim, é quase impossível que, para um problema do caixeiro viajante com $n = 100$ cidades, um AHC desse tipo encontre a solução ótima. Entretanto, se para um problema com $n = 1000$ cidades, as cidades estiverem uniformemente distribuídas em uma circunferência então o AHC anteriormente apresentado encontraria sempre a solução ótima.

- Agora supor que estamos resolvendo o problema do caixeiro viajante usando o AHC e o caixeiro se encontra na cidade k. Supor que existem ainda 80 cidades ainda não visitadas. Supor também que todos os arcos já incorporados foram ótimos. Nesse contexto, supor que apenas uma cidade (a cidade s) se encontra muito perto da cidade corrente k e todas as outras 79 se encontram muito distantes. Nesse contexto podemos ter a certeza absoluta (ou quase absoluta) de que a próxima cidade a ser visitada é a cidade s e o arco $k - s$ deve ser incorporada na solução em construção. Portanto, neste contexto o AHC funciona de forma eficiente e não precisaria de modificação.

- Supor novamente que estamos resolvendo o problema do caixeiro viajante usando o AHC e o caixeiro, se encontra na cidade k. Supor que existem ainda 80 cidades ainda não visitadas. Supor também que, todos os arcos já incorporados foram ótimos. Nesse contexto, supor que existem 5 cidades relativamente próximas da cidade k, mas a distância entre a cidade k e cada uma dessas cidades se encontra razoavelmente separadas e que as outras 75 cidades se encontram muito distantes. Nesse contexto podemos ter a certeza absoluta (ou quase absoluta) de que, uma das cinco cidades mais próximas deve ser escolhida pelo caixeiro, para ser visitada e as outras 75 estão praticamente descartadas dessa decisão. Obviamente, o AHC escolheria, dentre as cinco cidades, aquela que se encontra mais perto, mas neste caso existe a possibilidade de errar na decisão porque existe a possibilidade de que uma das outras quatro cidades façam

parte da solução ótima. Na verdade existe alguma probabilidade de que o arco de qualquer das cinco cidades seja a ótima. Nesse contexto, existe uma probabilidade nada desprezível de que a escolha da cidade localizada mais perto não seja a ótima. Portanto, se pretendemos encontrar a solução ótima, nada mais natural escolher uma das cinco cidades dando apenas maior probabilidade para a cidade que se encontra mais perto. Em resumo, uma estratégia eficiente deve levar em conta esse fato e o GRASP na fase construtiva tenta justamente incorporar esse tipo de decisão.

- Supor novamente que estamos resolvendo o problema do caixeiro viajante usando o AHC e o caixeiro se encontra na cidade k. Supor que existem ainda 80 cidades ainda não visitadas. Supor também que todos os arcos já incorporados foram ótimos. Nesse contexto, supor que existem 5 cidades próximas da cidade k e estão quase equidistantes (o mesmo fenômeno acontece se as 5 cidades se encontram à mesma distância da cidade k) e que as outras 75 cidades se encontram muito distantes. Nesse contexto também podemos ter a certeza absoluta (ou quase absoluta) de que uma das cinco cidades mais próximas deve ser escolhida pelo caixeiro para ser visitada e as outras 75 estão praticamente descartadas dessa decisão. O AHC escolheria, dentre as cinco cidades, aquela que se encontra mais perto, mas neste caso existe uma probabilidade elevada de errar na decisão. Na verdade existe alguma probabilidade próxima de 0,2 de que qualquer arco correspondente a uma das cinco cidades seja a ótima. Esse tipo de distribuição (casos em que o indicador de sensibilidade das melhores propostas são muito próximas ou quando o indicador de sensibilidade perde qualidade de discernimento) representa uma verdadeira armadilha para os AHCs. Obviamente, neste contexto a probabilidade de erro de escolha realizado por um AHC é muito grande e, portanto, pode ser mais interessante escolher de forma aleatória entre as 5 candidatas. Em resumo, o GRASP na fase construtiva deve tentar incorporar esse tipo de decisão.

Em resumo, a fase construtiva do GRASP tenta contornar as limitações de um AHC e também permite gerar um número elevado de propostas de solução diferentes. A lógica fundamental do GRASP na fase construtiva consiste em escolher a próxima componente da solução em construção dentre uma lista reduzida de candidatas, chamada de lista RCL. Um algoritmo heurístico construtivo escolheria o primeiro elemento dessa lista RCL. O número de elementos da lista RCL deve ser variável e deve levar em conta a qualidade dos componentes candidatos adicionados. Assim, para o caso do caixeiro viajante analisado anteriormente, a lista RCL pode ter apenas um elemento (apenas uma cidade se

encontra perto e todas as outras se encontram muito longe), vários elementos (algumas cidades se encontram muito perto e todas as outras se encontram muito longe) ou muitos elementos. Esse número de elementos depende da qualidade das componentes candidatas a serem incorporadas na solução em construção.

A fase construtiva do algoritmo GRASP apresenta os seguintes passos:

1. Escolher a solução inicial que pode ser vazia, isto é, sem adição de componentes, que se transforma na solução em construção.

2. Para a solução em construção (com alguns elementos já adicionados) e usando um indicador de sensibilidade, elaborar uma lista RCL com as k componentes mais atrativas. O valor de k é determinado de forma adaptativa como é mostrado adiante.

3. Escolher um elemento (componente) dos k elementos existentes na lista RCL e atualizar a solução em construção corrente com a adição da componente escolhida.

4. Se a solução em construção corrente representa uma solução factível ou foi satisfeito o critério de parada (sem encontrar uma solução factível), termina com a fase construtiva. Caso contrário voltar ao passo 3.

Os passos da fase construtiva do GRASP estão claramente apresentados, sendo que precisamos apenas analisar a forma em que se encontra os elementos da lista reduzida RCL. Assim, analisamos, brevemente, uma forma adequada para encontrar os elementos da lista RCL. Seja $f(x)$ a função objetivo de um problema com variáveis x. Tipicamente, uma função de mérito ou indicador de sensibilidade apresenta a seguinte forma:

$$h(x_i) \propto \frac{\partial z}{\partial x_i}$$

A função $h(x_i)$ na verdade representa qualquer estratégia que nos permite identificar a qualidade de uma decisão relacionada com a variação da função objetivo. Portanto, $h(x_i)$ pode ser obtida matemáticamente de maneira formal ou pode representar apenas uma informação obtida de forma intuitiva ou da experiência do pesquisador. Considerando que o problema é de minimização da função objetivo $f(x)$, então a variável mais atrativa, x_i, identificada por um algoritmo guloso, é aquela que apresenta um menor valor de $h(x_i)$. Portanto, fazem parte do conjunto RCL todas as variáveis cujos índices satisfazem a seguinte relação:

$$RCL = \left\{ i \in X/.\ h^{min} \leq h(x_i) \leq h^{min} + \alpha(h^{max} - h^{min}) \right\} \tag{19}$$

em que X é o conjunto de índices das variáveis que ainda podem ser adicionadas e α é um parâmetro fornecido pelo usuário com valores $0 \leq \alpha \leq 1$. h^{max} e h^{min} assumem a seguinte forma:

$$h^{max} = \max_{i \in X} \{h(x_i)\}$$

$$h^{min} = \min_{i \in X} \{h(x_i)\}$$

Logicamente, um algoritmo heurístico construtivo (guloso) escolheria a variável x_i com $h(x_i) = h^{min}$. O parâmeto α representa um compromisso entre escolha aleatória e gulosa e, pode-se verificar que $\alpha = 1$ representa um processo totalmente aleatório e $\alpha = 0$ um processo guloso.

Para mostrar a forma diversificada em que pode ser escolhido o indicador de sensibilidade $h(x)$ analisamos o problema do caixeiro viajante. Nesse caso, duas formas alternativas de escolher o indicador de sensibilidade $h(x)$ são as seguintes: (1) usar a informação de distância e, (2) usar a informação da solução do próprio modelo matemático após relaxar as restrições de integralidade e/ou as restrições que evitam a formação de subtours. Analisamos com detalhes apenas a primeira alternativa.

Uma forma simples de escolha de indicador de sensibilidade seria simplesmente a distância. Assim, supor que o caixeiro viajante se encontra na cidade k e seja R o conjunto de cidades ainda não visitadas. Nesse contexto, as cidades candidatas para escolher a próxima a ser visitada são identificadas usando a seguinte relação:

$$d_{kj}^{min} \leq d_{kj} \leq d_{kj}^{min} + \alpha(d_{kj}^{max} - d_{kj}^{min})$$

$j \in R$ são os índices das cidades ainda não visitadas, d_{kj}^{min} é a cidade que se encontra mais perto da cidade k e d_{kj}^{max} é a cidade que se encontra mais longe da cidade k. Assim, uma cidade pode ser visitada, se a distância se encontra dentro dos limites definidos pela relação anterior. Portanto, a lista RCL (equivalente com a relação (equação 19)) pode ser montada usando a seguinte relação:

$$\text{RCL} = \{(k, j)\ \forall j \in R/.\quad d_{kj}^{min} \leq d_{kj} \leq d_{kj}^{min} + \alpha(d_{kj}^{max} - d_{kj}^{min})\}$$

Na relação anterior podemos novamente verificar que se $\alpha = 0$, então o conjunto RCL tem apenas um elemento e se comporta como um AHC de tipo guloso e se $\alpha = 1$, então todos os elementos de R fazem parte da lista RCL (todas as cidades ainda não visitadas podem ser escolhidas) e o processo de escolha se torna totalmente aleatório. Adicionalmente devemos realizar as seguintes observações:

- O número de elementos do conjunto RCL varia em cada passo da fase construtiva. O tamanho da lista RCL depende do valor de α, mas também depende dos valores de indicador de sensibilidade de cada elemento. Para ilustrar, consideremos novamente o problema do caixeiro viajante analisado anteriormente com o caixeiro posicionado na cidade k. Nesse contexto, se apenas uma cidade se encontra perto da cidade k e todas as outras se encontram muito distantes, então a lista RCL pode ter apenas um elemento mesmo para valores de α próximos de 1. Por outro lado, se existem muitas cidades muito próximas da cidade k e um número reduzido de cidades se encontra muito distante da cidade k, então a lista RCL pode ter muitos elementos mesmo para valores pequenos de α. Em resumo, para um mesmo valor de α o número de elementos da lista RCL pode variar muito em cada iteração da fase construtiva.

- Existe a possibilidade de usar um α variável na fase construtiva como uma forma adicional de dispor de uma lista RCL variável em cada iteração da fase construtiva e também para usar o melhor valor de α em cada passo do algoritmo. Esse tipo de algoritmo GRASP é chamado de algoritmo GRASP reativo.

Uma vez montado os elementos da lista RCL, então o próximo passo é a escolha de um elemento dessa lista e que deve ser adicionado na solução em construção. Essa escolha pode ser realizada de duas formas: (1) aleatoriamente e, (2) usando uma função de distribuição de probabilidade como, por exemplo, a função de distribuição de probabilidade linear $b_i = 1/r_i$ em que r_i representa a posição que ocupa a componente i na lista ordenada de RCL, de acordo com a qualidade do indicador de sensibilidade (a melhor é a primeira da lista). Portanto, a probabilidade de escolha do elemento i é encontrada usando a relação:

$$p_i = \frac{b_i}{\sum_{j \in RCL} b_j} \tag{20}$$

Para fins de ilustração supor que está sendo resolvido um problema usando o GRASP e foram selecionadas as seguintes componentes ordenadas em ordem de qualidade: x_2, x_4, x_{32}, x_9 e x_7.

Usando a função de distribuição de probabilidade linear ($b_i = 1/r_i$) e a relação (equação 20) se encontra facilmente a probabilidade de escolha de cada componente mostrada na Tabela 1. Para o caso aleatório a probabilidade é igual para cada uma das variáveis e igual a 0,20.

Tabela 1: Probabilidade de escolha das componentes.

Variável x_i	r_i	b_i	p_i
x_2	1	1	60/137
x_4	2	1/2	30/137
x_{32}	3	1/3	20/137
x_9	4	1/4	15/137
x_7	5	1/5	12/137

Fonte: Informações da pesquisa da autora.

Logicamente, a soma de probabilidades deve ser igual à unidade. A escolha da componente pode ser implementada usando um gerador de números aleatórios como na implementação da seleção proporcional no algoritmo genético.

2.3.2 O Algoritmo GRASP Reativo

Um tipo especial do algoritmo GRASP é o chamado algoritmo GRASP reativo que usa um valor de α adaptativo, isto é, são usados valores diferentes de α durante a fase de construção e, portanto, o tamanho da lista RCL também deve variar devido ao uso de um α diferente em cada passo do algoritmo GRASP. A ideia básica consiste em escolher um conjunto de valores para α e usar, preferencialmente, aquele valor de α que está apresentando melhor desempenho durante o processo. Foi mostrado experimentalmente que o GRASP reativo apresenta melhor desempenho comparado com o GRASP convencional. Seja A um conjunto de m valores para α previamente especificados. Pretende-se encontrar uma estratégia para identificar aquele valor de α mais atrativo. Assim, deve-se determinar a probabilidade p_i de que o valor especificado α_i seja escolhido em uma iteração do algoritmo GRASP.

No GRASP reativo, o processo é iniciado dando igual probabilidade para cada um dos m valores de α especificados, isto é, $p_i = 1/m;\ i = 1, \ldots, m$. Durante o processo, a distribuição de probabilidade é atualizada com o histórico do processo. Para um problema de minimização com

função objetivo $v(x)$ são calculados os valores médios da função objetivo, para cada um dos valores α_i usados durante as últimas n_p transições, para encontrar a distribuição q_i da seguinte forma:

$$q_i = \left(\frac{v^*}{\overline{v}_i}\right)^{\delta} \qquad i = 1, \ldots, m \tag{21}$$

em que, v^* é a incumbente do processo GRASP reativo e $\overline{v}_i$ é o valor médio dos valores das funções objetivos obtidos usando α_i. Para encontrar a distribuição de probabilidade padronizada p_i, deve ser usada a relação:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j} \tag{22}$$

A relação anterior permite selecionar o α_i que deve ser usado em cada passo do processo GRASP reativo. Após n_p transições, os valores dos p_i são atualizados, recalculando os valores médios $\overline{v}_i$ e usando (equação 21) e (equação 22). Esses novos valores dos p_i são usados para selecionar o α_i nas próximas n_p transições. Este processo de escolha, também é muito parecido com a seleção por escalonamento linear no algoritmo genético. O processo GRASP deve priorizar o uso daquele α_i que está encontrando soluções com funções objetivo de melhor qualidade, porque esse α_i deve apresentar um maior valor de p_i e, portanto, com maior chance de ser selecionado. Para selecionar o α_i, que deve ser usado, pode ser implementada uma estratégia similar, à seleção proporcional no algoritmo genético, usando um gerador de número aleatório $p \in [0, 1]$.

Para ilustrar o uso do GRASP reativo, supor um problema de minimização com função objetivo $f(x)$, e usa um valor de α_i em cada transição. Na resolução desse problema, nas últimas $n_p = 10$ transições foram obtidos os resultados mostrados na Tabela 2 e usando 4 valores de α, isto é, $A = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{1/5, 1/3, 1/2, 3/4\}$.

Tabela 2: Funções objetivos de 10 transições.

Teste	1	2	3	4	5	6	7	8	9	10
α_i	α_3	α_2	α_4	α_1	α_1	α_2	α_1	α_2	α_4	α_2
$f(x)$	80	52	56	64	60	48	56	51	52	49

Fonte: Informações da pesquisa da autora.

Encontramos os valores de probabilidade p_i que permitam encontrar o α_i para continuar o processo GRASP reativo. Consideremos dois casos: (1) $\delta = 1$ e, (2) $\delta = 5$.

Da Tabela 2 se encontra facilmente os valores médios: $\overline{v}_1 = 60$, $\overline{v}_2 = 50$, $\overline{v}_3 = 80$ e $\overline{v}_4 = 54$.

Com os valores anteriores e considerando que a incumbente é $f^* = 48$ podem ser encontrados os valores dos q_i e os valores de probabilidade p_i usando as relações (equação 21) e (equação 22). Os resultados, para $\delta = 1$ e $\delta = 5$ são apresentados na Tabela 3.

Tabela 3: Valores de probabilidade p_i.

$\delta = 1$							
q_1	q_2	q_3	q_4	p_1	p_2	p_3	p_4
0,800	0,960	0,600	0,889	0,246	0,295	0,185	0,274
$\delta = 5$							
0,328	0,815	0,078	0,555	0,185	0,459	0,044	0,312

Fonte: Informações da pesquisa da autora.

Deve-se observar que o algoritmo GRASP reativo apresenta em cada passo duas decisões de caráter aleatório: (1) escolha de uma variável da lista RCL e, (2) escolha do parâmetro α_i que determina o tamanho da lista RCL.

A fase de pós-processamento do GRASP é implementada para cada solução encontrada na fase construtiva. Este tópico é analisado na próxima Subseção.

2.3.3 *A Fase de Busca Local do GRASP*

Na formulação inicial do GRASP a fase de pós-processamento ou de busca local era realizada por uma heurística de busca através de vizinhança, isto é, por uma heurística parecida com a heurística SDH. A ideia central é tentar encontrar uma solução ótima local na vizinhança da solução encontrada na fase construtiva.

Através do tempo o GRASP foi sofisticando a proposta de otimização na fase de busca local, após a fase construtiva. Assim, nas propostas iniciais a proposta era usar uma heurística de busca local, como a heurística SDH ou como o processo de intensificação no algoritmo *tabu search*.

Em geral, a fase de busca local pode ser um processo muito simples ou muito complexo. Assim, no caso mais simples pode ser implementada uma heurística tipo SDH. No caso mais sofisticado pode ser outra metaheurística tais como *simulated annealing*, o algoritmo genético, *tabu search*, VNS, etc. Propostas intermediárias podem incorporar estratégias de otimização que fazem parte de

metaheurísticas sofisticadas como, por exemplo, a estratégia de *path relinking* que foi inventada como sendo parte do *tabu search*.

Em resumo, o algoritmo GRASP é uma integração e generalização sofisticada de dois algoritmos que já existiam no campo da pesquisa operacional, isto é, do algoritmo heurístico de tipo guloso e da heurística de busca através de vizinhança clássico que neste trabalho chamamos de heurística SDH. Também mostramos neste trabalho, que metaheurísticas tais como o algoritmo simulated annealing, tabu search e o algoritmo genético podem ser idealizados como generalizações da heurística SDH. Em resumo, o algoritmo GRASP apresenta as seguintes características fundamentais:

- Uma fase de pré-processamento, inspirada em conceitos existentes na pesquisa operacional clássica e cujo objetivo é incorporar ou descartar elementos que formam parte da solução ótima e, portanto, reduzindo o espaço de busca.

- Uma fase construtiva, que na verdade é um algoritmo heurístico construtivo generalizado, onde se tenta contornar a característica gulosa do algoritmo heurístico construtivo e ter a possibilidade de gerar, um conjunto elevado de soluções de qualidade usando uma mesma heurística construtiva.

- Uma fase de pós-processamento, que na verdade é uma fase de melhoria da solução encontrada na fase construtiva, que originalmente era implementada usando uma heurística simples de busca através de vizinhança, como a heurística SDH. Entretanto, essa fase de busca local pode ser implementada usando heurísticas ou metaheurísticas sofisticadas, como qualquer outra metaheurística existente na literatura especializada.

Adicionalmente, como a fase construtiva e de busca local podem ser repetidos, então existe grande possibilidade de encontrar uma solução ótima global ou soluções quase-ótimas. Finalmente, podemos mencionar que, segundo um dos inventores do GRASP, o Feo e Resende (1989), uma das estratégias mais eficientes que podem ser usadas na fase de busca local, é a estratégia de *path relinking*.

Capítulo 3

REVISÃO BIBLIOGRÁFICA

Existe muita bibliografia relacionada com o problema de otimização do carregamento de contêiner. Uma análise panorâmica sobre as propostas de otimização para o PCC permite verificar que ainda existe pouco interesse em resolver esse problema desenvolvendo técnicas de otimização exatas tais como algoritmos tipo *Branch and Bound* especializados. Assim, a maioria dos pesquisadores ainda trabalha desenvolvendo técnicas heurísticas e metaheurísticas.

Geralmente, em trabalhos de tese é típico apresentar a revisão bibliográfica citando em pequenos parágrafos muitos trabalhos relacionados com o tópico de pesquisa. Neste caso, optamos por uma estratégia diferente e apresentamos na parte de revisão bibliográfica poucos trabalhos, mas esses trabalhos são analisados com muito detalhe. Também, deve-se mencionar que foram escolhidos para análise e discussão os trabalhos que consideramos mais interessantes e aqueles mais próximos da técnica de otimização escolhida nesta tese. Essa revisão bibliográfica de poucos trabalhos é realizada nas seções seguintes.

Na literatura especializada existem vários modelos matemáticos para resolver PCC. Os modelos matemáticos podem ser considerados como sendo de programação linear inteira de grande porte. Segundo a maioria dos pesquisadores, o crescimento explosivo do número de variáveis torna pouco viável resolver o PCC usando técnicas de otimização exatas e software gerais desenvolvidos para resolver problemas de programação inteira como, por exemplo, o CPLEX. Neste trabalho não foi explorada essa linha de pesquisa e foi priorizada uma análise detalhada das heurísticas e metaheurísticas como técnicas de otimização para o PCC.

3.1 Principais Algoritmos Heurísticos Usados no Problema de Carregamento de Contêiner

Existem muitas propostas de otimização da família das heurísticas usadas na solução do PCC. Entretanto, neste trabalho analisamos com detalhes a heurística de George e Robinson (1980), a heurística de Cecilio (2003) e a heurística de Pisinger (2002).

3.1.1 *A Heurística de George e Robinson (1980)*

George e Robinson (1980) desenvolveram um algoritmo heurístico que preenche o contêiner em camadas ao longo do comprimento do contêiner.

Os itens a serem carregados foram chamados pelos autores, de "caixas" (boxes, termo utilizado até hoje), sendo estes divididas em duas classes: tipo "aberta" (open), são as caixas já utilizadas no carregamento das camadas anteriores, e as do tipo "fechada", são as caixas que ainda não foram utilizadas no carregamento do contêiner. As caixas do tipo "abertas" têm preferência quando se inicia uma camada. Este critério é determinante na escolha da caixa e na definição da profundidade da camada. Os outros tipos de caixas são utilizadas apenas para preencher os espaços residuais deixados dentro da camada atual.

O método de George e Robinson (1980) carrega as caixas em colunas compostas de caixas do mesmo tipo. Cada vez que um tipo de caixa é utilizada, a ela é dada prioridade sobre as demais. O contêiner é carregado do seu fundo para sua entrada, tentando sempre manter uma superfície de trabalho plana. Note que pode não ser possível colocar todas as caixas no contêiner, mesmo que o volume total das caixas seja menor que o volume do contêiner, pois, desperdícios de espaço são inevitáveis devido ao arranjo geométrico das caixas. O volume não ocupado por caixas é considerado volume perdido.

A heurística considera apenas as restrições geométricas do problema de carregamento, não são feitas restrições quanto ao número de caixas que podem ser empilhadas umas sobre as outras, nem sobre qual face das caixas deve ficar voltada para cima.

A heurística baseia-se no preenchimento do contêiner camada por camada. Uma camada é uma seção do comprimento do contêiner na sua completa altura e largura. O comprimento da camada é

determinado pela maior dimensão da primeira caixa carregada. Nas Figuras de 2 a 11 mostra-se o carregamento, passo a passo da heurística de George e Robinson (1980), dividida em 10 camadas e o contêiner é carregado do seu fundo para sua entrada, tentando sempre manter uma superfície de trabalho plana. Uma nova camada não deve ser iniciada até que a camada anterior esteja totalmente preenchida. Para que a profundidade da camada tenha um tamanho considerável, algumas restrições são impostas na escolha da primeira caixa. Outros tipos de caixas são considerados apenas para preencher espaços deixados dentro da camada (quando não é possível preenchê-la totalmente com caixas de um mesmo tipo) ou espaços combinados com espaços que sobraram em camadas anteriores.

Para preencher uma camada, escolhe-se um tipo de caixa e preenche-se a camada com quantas colunas completas forem possíveis. Se a camada não for completamente preenchida pelo tipo de caixa que iniciou, os espaços vazios na camada são considerados como novos espaços a serem preenchidos. Esses espaços não constituem uma nova camada e são preenchidos de acordo com a profundidade da camada, levando-se em conta o número de colunas que cabem no espaço e buscando o melhor preenchimento.

No PCC cujo objetivo é carregar i tipos de caixas diferentes com dimensões (i é o número de tipos de caixas), l_i (comprimento) x w_i (largura) x h_i (altura) e quantidade conhecida (b_i) dentro de um contêiner de dimensão L (comprimento) x W (largura) x H (altura). O problema consiste em determinar uma ordem de carregamento e a posição das caixas dentro do contêiner de forma a otimizar o aproveitamento do espaço do contêiner. Mesmo que a soma dos volumes das caixas seja menor que o volume do contêiner, é possível que alguma(s) caixa(s) não seja(m) carregada(s) devido ao arranjo geométrico das mesmas.

O contêiner é preenchido camada por camada e uma nova camada só é criada quando a camada atual estiver carregada (CECILIO, 2003).

A Heurística proposta por George e Robinson (1980) pode ser dividida em cinco etapas:

1ª etapa: Achar o primeiro tipo de caixa a ser carregado em uma nova camada e o tamanho da profundidade da camada;

2ª etapa: Definir a altura e a largura da caixa que será utilizada e a quantidade de caixas carregadas;

3ª etapa: Criar novos espaços de carregamento;

4ª etapa: Escolher o novo espaço de carregamento;

5ª etapa: Achar o tipo de caixa para preencher os espaços vazios já existentes nas camadas e o tamanho da profundidade.

Antes de explicarmos passo a passo a heurística proposta, iremos introduzir o conceito de ranque (ranking) das caixas que será utilizado em alguns passos da heurística. O ranqueamento das caixas é feito da seguinte forma:

1. primeiramente escolhe-se, entre as menores dimensões, o tipo de caixa com a maior dimensão, ou seja, seleciona-se a menor dimensão entre o comprimento, largura e altura de cada tipo de caixa e dentro dessa lista (das caixas de menor dimensão), escolhe o tipo de caixa que tiver maior dimensão.

2. Havendo empate neste critério, escolhe-se o tipo de caixa de maior quantidade.

3. E como último critério de desempate, escolhe o tipo de caixa com maior dimensão.

O exemplo ilustrado na Tabela 4 explica melhor os critérios de desempate:

Tabela 4: Critérios de desempate.

Tipo de caixa(i)	Quantidade(i)	Comprimento l_i	Largura w_i	Altura h_i	Menor Dimensão
1	12	7	3	4	3
2	12	6	3	3	3
3	14	5	4	5	4
4	16	5	5	6	5

Fonte: Cecilio (2003).

Neste caso, a primeira caixa a ser carregada é a caixa do tipo 4, seguida pela caixa do tipo 3. As caixas dos tipos 1 e 2 ficaram empatadas tanto em relação ao critério, caixa com maior dimensão entre as menores dimensões, quanto em relação ao critério: quantidade de caixas. O desempate entre as caixas ocorre em relação ao critério: caixa com maior dimensão (das 3 dimensões de cada tipo de caixa empatada, verifica qual tem maior dimensão), devendo a caixa 1 ser colocada antes da caixa 2. Portanto, a ordem de carregamento é:

4	**3**	**2**	**1**

A heurística passo a passo:

1ª etapa: Definir o tipo de caixa a ser carregado, iniciando uma nova camada e o tamanho da profundidade da camada.

Entende-se por camada carregada uma parte do comprimento do contêiner (L) na sua altura completa (H) e largura (W). A seção do comprimento do contêiner utilizada é denominada profundidade da camada.

Para iniciar o carregamento de uma nova camada, verifica-se se existe alguma caixa classificada como "aberta".

- Se existir mais de um tipo de caixa classificada como "aberta", a heurística sugere dois critérios de desempate: escolher o tipo de caixa com maior quantidade de caixas ou escolher o tipo de caixa com melhor ranque (posição de escolha da caixa);
- Caso contrário, se existir apenas caixas classificadas como "fechadas", escolhe-se o tipo de caixa com melhor posição no ranque das caixas.

Em cada camada, a maior dimensão da primeira caixa a ser carregada determina a profundidade da camada (o espaço do comprimento de cada camada). As outras dimensões da caixa (altura e largura) devem ser rotacionadas de acordo com o algoritmo, para que seja alocada da melhor forma possível.

As caixas que ainda não foram carregadas são armazenadas na "Lista de Caixas não Carregadas".

2ª etapa: Definir a altura e a largura da caixa que será utilizada e a quantidade de caixas carregadas.

Definido o tipo de caixa que será utilizado e a dimensão da caixa que será fixa no comprimento do contêiner, o algoritmo verifica a dimensão da altura e da largura e a quantidade de caixas que serão carregadas.

- Se a quantidade de caixas for suficiente para preencher mais do que uma coluna completa utilizando quaisquer dos dois posicionamentos possíveis (trocando a altura com a largura), então o algoritmo escolhe a dimensão que minimiza a perda da altura do contêiner.
- Caso contrário, se não for possível carregar uma coluna completa escolhe-se a maior, entre as duas dimensões que restaram. E fixa-se essa dimensão da caixa na largura do contêiner, a fim de facilitar o carregamento posterior.

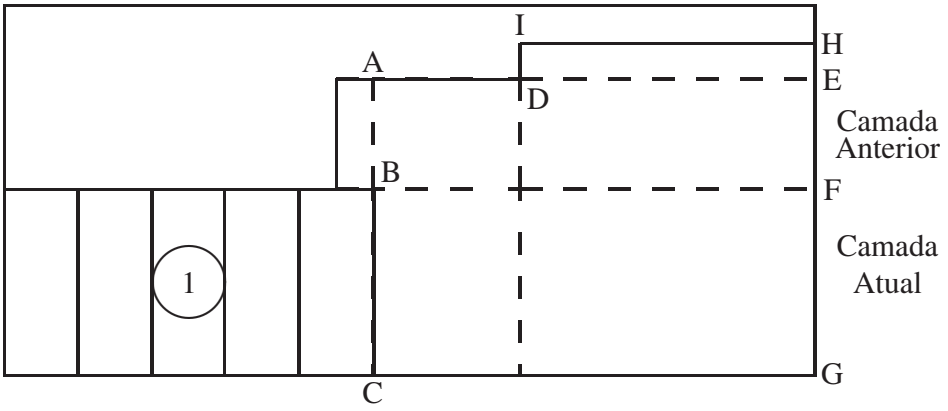
Definido o posicionamento da caixa, verifica-se a possibilidade de colocar uma coluna inteira com esse tipo de caixa.

- Se não for possível o carregamento de uma coluna completa, colocam-se quantas colunas completas forem possíveis.

- No caso da viabilidade do carregamento em mais de uma coluna, torna-se necessário o cálculo da Largura Flexível.

George e Robinson (1980) introduziram o conceito de "Largura Flexível" (flexible width) a fim de combinar espaços não ocupados entre camadas para aumentar a utilização do contêiner. Na Figura 2, após o carregamento de **1** cria-se o espaço BCGF como espaço de carregamento na largura do contêiner (w). Porém, verificamos que é possível unir o espaço vazio da camada anterior **ABFE** com o espaço da camada atual formando então uma região de carregamento **ACGE**. Ao criar essa nova região de carregamento, percebemos que o espaço **IDEH** ficaria vazio. Para não ocorrer esse tipo de problema, a heurística calcula a Largura Flexível **AD**. Pela Figura 2, a próxima região a ser carregada terá sua profundidade aumentada de **BC** para **AC**.

Figura 2: Largura Flexível.



Fonte: Informações da pesquisa da autora.

Calculada a Largura Flexível, o algoritmo verifica, se o número de colunas que serão carregadas é maior que a Largura Flexível.

- Se sim, então carrega-se uma coluna a mais do que a permitida pela Largura Flexível, conforme Figura 2.

- Senão, carrega-se o maior número possível de colunas completas.

Após essa etapa, o algoritmo atualiza a quantidade de caixas e a classificação ("aberta" ou "fechada").

O carregamento da região **ACGE** é feito até exceder em no máximo uma coluna a largura flexível **AD**. Agora, a heurística tentará aumentar a profundidade do carregamento aproveitando o espaço livre **IDEH**.

3ª etapa: Criar os novos espaços de carregamento.

Os novos espaços para o carregamento são criados conforme descrito a seguir:

Primeiramente, geram-se os novos espaços de carregamento "à frente" e colocam-se esses espaços na "Lista de Espaços a serem Carregados". A seguir, cria-se o espaço "ao lado" e verifica, se a largura do espaço é maior ou igual à menor dimensão das caixas que ainda não foram carregadas. Se a largura for maior ou igual, então adiciona-se esse espaço à "Lista de Espaços a serem Carregados". Caso contrário descarta-se esse espaço. O último espaço gerado é o "acima". Gerado esse espaço verifica, se a altura desse espaço é maior ou igual que a menor dimensão das caixas que ainda não foram carregadas. Se a altura for maior ou igual, adiciona-se esse espaço à "Lista de Espaços a serem Carregados". Caso contrário, descarta-se o espaço.

Independente da dimensão do espaço gerado "à frente", esse nunca é descartado uma vez que o algoritmo tenta unir esses espaços através do cálculo da Largura Flexível.

Os espaços gerados são armazenados na "Lista de Espaços a serem Carregados" de forma que, carrega-se primeiro o espaço gerado "acima", seguido "ao lado" e finalmente "à frente".

4ª etapa: Escolher o novo espaço a ser carregado.

Nessa etapa, o algoritmo verifica se existe alguma caixa que ainda não foi carregada e se existem espaços ociosos a serem carregados.

- Verifica-se a existência de alguma caixa na "Lista de Caixas não Carregadas";
  - Se existir, então verifica-se a existência de espaços a serem carregados na "Lista de Espaços a serem Carregados" e escolhe-se o espaço que estiver no topo da lista.
    - Verifica-se a viabilidade de unir o espaço atual de carregamento com algum espaço da "Lista de Espaços Rejeitados" definido na segunda etapa da heurística. Para unir os espaços é necessário que a altura e a largura do espaço da "Lista de Espaços a serem Carregados" sejam menores ou iguais à altura e a largura do espaço rejeitado, respectivamente.
  - Se for possível, unem-se os espaços e verifica-se novamente a possibilidade de unir a esse espaço criando outro espaço da "Lista de Espaços Rejeitados";

  - Se não for viável unir os espaços, calcula-se a Largura Flexível.

- Caso contrário, significa que todas as caixas foram carregadas.

Se o espaço a ser carregado for considerado uma nova camada, o algoritmo volta para a 1ª etapa. Caso contrário, vai para a 5ª etapa.

5ª etapa: Achar o tipo de caixa para preencher os espaços vazios já existentes nas camadas.

Os novos espaços de carregamento possuem suas dimensões (profundidade, largura e altura) definidas. Para o carregamento desses espaços, o algoritmo verifica se algum tipo de caixa cabe no espaço.

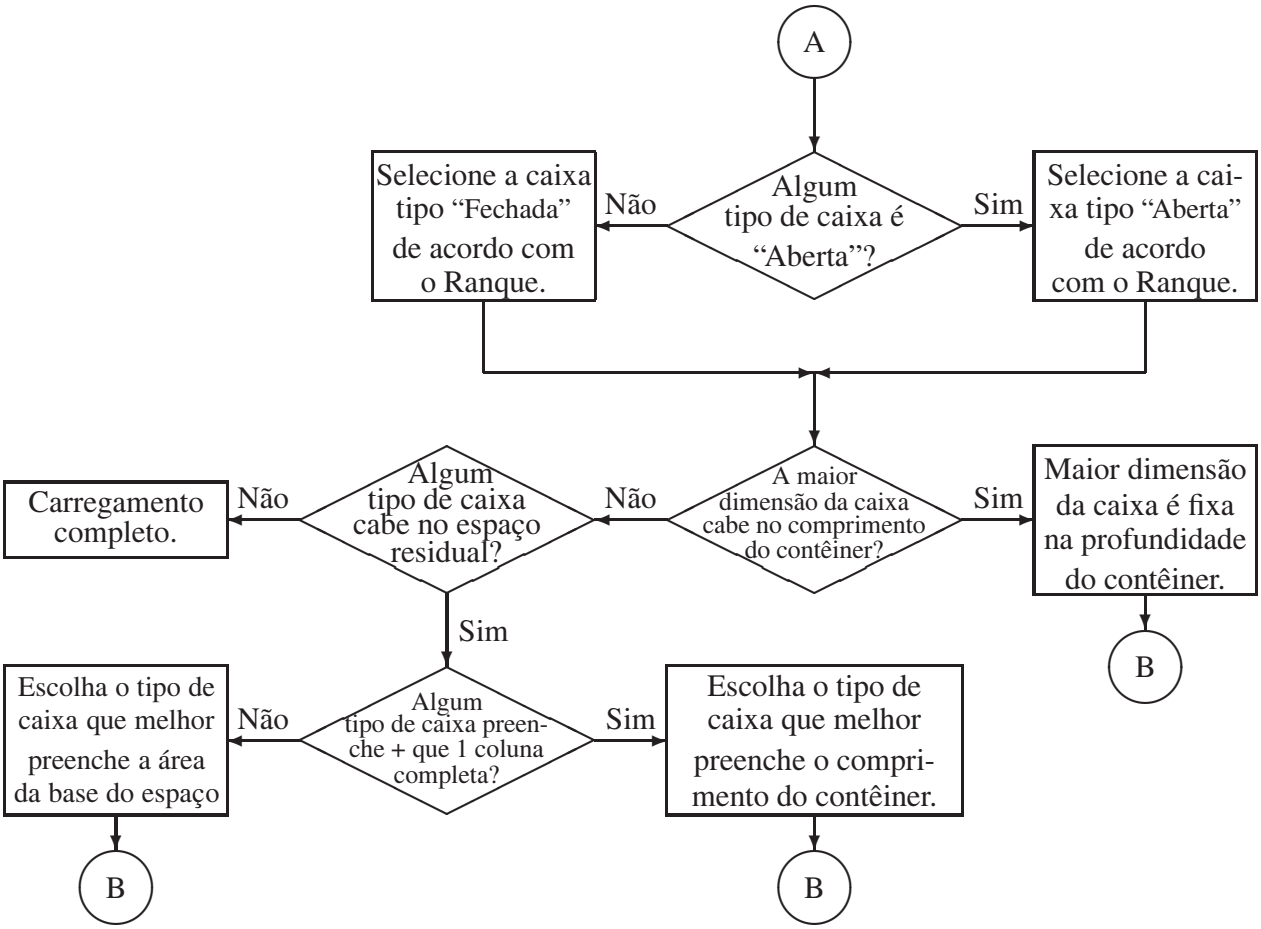
- Se couber verifica se algum tipo de caixa preenche mais de uma coluna.

  - Se sim, então escolhe-se a caixa que melhor preencha a profundidade e a heurística volta para a 2ª etapa.
  - Caso contrário, escolhe-se o tipo de caixa que melhor preencha a área da base e a heurística volta para a 2ª etapa.

- Se o algoritmo verificar que nenhuma caixa cabe no espaço vazio, armazena-se o espaço na "Lista de Espaços Rejeitados". A heurística volta para a 4ª etapa.

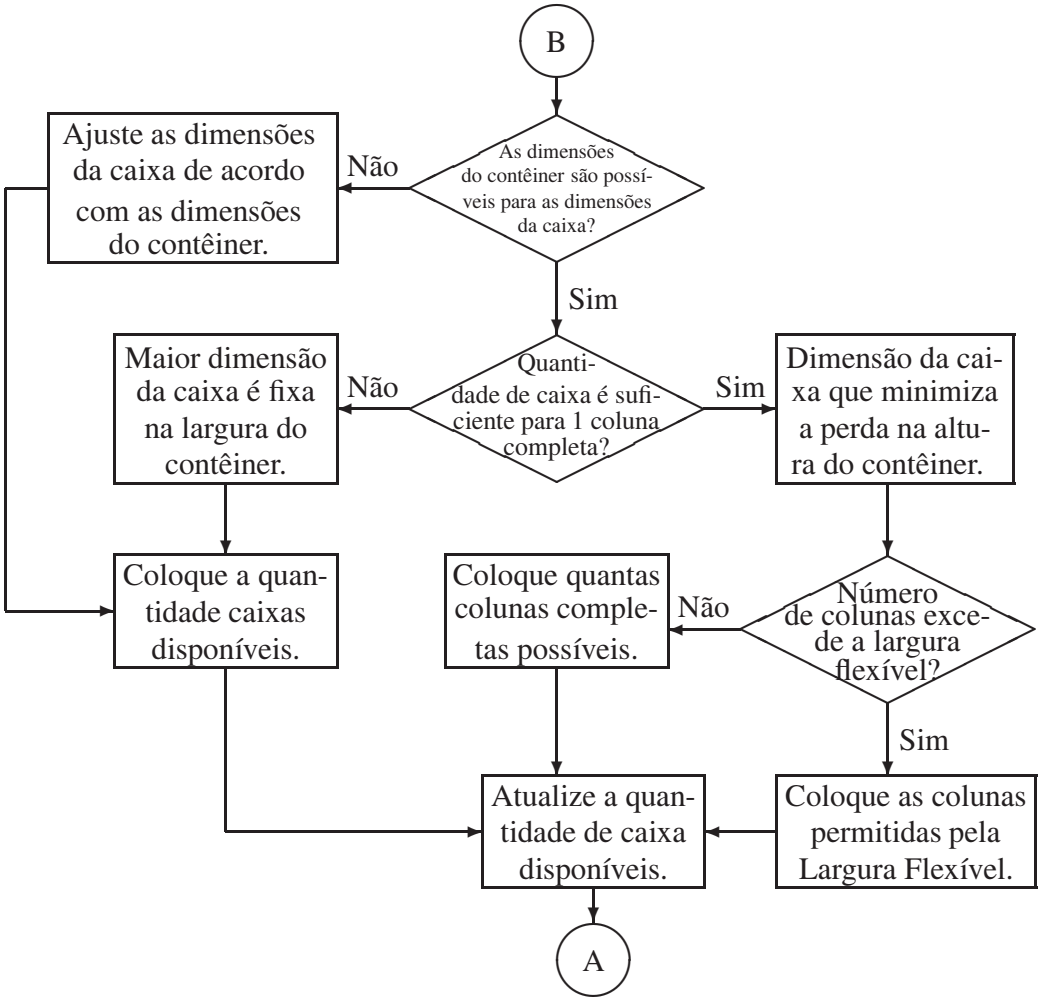
As Figuras 3 e 4 apresentam o fluxograma em diagrama de blocos da heurística de George e Robinson (1980).

Figura 3: Escolha do tipo de caixa a ser carregada.



Fonte: Informações da pesquisa da autora.

Figura 4: Carregamento das caixas.



Fonte: Informações da pesquisa da autora.

Resultado:

A heurística foi reproduzida, com os dados apresentados por George e Robinson (1980), em que utilizam os diferentes tipos de caixas disponíveis para o carregamento de um único contêiner. Os dados apresentados pelos autores tem-se **8** tipos de caixas diferentes e um total de **784** caixas. O contêiner tem dimensões de **5,793m** de comprimento, **2,236m** de largura e **2,261m** de altura.

A heurística foi reproduzida, em Matlab 7.1 e o teste foi feito em um PC Dell Processador Core 2 duo/2GB e memória RAM 2GB, com sistema operacional Windows XP.

O percentual de aproveitamento do volume do contêiner foi de **89,7578%** de carga alocada em seu interior, deixando de fora apenas **uma caixa do tipo 07**.

A heurística reproduzida mostrou uma ocupação de **26,28746**m^3 do volume ocupado para o teste. Como foi possível observar os resultados obtidos neste trabalho são coerentes aos resultados de George e Robinson (1980).

Foram colocadas no total **783 caixas**, as caixas são organizadas em **10 camadas**. As caixas que compõem as camadas estão apresentadas passo a passo nas Tabelas de 5 a 15 e nas Figuras 5 a 14 onde:

- i = tipos de caixas.
- c_i = comprimento da caixa i.
- l_i = largura da caixa i.
- h_i = altura da caixa i.
- q_i = quantidade de cada tipo de caixa.
- v_i = volume unitário de cada tipo de caixa.
- vt_i = volume da quantidade total de cada tipo de caixa.

Nas Tabelas de 5 a 15 apresentamos os dados das caixas disponíveis para serem carregadas em cada camada e nas Figuras de 5 a 14 o carregamento de cada camada completa.

Tabela 5: Dados de George e Robinson (1980) com 784 caixas

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	400	0,029788	11,9152
2	901	185	195	160	0,0325	5,2000
3	901	195	265	40	0,046559	1,8624
4	1477	135	195	40	0,038882	1,5553
5	614	480	185	8	0,054523	0,4362
6	400	400	135	16	0,0216	0,3456
7	264	400	400	80	0,04224	3,3792
8	385	365	290	40	0,040752	1,6301

Fonte: George e Robinson (1980).

A ordem de ranqueamento dos oito tipos de caixas, da Tabela 5 são:

Ranqueamento

08	**07**	**03**	**02**	**05**	**01**	**04**	**06**

Fonte: Informações da pesquisa da autora.

Camada 01:

Escolhemos as caixas do tipo 8, de acordo com o ranque.

O carregamento é feito na forma $385 \times 290 \times 365$. Assim, a camada 01 assume a seguinte distribuição: $1 \times 6 \times 6$, isto é, a camada 01 é formada por 36 caixas do tipo 8.

- Espaço residual de largura: $385 \times 496 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar as caixas do tipo 5 na seguinte estrutura: $185 \times 480 \times 614$, isto é, $2 \times 1 \times 3 = 6$ caixas.

  Existe ainda um espaço residual de altura no espaço residual de largura.

  O espaço residual de altura apresenta as dimensões: $370 \times 480 \times 419$. Nesse espaço podem ser alocadas caixas tipo 8 com a seguinte estrutura: $290 \times 385 \times 365$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 8.

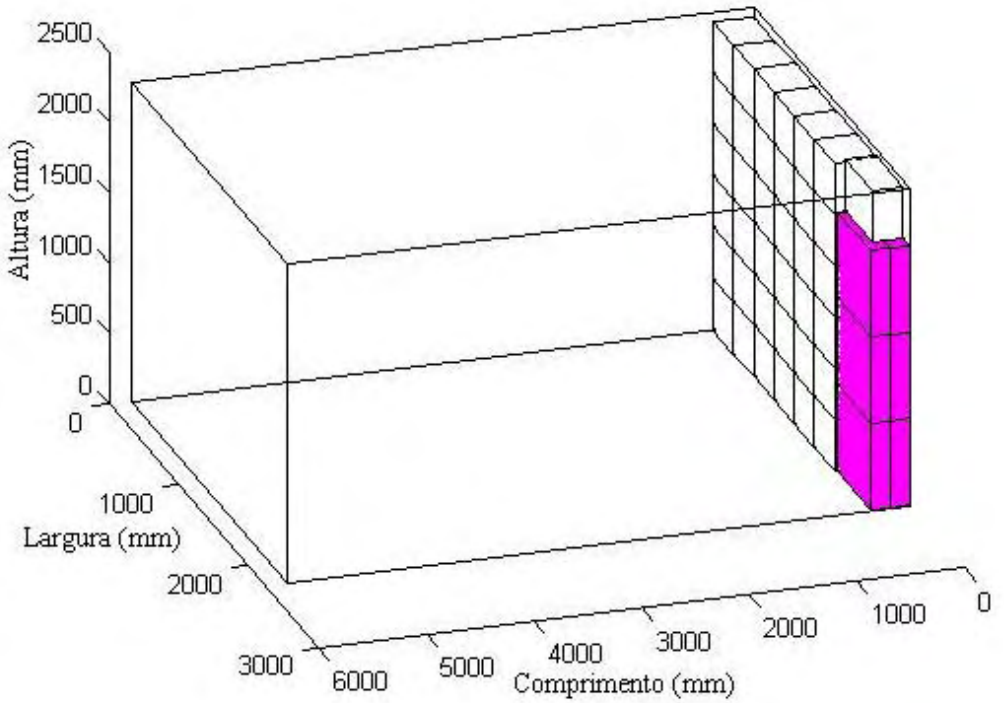
Resumo da primeira camada:

- 37 caixas do tipo 8 e 6 caixas do tipo 5 alocadas.
- Volume ocupado: 37(0,040752) + 6(0,054523) = 1,8350 m^3.
- Volume usado: $385 \times 2236 \times 2261 = 1,9464\ m^3$.
- Taxa de ocupação: 94,2766 %.
- Profundidade usada: 385 mm.
- Largura Flexível $15 \times 496 \times 2261$.
- Na Figura 5 apresentamos o carregamento da primeira camada, com 43 caixas.
- Na Tabela 6 apresenta-se os tipos de caixas não carregadas e a quantidade de caixas atualizadas.

Camada 02:

Escolhemos as caixas do tipo 8, de acordo com o ranque.

Figura 5: Carregamento da primeira camada com 43 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 6: Caixas restantes após a primeira camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	400	0,029788	11,9152
2	901	185	195	160	0,0325	5,2000
3	901	195	265	40	0,046559	1,8624
4	1477	135	195	40	0,038882	1,5553
5	614	480	185	2	0,054523	0,1091
6	400	400	135	16	0,0216	0,3456
7	264	400	400	80	0,04224	3,3792
8	385	365	290	3	0,040752	0,1223

Fonte: Informações da pesquisa da autora.

O carregamento é feito na forma $385 \times 365 \times 290$. Assim, a camada 02 assume a seguinte distribuição: $1 \times 1 \times 3$, isto é, a camada 02 é formada por 3 caixas do tipo 8.

- Espaço residual de altura: $385 \times 365 \times 1391$.

  Neste caso pode ser usado esse espaço residual para alocar 2 caixas tipo 2 com a seguinte estrutura: $185 \times 195 \times 901$, isto é, $2 \times 1 \times 1 = 2$ caixas do tipo 2.

- Espaço residual de largura: $385 \times 1871 \times 2261$.

  No espaço residual podemos carregar 44 caixas do tipo 2, com a seguinte estrutura: $185 \times 901 \times 195$, isto é, $2 \times 2 \times 11 = 44$ caixas do tipo 2.

Resumo da segunda camada:

- 3 caixas tipo 8 e 46 caixas tipo 2 alocadas.
- Volume ocupado: 3(0,040752) + 46(0,0325) = 1,6173 m^3.
- Volume usado: $385 \times 2,236 \times 2,261 = 1,9464\ m^3$.
- Taxa de ocupação: 83,0894 %.
- Profundidade usada: 770 mm.
- Largura Flexível: $15 \times 1871 \times 2261$.
- Na Figura 6 apresentamos o carregamento da segunda camada, com 49 caixas.
- Tabela 7 apresenta-se a quantidade e os tipos de caixas não carregadas no contêiner, onde temos que a caixa tipo 8 é totalmente carregada, portanto retirada.

Camada 03:

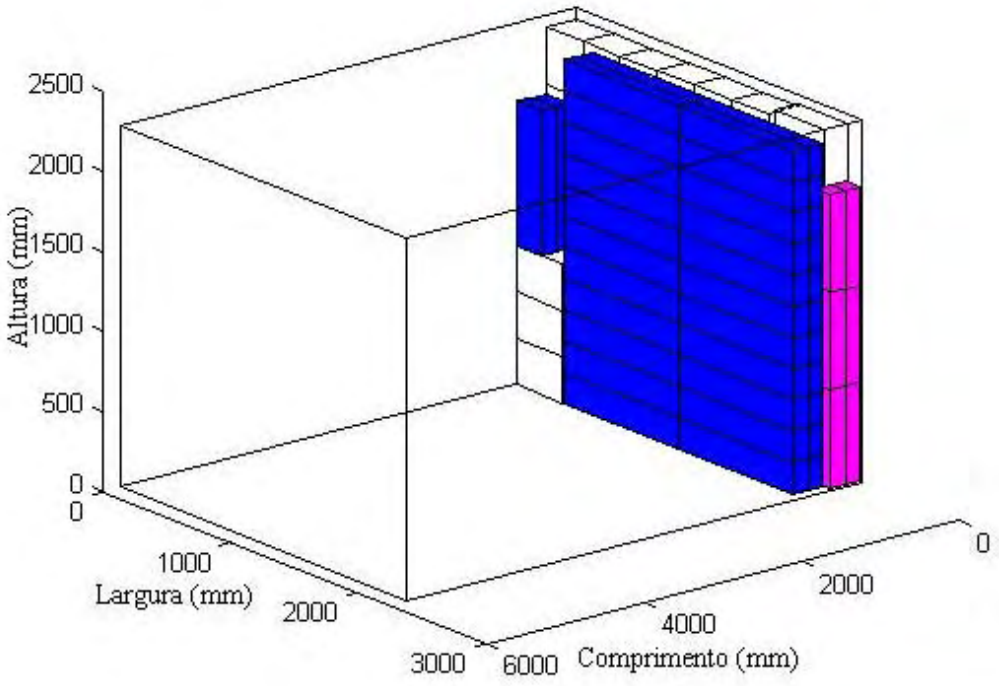
Escolhemos as caixas do tipo 2, de acordo com o ranque.

A melhor forma de carregar a camada seria $901 \times 195 \times 185$. Assim, a camada 03 assume a seguinte distribuição: $1 \times 2 \times 12$, isto é, a camada 03 é formada por 24 caixas do tipo 2.

- Largura Flexível $15 \times 1871 \times 2261$.
- Espaço residual de largura: $916 \times 1846 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 84 caixas do tipo 2 com a seguinte estrutura: $901 \times 195 \times 185$, isto é, $1 \times 7 \times 12 = 84$ caixas do tipo 2.

Figura 6: Carregamento da segunda camada com 49 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 7: Caixas restantes após a segunda camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	400	0,029788	11,9152
2	901	185	195	114	0,0325	3,7050
3	901	195	265	40	0,046559	1,8624
4	1477	135	195	40	0,038882	1,5553
5	614	480	185	2	0,054523	0,1091
6	400	400	135	16	0,0216	0,3456
7	264	400	400	80	0,04224	3,3792

Fonte: Informações da pesquisa da autora.

Existe espaço residual de largura de $916 \times 481 \times 2261$.

Podemos carregar ainda no espaço residual lateral mais 11 caixas do tipo 3, com a seguinte estrutura: $901 \times 265 \times 195$, isto é, $1 \times 1 \times 11 = 11$ caixas do tipo 3.

Existe espaço residual de largura de $916 \times 216 \times 2261$.

E mais 8 caixas do tipo 3 no espaço residual de largura, com a seguinte estrutura: $901 \times 195 \times 265$, isto é, $1 \times 1 \times 8 = 8$ caixas do tipo 3.

Resumo da terceira camada:

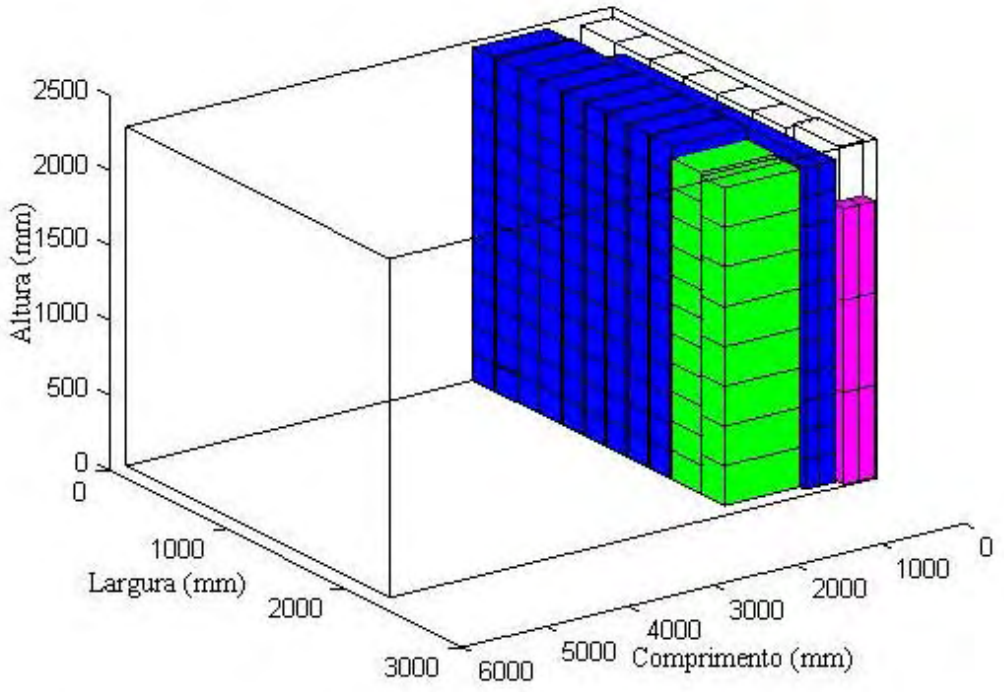
- 108 caixas do tipo 2 e 19 caixas do tipo 3 alocadas.
- Volume ocupado: 108(0,0325) + 19(0,046559) = 4,3946 m^3.
- Volume usado: $901 \times 2,236 \times 2,261 = 4,5551\ m^3$.
- Taxa de ocupação: 96,4771 %.
- Largura Flexível $15 \times 1846 \times 2261$.
- Profundidade usada: 1671 mm.
- Na Figura 7 apresentamos o carregamento da terceira camada, com 127 caixas.
- Na Tabela 8 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Tabela 8: Caixas restantes após a terceira camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	400	0,029788	11,9152
2	901	185	195	6	0,0325	0,1950
3	901	195	265	21	0,0325	0,6825
4	1477	135	195	40	0,038882	1,5553
5	614	480	185	2	0,054523	0,1091
6	400	400	135	16	0,0216	0,3456
7	264	400	400	80	0,04224	3,3792

Fonte: Informações da pesquisa da autora.

Figura 7: Carregamento da terceira camada com 127 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Camada 04:

Escolhemos as caixas do tipo 3, de acordo com o ranque.

A melhor forma de carregar a camada seria $901 \times 265 \times 195$. Assim, a camada 04 assume a seguinte distribuição: $1 \times 1 \times 11$, isto é, a camada 04 é formada por 11 caixas do tipo 3.

- Espaço residual de largura: $901 \times 1971 \times 2261$. Neste caso, a melhor opção é carregar 8 caixas do tipo 3 com a seguinte estrutura: $301 \times 195 \times 265$, isto é, $1 \times 1 \times 8 = 8$ caixas do tipo 3.

- Largura Flexível - Espaço residual de largura: $916 \times 1776 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 2 caixas do tipo 3 com a seguinte estrutura: $901 \times 195 \times 265$, isto é, $1 \times 1 \times 2 = 2$ caixas do tipo 3.

  Existe espaço residual acima do espaço de largura de $901 \times 195 \times 1731$. No espaço residual acima do espaço residual de largura, ou seja, acima das 2 caixas do tipo 3 podemos carregar mais 6 caixas do tipo 2 com a seguinte estrutura: $901 \times 195 \times 273$, isto é, $1 \times 1 \times 6 = 6$ caixas

do tipo 2.

Existe ainda espaço residual de largura de $916 \times 1581 \times 2261$.

Podemos carregar ainda no espaço residual de largura mais 96 caixas do tipo 1, com a seguinte estrutura: $139 \times 785 \times 273$, isto é, $6 \times 2 \times 8 = 96$ caixas do tipo 1.

Resumo da quarta camada:

- 96 caixas do tipo 1, 6 caixas do tipo 2 e 21 caixas do tipo 3 alocadas.
- Volume ocupado: 96(0,029788) + 6(0,0325) + 21(0,046559) = 3,939269 m^3.
- Volume usado: $901 \times 2,236 \times 2,261 = 4,5551\ m^3$.
- Taxa de ocupação: 86,4806 %.
- Largura Flexível $82 \times 1581 \times 2261$.
- Profundidade usada: 2572 mm.
- Na Figura 8 apresentamos o carregamento da quarta camada, com 123 caixas.
- Na Tabela 9 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Tabela 9: Caixas restantes após a quarta camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	304	0,029788	9,0556
4	1477	135	195	40	0,038882	1,5553
5	614	480	185	2	0,054523	0,1091
6	400	400	135	16	0,0216	0,3456
7	264	400	400	80	0,04224	3,3792

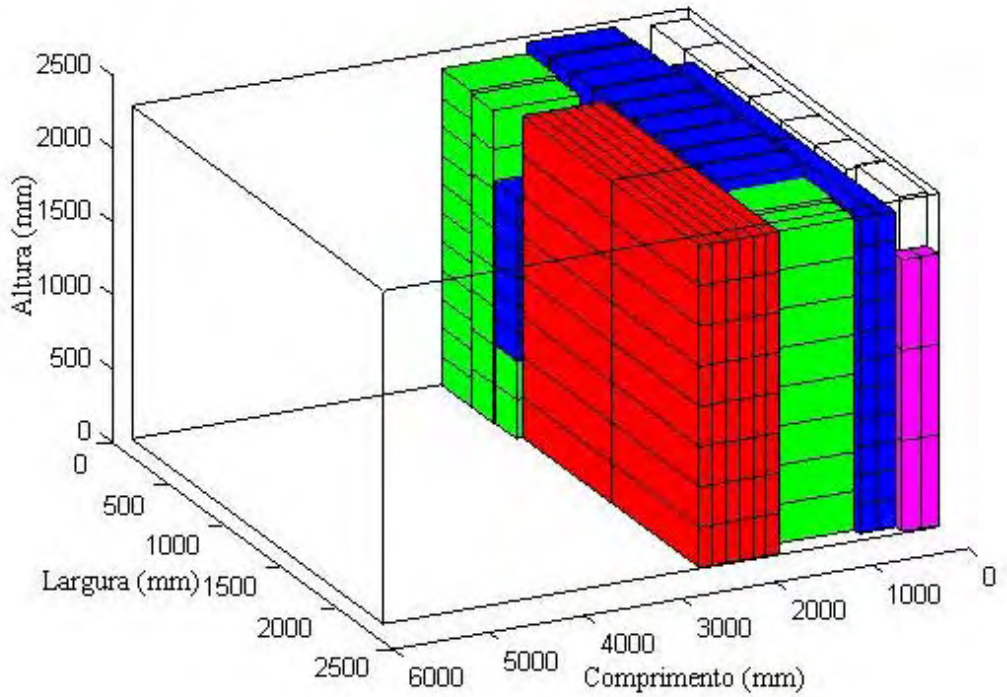
Fonte: Informações da pesquisa da autora.

Camada 05:

Escolhemos as caixas do tipo 5, de acordo com o ranque.

A melhor forma de carregar a camada seria $614 \times 480 \times 185$. Assim, a camada 05 assume a seguinte distribuição: $1 \times 1 \times 2$, isto é, a camada 05 é formada por 2 caixas do tipo 5.

Figura 8: Carregamento da quarta camada com 123 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

- Espaço residual de altura: $614 \times 480 \times 1891$

  No espaço residual de altura são alocadas 9 caixas do tipo 4 na forma: $195 \times 135 \times 1477$, isto é, $3 \times 3 \times 1 = 9$ caixas do tipo 4.

  Existe espaço residual acima do espaço residual de altura de $585 \times 405 \times 414$.

  No espaço residual acima do espaço residual da altura, ou seja, acima das 9 caixas do tipo 4 podemos carregar mais 4 caixas do tipo 6 com a seguinte estrutura: $135 \times 400 \times 400$, isto é, $4 \times 1 \times 1 = 4$ caixas do tipo 6.

- Largura Flexível - Espaço residual de largura: $629 \times 1756 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 4 caixas do tipo 1 com a seguinte estrutura: $273 \times 139 \times 785$, isto é, $2 \times 1 \times 2 = 4$ caixas do tipo 1.

  Existe espaço residual acima do espaço residual de largura de $546 \times 139 \times 691$.

Podemos carregar ainda no espaço residual mais 1 caixa do tipo 6, com a seguinte estrutura: $400 \times 135 \times 400$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 6.

- Largura Flexível - Espaço residual de largura: $696 \times 1581 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 80 caixas do tipo 1 com a seguinte estrutura: $139 \times 785 \times 273$, isto é, $5 \times 2 \times 8 = 80$ caixas do tipo 1.

Resumo da quinta camada:

- 84 caixas do tipo 1, 9 caixas do tipo 4, 2 caixas do tipo 5 e 5 caixas do tipo 6 alocadas.
- Volume ocupado: 84(0,029788) + 9(0,038882) + 2(0,054523) + 5(0,0216) = 3,0692 m^3.
- Volume usado: $614 \times 2,236 \times 2,261 = 3,1041\ m^3$.
- Taxa de ocupação: 98,8738 %.
- Largura Flexível: descartado.
- Profundidade usada: 3186 mm.
- Na Figura 9 apresentamos o carregamento da quinta camada, com 100 caixas.
- Na Tabela 10 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Tabela 10: Caixas restantes após a quinta camada

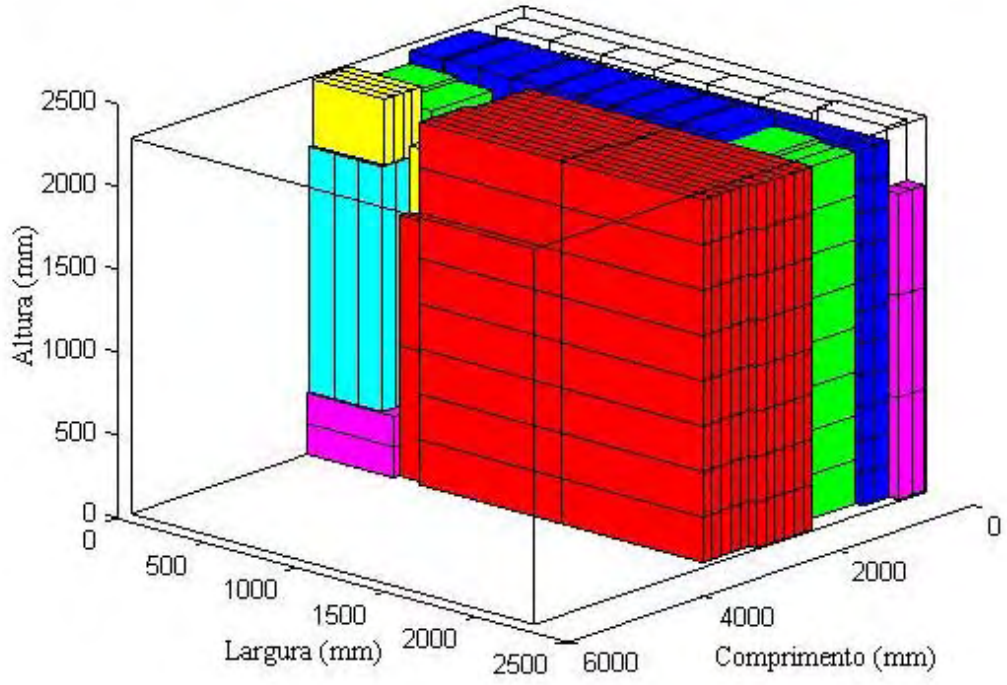
i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	220	0,029788	6,5534
4	1477	135	195	31	0,038882	1,2055
6	400	400	135	11	0,0216	0,2376
7	264	400	400	80	0,04224	3,3792

Fonte: Informações da pesquisa da autora.

Camada 06:

Escolhemos as caixas do tipo 1, de acordo com o ranque.

Figura 9: Carregamento da quinta camada com 100 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

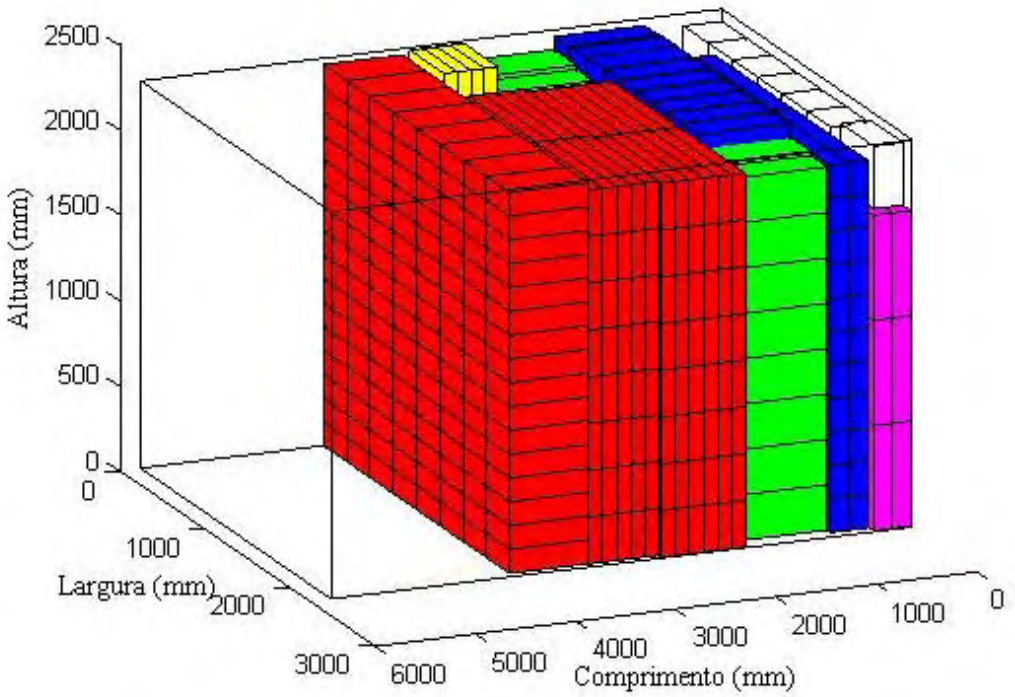
A melhor forma de carregar a camada seria $785 \times 273 \times 139$. Assim, a camada 06 assume a seguinte distribuição: $1 \times 8 \times 16$, isto é, a camada 06 é formada por 128 caixas do tipo 1.

Resumo da sexta camada:

- 128 caixas do tipo 1 alocadas.
- Volume ocupado: 128(0,029788) = 3,8129 m^3.
- Volume usado: $785 \times 2,236 \times 2,261 = 3,9686\ m^3$.
- Taxa de ocupação: 96,0767 %.
- Largura Flexível: não tem.
- Profundidade usada: 3971 mm.
- Na Figura 10 apresentamos o carregamento da sexta camada, com 128 caixas.

- Na Tabela 11 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Figura 10: Carregamento da sexta camada com 128 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 11: Caixas restantes após a sexta camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	785	139	273	92	0,029788	2,7405
4	1477	135	195	31	0,038882	1,2055
6	400	400	135	11	0,0216	0,2376
7	264	400	400	80	0,04224	3,3792

Fonte: Informações da pesquisa da autora.

Camada 07:

Escolhemos as caixas do tipo 1, de acordo com o ranque.

A melhor forma de carregar a camada seria $785 \times 273 \times 139$. Assim, a camada 07 assume a seguinte distribuição: $1 \times 5 \times 16$, isto é, a camada 07 é formada por 80 caixas do tipo 1.

- Espaço residual de largura: $785 \times 871 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 12 caixas tipo 1 com a seguinte estrutura: $785 \times 273 \times 139$, isto é, $1 \times 1 \times 12 = 12$ caixas do tipo 1.

  Existe espaço residual acima do espaço de largura de $785 \times 273 \times 593$.

  No espaço residual acima do espaço residual de largura, ou seja, acima das 12 caixas do tipo 1 podemos carregar mais 1 caixa do tipo 7 com a seguinte estrutura: $400 \times 264 \times 400$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 7.

- Espaço residual de largura: $785 \times 598 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 16 caixas do tipo 4 com a seguinte estrutura: $195 \times 135 \times 1477$, isto é, $4 \times 4 \times 1 = 16$ caixas do tipo 4.

  Existe espaço residual acima do espaço residual de largura de $780 \times 540 \times 784$.

  No espaço residual acima do espaço residual de largura, ou seja, acima das 16 caixas do tipo 4 podemos carregar mais 5 caixas do tipo 6 com a seguinte estrutura: $135 \times 400 \times 400$, isto é, $5 \times 1 \times 1 = 5$ caixas do tipo 6.

  Existe espaço residual acima do espaço residual de largura de $675 \times 400 \times 384$.

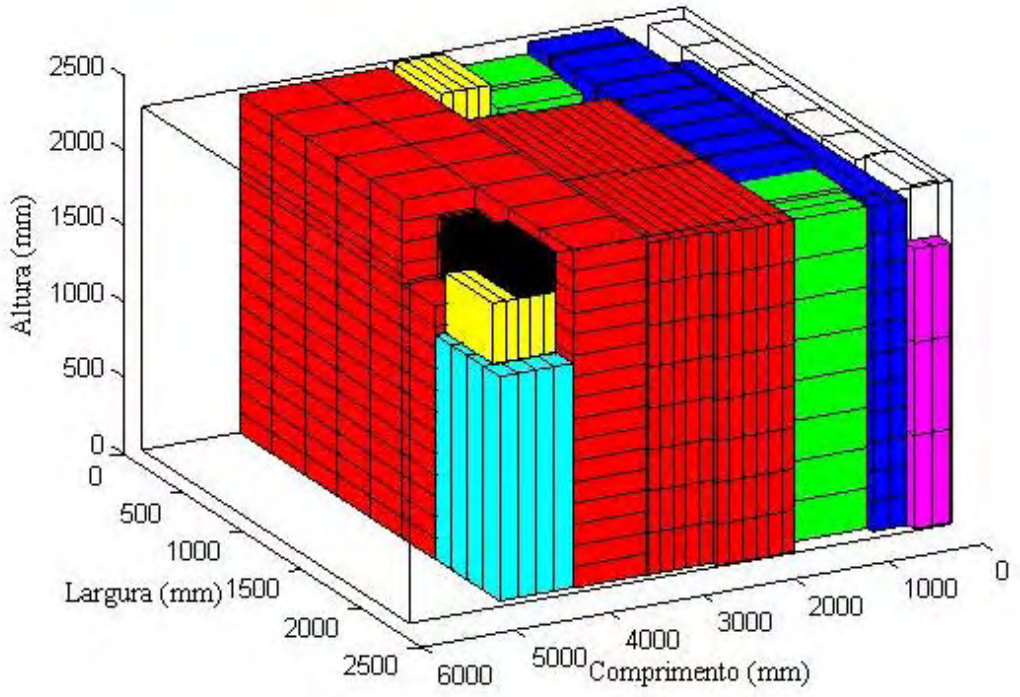
  No espaço residual acima do espaço residual de largura, ou seja, acima das 5 caixas do tipo 6 podemos carregar mais 1 caixa do tipo 7 com a seguinte estrutura: $400 \times 400 \times 264$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 7.

Resumo da sétima camada:

- 92 caixas do tipo 1, 16 caixas do tipo 4, 5 caixas do tipo 6 e 2 caixas do tipo 7 alocadas.
- Volume ocupado: 92(0,029788) + 16(0,038882) + 5(0,0216) + 2(0,04224) = 3,5551 m^3.
- Volume usado: $785 \times 2,236 \times 2,261 = 3,9686\ m^3$.
- Taxa de ocupação: 89,5804 %.
- Largura Flexível: $5 \times 598 \times 2261$.

- Profundidade usada: 4756 mm.
- Na Figura 11 apresentamos o carregamento da sétima camada, com 115 caixas.
- Na Tabela 12 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Figura 11: Carregamento da sétima camada com 115 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 12: Caixas restantes após a sétima camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
4	1477	135	195	15	0,038882	0,5832
6	400	400	135	6	0,0216	0,1296
7	264	400	400	78	0,04224	3,2947

Fonte: Informações da pesquisa da autora.

Camada 08:

Escolhemos as caixas do tipo 7, de acordo com o ranque.

A melhor forma de carregar a camada seria $400 \times 400 \times 264$. Assim, a camada 08 assume a seguinte distribuição: $1 \times 5 \times 8$, isto é, a camada 08 é formada por 40 caixas do tipo 7.

- Espaço residual de altura: $400 \times 2000 \times 149$.

  Neste caso pode ser usado esse espaço residual para alocar 5 caixas do tipo 6 com a seguinte estrutura: $400 \times 400 \times 135$, isto é, $1 \times 5 \times 1 = 5$ caixas do tipo 6.

- Largura Flexível - Espaço residual de largura: $405 \times 236 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 3 caixas tipo 4 com a seguinte estrutura: $135 \times 195 \times 1477$, isto é, $3 \times 1 \times 1 = 3$ caixas do tipo 4.

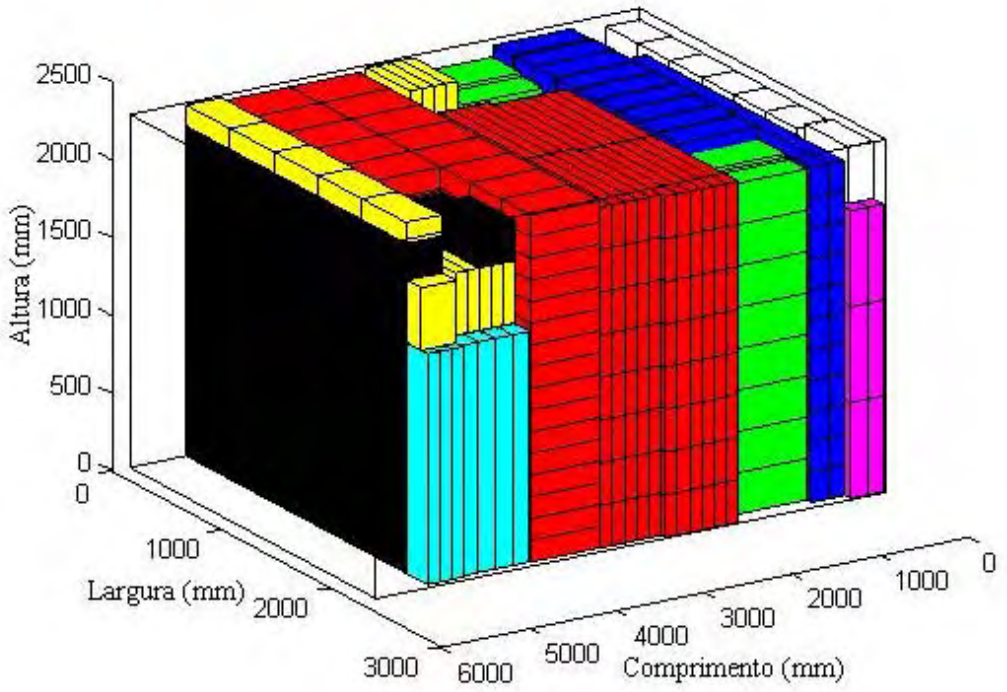
  Existe espaço residual acima do espaço residual de largura de $405 \times 195 \times 784$.

  No espaço residual acima do espaço residual de largura, ou seja, acima das 3 caixas tipo 4 podemos carregar mais 1 caixa do tipo 6 com a seguinte estrutura: $400 \times 135 \times 400$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 6.

Resumo da oitava camada:

- 3 caixas do tipo 4, 6 caixas do tipo 6 e 40 caixas do tipo 7 alocadas.
- Volume ocupado: 3(0,038882) + 6(0,0216) + 40(0,04224) = 1,9359 m^3.
- Volume usado: $400 \times 2,236 \times 2,261 = 2,0222\ m^3$.
- Taxa de ocupação: 95,7297 %.
- Largura Flexível: não tem.
- Profundidade usada: 5156 mm.
- Na Figura 12 apresentamos o carregamento da oitava camada, com 49 caixas.
- Na Tabela 13 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Figura 12: Carregamento da oitava camada com 49 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 13: Caixas restantes após a oitava camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
4	1477	135	195	12	0,038882	0,4666
7	264	400	400	38	0,04224	1,6051

Fonte: Informações da pesquisa da autora.

Camada 09:

Escolhemos as caixas do tipo 7, de acordo com o ranque.

A melhor forma de carregar a camada seria $400 \times 400 \times 264$. Assim, a camada 08 assume a seguinte distribuição: $1 \times 4 \times 8$, isto é, a camada 09 está formado por 32 caixas do tipo 7.

- Espaço residual de altura: $400 \times 400 \times 149$.

  O espaço residual de altura não pode ser utilizado, pois não tem caixas do tipo 6 disponíveis.

- Espaço residual de largura: $400 \times 636 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 5 caixas do tipo 7 com a seguinte estrutura: $400 \times 264 \times 400$, isto é, $1 \times 5 \times 1 = 5$ caixas do tipo 7.

- Espaço residual de largura: $400 \times 372 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 4 caixas do tipo 4 com a seguinte estrutura: $195 \times 135 \times 1477$, isto é, $2 \times 2 \times 1 = 4$ caixas do tipo 4.

Resumo da nona camada:

- 37 caixas do tipo 7 e 4 caixas do tipo 4 alocadas.
- Volume ocupado: 37(0,04224) + 4(0,038882) = 1,7184 m^3.
- Volume usado: $400 \times 2,236 \times 2,261 = 2,0222\ m^3$.
- Taxa de ocupação: 84,9772 %.
- Largura Flexível: não tem.
- Profundidade usada: 5556 mm.
- Na Figura 13 apresentamos o carregamento da nona camada, com 41 caixas.
- Na Tabela 14 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Tabela 14: Caixas restantes após a nona camada

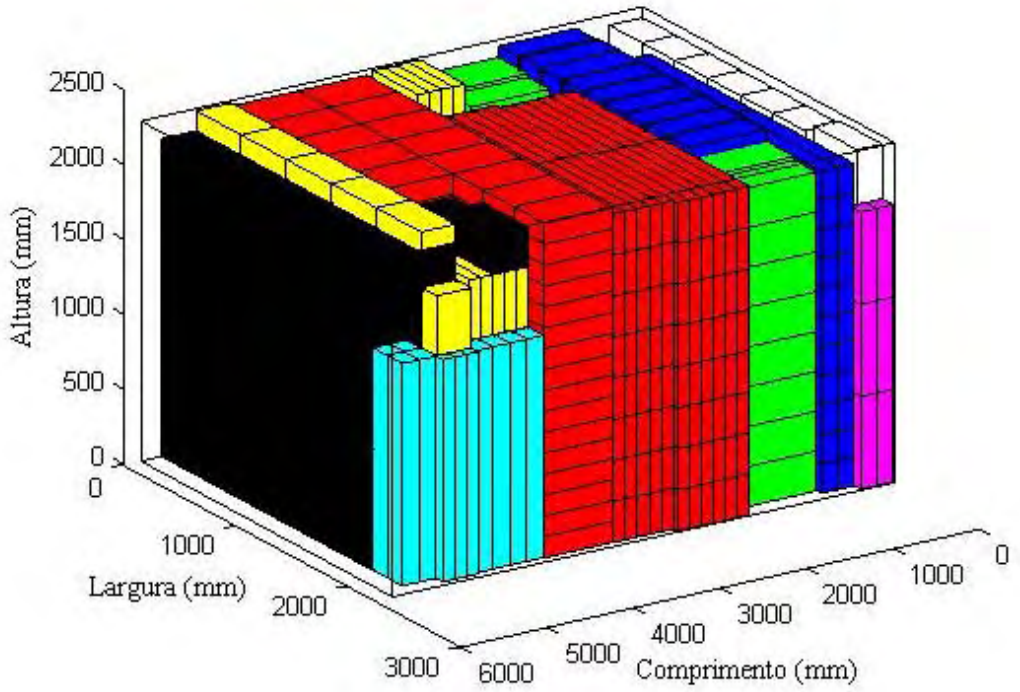
i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
4	1477	135	195	8	0,038882	0,3111
7	264	400	400	1	0,04224	0,04224

Fonte: Informações da pesquisa da autora.

Camada 10:

Escolhemos as caixas do tipo 4, a única que cabe.

Figura 13: Carregamento da nona camada com 41 caixas (GEORGE; ROBINSON, 1980).



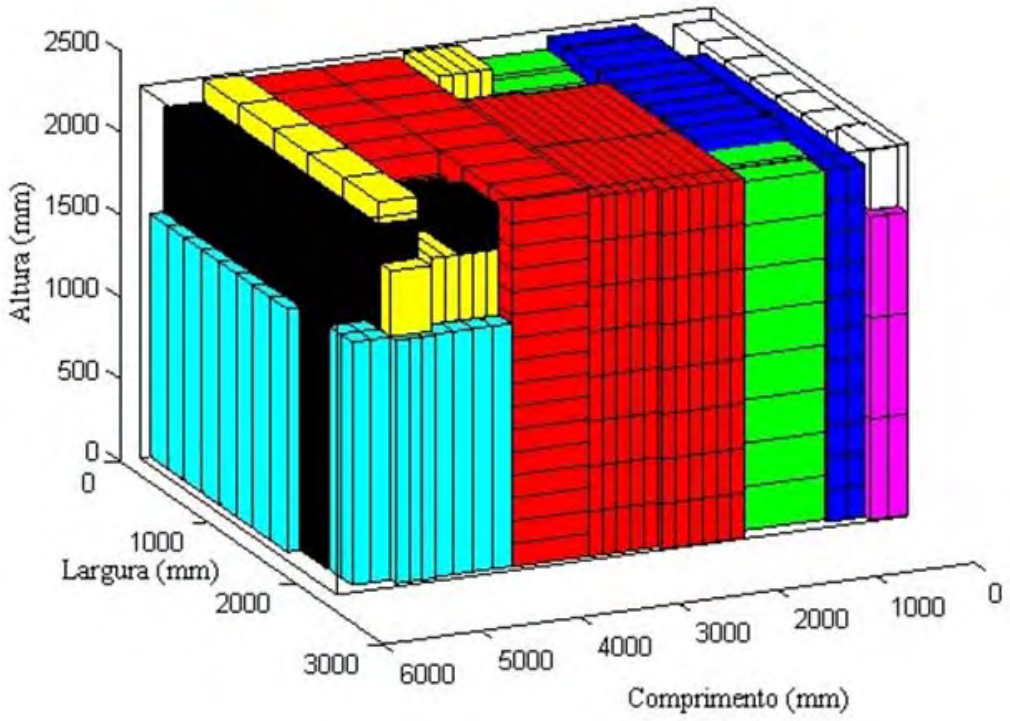
Fonte: Informações da pesquisa da autora.

A melhor forma de carregar a camada é na forma $195 \times 135 \times 1477$. Assim, a camada 10 assume a seguinte distribuição: $1 \times 8 \times 1$, isto é, a camada 10 é formada por 8 caixas do tipo 4.

Resumo da décima camada:

- 8 caixas do tipo 4 alocadas.
- Volume ocupado: 8(0,038882) = 0,31106 m^3.
- Volume usado: $195 \times 2,236 \times 2,261 = 0,9858\ m^3$.
- Taxa de ocupação: 31,5541 %.
- Profundidade usada: 5751 mm.
- Na Figura 14 apresentamos o carregamento da décima camada, com 8 caixas do tipo 4.
- Na Tabela 15 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar uma nova camada.

Figura 14: Carregamento da décima camada com 8 caixas (GEORGE; ROBINSON, 1980).



Fonte: Informações da pesquisa da autora.

Tabela 15: Caixas restantes após a décima camada

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
7	264	400	400	1	0,04224	0,04224

Fonte: Informações da pesquisa da autora.

Resumo da décima camada:

- Profundidade usada: 5751 mm.
- Espaço residual de profundidade: 42 mm.
- Taxa de ocupação total: 89,7381 %.

3.1.2 *A Heurística de Cecilio (2003)*

No trabalho de Cecilio (2003) são apresentados 5 novos métodos de solução para o PCC. Definindo 6 métodos de solução ligeiramente diferentes entre eles, compostos pela heurística original de George e Robinson (1980) e cinco variantes, Cecilio (2003) apresenta um refinamento simples da conhecida heurística de George e Robinson (1980) e apresenta também duas novas versões para carregar o contêiner que Cecilio (2003) chama de versão arranjo e de versão camadas. Os métodos são:

Método 1: heurística de George e Robinson (1980) original;

Método 2: método 1 + refinamento proposto;

Método 3: método 1 + versão Arranjo;

Método 4: método 3 + refinamento;

Método 5: método 1 + versão Camada;

Método 6: método 5 + refinamento;

As duas novas versões apresentada por Cecilio (2003), adicionadas às outras três de George e Robinson (1980), formam o total de cinco critérios de classificação na escolha do tipo de caixa para determinar o carregamento, sendo:

Critério 1: Caixa com a maior das menores dimensões. Segundo Cecilio (2003) a justificativa é que seria mais difícil ajustá-las depois.

Critério 2: Caixa com a maior quantidade disponível. A jusficativa seria, porque preenche a maior parte da camada.

Critério 3: Caixa com a maior dimensão.

Critério 4: Caixa com maior volume.

Critério 5: Caixa com a maior razão dada por maior dimensão / menor dimensão.

Os 5 critérios de decisão são executadas, utilizando um arranjo de três dos cincos critérios de classificação ($A_{5,3} = \frac{5!}{2!} = 60$). A justificativa para a escolha de apenas três entre os cincos critérios de classificação é que dificilmente o desempate entre as caixas passa do terceiro critério.

Cecilio (2003) apresenta duas metodologias de carregamento de contêiner, sendo chamado de Versão Arranjo e Versão Camadas.

Versão Arranjo da heurística de George e Robinson (1980)

Na versão Arranjo, a heurística então é executada 60 vezes e, a cada iteração um arranjo de três critérios diferentes é utilizado na execução do carregamento completo do contêiner, então o melhor

resultado, em termos de volume carregado é atualizado em cada iteração.

Versão Camada da heurística de George e Robinson (1980)

Neste método Cecilio (2003) apresenta, para cada camada do contêiner, uma heurística que é executada 60 vezes, esgotando todas as possibilidades de selecionar o melhor tipo de caixa possível para o carregamento.

Assim, a heurística elaborada por Cecilio (2003) começa a carregar uma nova camada após executar 60 vezes um arranjo de três critérios de classificação diferentes. Esse procedimento é repetido sempre que uma nova camada iniciar e o melhor resultado de cada camada é determinado pela razão volume de caixas carregadas em relação ao volume parcial do contêiner.

Refinamento de Cecilio (2003) da heurística original de George e Robinson (1980)

Um simples refinamento da heurística de George e Robinson (1980) pode ser obtido alterando um único procedimento do algoritmo heurístico original. A alteração é o da escolha do tipo de caixa para os espaços residuais de cada camada, ao contrário de George e Robinson (1980) que escolhe o tipo de caixa que preenche a maior área da base do espaço residual a ser preenchido. A heurística de Cecilio (2003) modifica esse passo do algoritmo fazendo uma combinação de caixas iguais em relação ao comprimento do espaço a ser carregado, sendo escolhido o tipo que melhor utilizar a área da base.

Resultados do algoritmo heurístico de Cecilio (2003)

Com o refinamento da heurística proposta por Cecilio (2003) obteve-se o resultado de 89,9% do volume do contêiner, assim conseguiu-se carregar todas as 784 caixas dentro do contêiner, utilizando 10 camadas. Porém não é mencionado o tamanho do espaço residual frontal dentro do contêiner.

Cecilio (2003) também faz diversos outros testes, porém os dados destes não se encontram disponíveis no trabalho da autora.

3.1.3 *A Heurística de Pisinger (2002)*

O algoritmo heurístico apresentado por Pisinger (2002) aparentemente é uma das melhores propostas de otimização apresentada na literatura especializada para otimizar o PCC. Embora seja possível classificar o algoritmo como sendo heurístico (porque não garante encontrar a solução ótima do problema), deve-se observar que essa proposta de otimização apresenta componentes de elevada sofisticação. A lógica fundamental da proposta de Pisinger (2002) é realizar o carregamento do contêiner usando a lógica de preenchimento através de camadas. Nessa proposta, a caixa que abre

uma nova camada é escolhida de uma forma muito mais sofisticada que outras propostas de otimização. Da mesma forma, uma vez escolhida a profundidade da camada, podemos escolher a dimensão de largura da caixa também de forma eficiente. Finalmente, após definir a profundidade e a largura de um preenchimento, a alocação das caixas na altura pode ser realizado usando uma técnica de otimização já que essa parte do problema é equivalente ao problema da mochila. Portanto, o preenchimento ótimo de uma pequena parcela do problema torna a proposta de otimização mais eficiente e também mais demorada, já que em alguns casos de testes pode ser necessário resolver até a otimalidade, centenas de milhares de pequenos problemas do tipo mochila. Assim, a resolução rápida e eficiente desses subproblemas da mochila é crucial na proposta de otimização de Pisinger (2002).

Um detalhe importante da proposta de Pisinger (2002) é que as caixas podem ser alocadas apenas sem sobreposição (as caixas ou partes das caixas não podem ocupar o mesmo espaço), mas sem levar em conta o suporte das caixas (uma caixa não precisa de superfície de contato pleno com a caixa ou caixas alocadas abaixo dela) já que os espaços livres podem ser preenchidos com material de transporte adequado dando suporte adequado para as caixas.

Pisinger (2002) também faz uma classificação interessante sobre as técnicas heurísticas usadas na resolução do PCC. Essa classificação é a seguinte: (1) algoritmos de formação de camadas (wall building algorithms) que foi introduzido por George e Robinson (1980) onde o contêiner é carregado por uma sequência de camadas de profundidade definida, (2) algoritmo de formação de empilhamento (stack building algorithm) onde as caixas são empilhadas em uma espécie de torre (uma caixa acima da outra até atingir a altura do contêiner) e depois esses empilhamentos (torres) são alocados no piso do contêiner resolvendo um problema de empacotamento (corte) bidimensional, (3) algoritmos de corte tipo guilhotina (guillotine cutting algorithms) que é baseado na representação em árvore do problema de empacotamento e, (4) algoritmos de carregamento tipo cuboide (cuboid arrangement algorithms) em que o preenchimento do contêiner é realizado preenchendo o contêiner com arranjos cubóides (uma caixa formada por várias outras caixas).

Pisinger (2002) chama sua proposta de otimização como sendo uma busca em árvore. Assim, para abrir uma nova camada existem muitas alternativas (na verdade todas as dimensões existentes nas caixas livres para serem utilizadas no preenchimento da camada) e para cada uma dessas alternativas existe ainda a escolha da largura inicial da camada que se pretende preencher. Como a análise de todas as combinações possíveis teria caráter combinatório, Pisinger (2002) sugere limitar essas alternativas. Assim, a proposta de Pisinger (2002) consiste em limitar as alternativas da escolha de profundidade para as M_1 melhores alternativas e para a escolha da tira de largura (ou altura) as M_2 melhores

alternativas. Essas melhores alternativas são escolhidas usando um ranque de profundidade e de largura. Pisinger (2002) apresenta 27 alternativas diferentes para a escolha das melhores alternativas de profundidade e da largura. Para isso, deve-se levar em conta que as funções de ranque estão baseadas em estatísticas das dimensões das caixas ainda não alocadas no contêiner. Assim, seja α a menor dimensão e β a maior dimensão entre as caixas ainda disponíveis para preenchimento e que podem ser usadas na geração da nova camada. Sejam w_i, h_i e d_i a largura, altura e profundidade de uma caixa livre i e seja k uma dimensão existente entre as caixas livres. Nesse contexto são definidas as seguintes funções de frequência:

1. f_k^1 é o número de vezes que existe a dimensão k entre as caixas livres e considerando todas as dimensões w_i, h_i e d_i das caixas.

2. f_k^2 é o número de vezes que existe a dimensão k entre as caixas livres e considerando apenas a maior dimensão de cada caixa.

3. f_k^3 é o número de vezes que existe a dimensão k entre as caixas livres e considerando apenas a menor dimensão de cada caixa.

Deve-se observar que as funções de frequência definidas anteriormente tenta identificar as maiores dimensões (com a pretensão de escolher essas caixas primeiro e evitar dificuldades futuras) e as dimensões mais frequentes (assim essas caixas podem ser juntadas no preenchimento e, dessa forma, melhorar o preenchimento). As seguintes regras de prioridade tentam encontrar um compromisso entre esses objetivos no preenchimento:

1. Escolher as M maiores dimensões existentes.

2. Escolher as maiores dimensões com incremento de frequência, isto é, primeiro escolhemos a maior dimensão k com $f_k \geq 1$. Depois escolhemos a maior dimensão $k^{'}$ com $f_k^{'} > f_k$. Repetir esse processo até que sejam escolhidas M dimensões.

3. Escolher as maiores dimensões com pseudo incremento da frequência (primeira versão), isto é, primeiro escolhemos a maior dimensão k com $f_k \geq 1$. Depois escolhemos a maior dimensão $k^{'}$ com $f_k^{'} \geq 2$. Assim, a i-ésima dimensão escolhida deve ter frequência $f_k \geq i$.

4. Escolher as maiores dimensões com pseudo incremento da frequência (segunda versão), isto é, primeiro escolhemos a maior dimensão k com $f_k \geq 1$. Depois escolhemos a maior dimensão

k' com $f'_k \geq 2$. A i-ésima dimensão escolhida deve ter frequência $f_k \geq \frac{4i}{M}$, onde M é o número de dimensões que se pretende escolher.

5. Escolher as dimensões de maior frequência, isto é, as M dimensões tipo k com os maiores valores de f_k.

6. Esta regra de prioridade é equivalente com a regra 2, mas se forem selecionados menos que M dimensões, então a regra de prioridade 1 é usada para terminar de selecionar as dimensões restantes.

7. Esta regra de prioridade é equivalente à regra 3, mas usando a regra 1 se menos que M dimensões são selecionadas.

8. Esta regra de prioridade é equivalente à regra 3, mas usando a regra 5 se menos que M dimensões, são selecionadas.

9. Esta regra de prioridade é equivalente à regra 4, mas usando a regra 5 se menos que M dimensões, são selecionadas.

Pisinger (2002) realizou testes usando todas as combinações possíveis, isto é, usando 27 critérios para a seleção das M_1 melhores alternativas para definir a profundidade de uma camada e as M_2 melhores alternativas para definir a largura (em cada caso existem 3 funções de frequência para escolha e 9 regras de prioridade). Nesse contexto, a heurística de George e Robinson (1980) é equivalente a uma implementação com a função de frequência f^3 e a regra de prioridade 1 na parte relacionada com a escolha da profundidade da camada. Nos testes, a função de frequência f^1 se mostrou superior à regra de prioridade 8. Em outras palavras, os melhores resultados foram obtidos usando as dimensões que aparecem com maior frequência considerando todas as dimensões das caixas livres para escolher a profundidade de uma camada e, entre essas dimensões, considerando as maiores dimensões.

Obviamente, o ponto forte da heurística de Pisinger (2002) é a proposta de preenchimento ótimo após definir a profundidade e a largura inicial de preenchimento. Assim, essa parcela do problema é resolvida de forma ótima usando um algoritmo que encontra a solução ótima do problema da mochila chamado de ”minknap´´ e desenvolvido pelo próprio autor. Os resultados encontrados parecem excelentes, especialmente para o caso em que o volume total das caixas é maior que o volume do contêiner onde as taxas de preenchimento estão na faixa de 94,20 % a 96,57 %.

A partir de uma análise da proposta de Pisinger (2002), deve-se realizar as seguintes observações:

- Embora a proposta de otimização seja classificada como uma heurística pelo próprio autor, a proposta de otimização é muito sofisticada e pode competir em termos de qualidade final de solução encontrada com as melhores metaheurísticas existentes na literatura especializada para resolver o PCC. O ponto alto da proposta está na resolução de pequenos subproblemas que aparecem na estratégia de otimização usando um algoritmo de otimização muito rápido e eficiente (um método exato para resolver o problema da mochila).

- A proposta de Pisinger (2002) contorna de forma eficiente a escolha da caixa que abre a camada. Nossa experiência mostra que essa escolha é crucial para o desempenho dos algoritmos que usam a lógica de preenchimento de camadas de forma independente. Pisinger (2002) escolhe M_1 alternativas diferentes e resolve o problema de preenchimento da camada para cada uma dessas M_1 propostas e escolhe como proposta final a melhor solução no preenchimento da camada.

Em geral, a proposta de Pisinger (2002) também pode ser considerado como uma proposta eficiente de otimização do PCC. Uma crítica que pode ser feita a este trabalho é o mesmo que da maioria de algoritmos que usam a lógica de geração de camadas em sequência. Nesse tipo de lógica de otimização Pisinger (2002) contorna de forma eficiente a escolha da caixa que abre a camada, mas persiste a lógica de geração de camadas em sequência que não permite gerar soluções que poderiam ser geradas usando a geração de arranjos (multicamadas). Em outras palavras, o espaço de busca também é menor que o espaço de busca quando geramos uma solução usando arranjos (multicamadas).

3.2 Metaheurísticas Usadas no Problema de Carregamento do Contêiner

Existem muitas propostas de otimização da família das metaheurísticas usadas na solução do PCC. Entretanto, neste trabalho analisamos com detalhes o algoritmo genético de Rodrigues (2005), o algoritmo genético de Gehring e Bortfeldt (1997), o algoritmo híbrido de Bortfeldt e Gehring (2001) e o algoritmo GRASP de Leite (2007).

3.2.1 *Algoritmo Genético de Rodrigues (2005)*

Rodrigues (2005) desenvolveu um algoritmo genético para o PCC como parte de uma dissertação de mestrado. Segundo o próprio autor, o algoritmo genético desenvolvido é uma integração do algoritmo genético desenvolvido por He e Cha (2002) e do algoritmo genético desenvolvido por Gehring e Bortfeldt (1997).

Segundo o autor, uma das propostas originais de He e Cha (2002) é apresentar o PCC como um problema multiobjetivo com 3 objetivos (maximizar o volume utilizado pelo contêiner, maximizar o peso de utilização do contêiner e minimizar a altura do centro de gravidade). Entretanto, tanto He e Cha (2002) assim como Rodrigues (2005) usam uma função objetivo unificada integrando as funções objetivo por uma combinação linear das funções objetivo. Adicionalmente, no trabalho de Rodrigues (2005) é incorporado um objetivo adicional, isto é, a maximização do valor monetário associado à carga.

Na implementação do algoritmo genético usado por Rodrigues (2005) é usada uma proposta de codificação de uma proposta de solução do PCC usando um vetor de dimensão n para um problema com n caixas. Nessa proposta, todas as caixas são numeradas de 1 a n. Portanto, qualquer vetor p_c de tamanho n formado internamente com os números de 1 a n representa uma proposta de codificação e, portanto, uma proposta de solução. Deve-se observar que essa proposta de solução indica a ordem em que as caixas devem ser alocadas no contêiner. Como as caixas podem ser alocadas de diferentes formas, então essa proposta de solução precisa de uma decodificação que indica a forma exata em que devem ser alocadas as caixas, isto é, uma proposta de solução deve ser mostrada através de um padrão de carregamento (packing pattern). Esse padrão de carregamento é encontrado a partir da proposta de codificação de uma proposta de solução. Nesse contexto, Rodrigues (2005) apresenta duas propostas de decodificação:

- Uma proposta de decodificação usada por He e Cha (2002) em que se percorre o vetor de codificação e cada caixa dessa sequência é alocada na ordem L (lateral), M (superior) e R (frontal) sempre que possível. Nessa estratégia, uma vez escolhida a caixa que abre uma nova camada, primeiro deve ser preenchido o espaço residual lateral, depois o espaço residual de altura e posteriormente o espaço residual frontal o que significa abrir uma nova camada.

- Uma proposta de decodificação usada por Gehring e Bortfeldt (1997) em que se percorre o vetor de codificação e cada caixa dessa sequência é alocada na ordem M (superior), L (lateral)

e R (frontal) sempre que possível. Nessa estratégia, uma vez escolhida a caixa que abre uma nova camada, primeiro deve ser preenchido o espaço residual de altura, depois o espaço residual lateral e posteriormente o espaço residual frontal o que significa abrir uma nova camada. Deve-se observar que após escolher a primeira caixa que inicia uma nova camada, deve-se preencher o espaço residual de altura, isto é, deve-se tratar de alocar caixas acima dessa caixa formando uma torre. A seguir começa uma nova torre usando o espaço residual lateral até esgotar esse espaço residual lateral. Nesse contexto, as torres são disjuntas, isto é, uma caixa deve fazer parte apenas de uma torre. A ideia de formação de torres foi anteriormente proposta por Gehring e Bortfeldt (1997). Entretanto, em Gehring e Bortfeldt (1997) as torres são formadas de forma independente e o algoritmo genético faz a alocação da base da torre na superficie do contêiner, isto é, uma vez formadas as torres, a alocação das torres é equivalente a um problema de corte já que se trata apenas de alocar a base da cada torre na superficie do contêiner.

O algoritmo genético de Rodrigues (2005) faz testes usando as duas propostas de decodificação, isto é, das duas formas de geração do padrão de carregamento. Assim, em dois problemas testados, a sequência L, M e R foi superior e no terceiro caso a sequência M, L e R foi superior. Em relação à qualidade dos resultados encontrados, o algoritmo genético de Rodrigues (2005) foi inferior quando resolveu o mesmo problema usado por He e Cha (2002) (um problema com 50 caixas diferentes). Entretanto, uma comparação rigorosa não é possível porque o algoritmo de Rodrigues (2005) não permite rotacionar a altura das caixas (representando um problema mais restrito), não foram usados os mesmos pesos na formação de uma única função objetivo obtido através de combinação linear e incorpora uma função objetivo adicional (o custo da carga).

Em relação ao algoritmo genético de Rodrigues (2005), pode-se fazer as seguintes observações adicionais:

- Dois testes correspondem a PCCs reais de empresas de Manaus, mostrando uma interação com o setor industrial.

- Os resultados encontrados com os dois testes de sistemas reais não são de excelente qualidade já que testes realizados posteriormente encontraram soluções de melhor qualidade. Uma explicação possível é a forma de codificação e de decodificação da proposta de solução. Por exemplo, a estratégia de formação de torres gera muita perda de volume já que uma caixa alocada acima da outra deve ter superficie de contato plena. Outro problema é a escolha da caixa que inicia uma nova camada. A caixa que inicia uma nova camada é qualquer uma que

se encontra como o próximo elemento a ser avaliado no vetor de codificação. Deve-se observar que a caixa que inicia uma nova camada é muito importante para ser escolhida de forma quase aleatória.

- A decodificação leva em conta apenas a restrição de volume da caixa, isto é, se existe a possibilidade de alocar a caixa sob análise no espaço residual correspondente e não é levada em conta as outras restrições (peso, valor e equilíbrio).

- Não está muito claro o processo de decodificação. Por exemplo, se uma caixa no vetor de codificação não for escolhida para entrar no padrão de carregamento então não está claro o instante em que deve ser tentada novamente a alocação dessa caixa. Aparentemente, após percorrer o vetor de codificação até o final, deve-se voltar ao ponto inicial para tentar alocar as caixas que ainda não foram alocadas. Assim, a forma de decodificação é um dos problemas críticos no algoritmo genético de Rodrigues (2005).

- Um outro detalhe no algoritmo genético de Rodrigues (2005) é a lógica de alocação através de camadas. Nesse contexto, a escolha da primeira caixa define as dimensões da camada e dos espaços residuais de largura, altura e frontal. Essa característica corresponde a todas as propostas que fazem o preenchimento através de uma sequência de camadas. Uma proposta alternativa seria ver a possibilidade de usar multicamadas ou arranjos usados, por exemplo, neste trabalho.

A função objetivo usada em Rodrigues (2005) assume a seguinte forma:

$$f(p) = \frac{k_1 R + k_2 W + k_3 G + k_4 V}{k_1 + k_2 + k_3 + k_4} \tag{23}$$

Na relação anterior, k_1, k_2, k_3 e k_4 são fatores de pesos e valores conhecidos e R, W, G e V são funções relacionados com o volume, peso, centro de gravidade e valor da carga da seguinte forma:

$$R = 100 \quad \frac{\sum_{i=1}^{m} R_{Bi}}{R_c} \tag{24}$$

onde R é a função de volume, R_{Bi} é o volume de cada caixa carregada em que i representa o índice da caixa, m o número de caixas carregadas e R_c é o volume disponibilizado pelo contêiner.

$$W = \begin{cases} 0 & \text{se } \sum_{i=1}^{m} W_{Bi} > W_c \\ 100 \ \frac{\sum_{i=1}^{m} W_{Bi}}{W_c} & \text{se } \sum_{i=1}^{m} W_{Bi} \leq W_c \end{cases} \tag{25}$$

onde W é a função de peso, W_{Bi} é o peso de cada caixa carregada em que i representa o índice da caixa, m é o número de caixas alocadas, W_c é o peso máximo suportado pelo contêiner. Assim, o objetivo é maximizar o peso das caixas carregadas no contêiner sem ultrapassar o limite máximo de peso suportado pelo contêiner.

$$G = 100 \ \frac{1,5H_c - \frac{\sum_{i=1}^{m} W_{Bi}G_{Bi}}{\sum_{i=1}^{m} W_{Bi}}}{H_c} \tag{26}$$

onde G é a função de gravidade, W_{Bi} é o peso de cada caixa carregada em que i representa o índice da caixa, m é o número de caixas alocadas, G_{Bi} representa o centro de gravidade da caixa i e assume-se que o valor médio da altura da caixa é o seu centro de gravidade e H_c é a altura do contêiner. Deve-se observar que se o centro de gravidade das caixas carregadas é igual a 0,5 H_c, então $G = 100\%$. Para valores menores G assume um valor maior de 100%.

$$V = \begin{cases} 0 & \text{se } \sum_{i=1}^{m} V_{Bi} > W_c \\ 100 \ \frac{\sum_{i=1}^{m} V_{Bi}}{V_c} & \text{se } \sum_{i=1}^{m} V_{Bi} \leq V_c \end{cases} \tag{27}$$

onde V é a função de custo da carga alocada no contêiner, V_{Bi} é o valor de cada caixa carregada em que i representa o índice da caixa, m é o número de caixas alocadas, V_c é o valor monetário máximo que a carga do contêiner deve possuir. Assim, o objetivo é maximizar o valor monetário das caixas carregadas no contêiner sem ultrapassar o limite máximo de valor monetário suportado pelo contêiner.

Deve-se observar que em (**??**) todas as funções estão em unidades porcentuais, isto é, se encontram parametrizadas.

3.2.2 *Algoritmo Genético de Gehring e Bortfeldt (1997)*

Em Gehring e Bortfeldt (1997) apresentam um algoritmo de otimização para resolver o PCC. Os autores desenvolveram um algoritmo genético em que a ideia central consiste em decompor o processo de solução em duas fases. Na primeira fase é realizada a formação de torres com as caixas e na segunda fase as torres são alocadas na superfície do contêiner. A fase de formação de torres é realizada usando uma estratégia heurística e a fase de alocação das torres na superfície do contêiner é realizada usando um algoritmo genético. Segundo os autores, o algoritmo desenvolvido é especialmente eficiente no carregamento de caixas fortemente heterogêneas.

No trabalho de Gehring e Bortfeldt (1997) são consideradas algumas restrições que não eram consideradas anteriormente. Assim, a metodologia desenvolvida considera as seguintes restrições:

- Restrições de orientação: Cada caixa pode ser alocada de forma restrita, isto é, uma ou duas das dimensões da caixa não podem ser orientadas na direção vertical.
- Restrições de alocação das caixas no topo: Um subconjunto de caixas não podem suportar o peso e, portanto, não existe a possibilidade de alocar caixas acima dessas caixas.
- Restrição de peso: O peso de uma carga completa não pode ultrapassar um valor de peso previamente especificado.
- Restrição de estabilidade: A estabilidade da caixa é calculada como sendo a relação entre a área de contato da caixa com as caixas alocadas abaixo dela com a área total de contato da caixa (base da caixa).
- Restrição de balanço: A distância entre a componente no eixo x do centro de gravidade da carga e o ponto médio na profundidade do contêiner deve ser menor que um valor previamente especificado. O mesmo deve acontecer com o eixo y.

O algoritmo genético de Gehring e Bortfeldt (1997) pode ser resumido da seguinte forma:

1. Na primeira fase as caixas são arranjadas em um conjunto de torres. A fase de formação de torres é realizada usando um algoritmo guloso que tende a minimizar o espaço perdido na alocação de caixas acima da caixa base. A definição de torre também é claramente especificada no trabalho de Gehring e Bortfeldt (1997). Assim, por exemplo, uma caixa pode ser alocada acima de outra se tem uma superfície de contato plena, e não é possivel alocar uma caixa acima de duas caixas, mas podem ser alocadas mais de uma caixa acima de uma única caixa.

2. Na segunda fase, deve-se alocar as bases das torres na superfície do contêiner. Esse problema bidimensional é resolvido usando um algoritmo genético que maximiza o volume total das caixas alocadas no contêiner.

3. As duas fases podem ser repetidas muitas vezes. O processo de repetição inclui a geração de novas variantes na formação de torres e de cobertura da superfície do contêiner. No final, a melhor solução encontrada é a proposta de solução da técnica de otimização proposta.

Em relação ao algoritmo genético desenvolvido por Gehring e Bortfeldt (1997), podemos fazer as seguintes observações:

1. O algoritmo genético apresenta alguns aspectos de sofisticação. Inicialmente é desenvolvido um algoritmo genético em que se evita a formação de duplicação de soluções na formação da população corrente.

2. Na proposta apresentada também se analisa com certo detalhe o problema de codificação, decodificação e representação de uma proposta de solução. Assim, um cromossomo (uma forma de representar uma proposta de solução) está representado por um vetor chamado $chrom(.)$ em que deve aparecer a informação da sequência em que as torres devem ser alocadas na superficie do contêiner, assim como a orientação dessas torres. Portanto, um elemento do vetor de alocação, $chrom(i)$ é uma estrutura de dados que armazena o identificador de torre e sua orientação.

3. A solução codificada como mostrada no item anterior deve ser decodificada. Assim, uma solução é encontrada após a decodificação da solução mostrada em $chrom(.)$ e armazenada no vetor $sol(.)$. Um elemento desse vetor, $sol(i)$ é também uma estrutura de dados, formada por quatro componentes, isto é, o identificador da torre, a rotação e as coordenadas x e y em relação a origem (o canto inferior esquerdo do contêiner). O processo de decodificação, que

transforma uma solução apresentada a nível de cromossomo como sendo uma sequência de torres e sua respectiva orientação, é um tanto complexa e os detalhes podem ser encontrados em Gehring e Bortfeldt (1997).

4. Considerando a forma de codificação escolhida para o PCC, a implementação da recombinação e da mutação é bastante trivial. Gehring e Bortfeldt (1997) mencionam que implementaram 3 tipos diferentes de recombinação e dois tipos de mutação.

5. A versão final do algoritmo desenvolvido inclui a formação da população inicial diferente da aleatória. Assim é usada uma heurística na formação da população inicial. Essa heurística também é usada no processo de decodificação em que se pode, por exemplo, mudar a orientação de uma torre.

6. Os resultados apresentados são excelentes quando comparados com várias proposta de otimização apresentadas anteriormente.

Em relação ao algoritmo genético apresentado por Gehring e Bortfeldt (1997) podemos fazer as seguintes observações adicionais:

- O algoritmo está adequadamente apresentado e formulado. O algoritmo genético se encontra muito bem desenvolvido. Os resultados apresentados também são excelentes. O processo de codificação e decodificação de uma proposta de solução estão adequadamente formulados, mas a implementação é um tanto complexa.

- Um problema na proposta de otimização é a separação do processo de solução em duas partes. Assim, a formação de torres pode gerar muita perda de espaço não usado e limitar o espaço de busca da fase de alocação de torres na superficie do contêiner usando o algoritmo genético.

3.2.3 *Algoritmo Genético Híbrido de Bortfeldt e Gehring (2001)*

Bortfeldt e Gehring (2001) apresentaram um excelente algoritmo de otimização para resolver o PCC. Bortfeldt e Gehring (2001) analisaram com certo detalhe o delicado problema de codificação ou de representação de uma proposta de solução para o PCC e sob a influência das propostas

desenvolvidas por Michalewicz (1994). Bortfeldt e Gehring (2001) reconhecem em 2001 que apesar de que já existem várias propostas de otimização para resolver o PCC, o desenvolvimento de métodos com capacidade de encontrar soluções com o elevado índice de utilização do volume do contêiner ou soluções quase ótimas ainda é um desafio. Influenciado pelas ideias de Michalewicz (1994), Bortfeldt e Gehring (2001) escrevem o seguinte:

- As soluções ou propostas de solução devem ser representadas naturalmente através de estruturas de dados complexas, tais como grafos ou matrizes.
- Sintonizado com a forma de representar uma proposta de solução, os operadores de um algoritmo genético usados para gerar novas soluções devem ser projetadas de forma específica levando em conta as características do problema, isto é, esses operadores devem manipular de forma adequada as estruturas de dados complexas usadas para representar uma proposta de solução.
- A ideia central desse tipo de algoritmo genético modificado é que a representação de uma proposta de solução de um problema complexo seja o mais próximo possível do nível do fenótipo (e não do nível do genótipo).

Bortfeldt e Gehring (2001) são os primeiros pesquisadores que discutem abertamente o problema de uma adequada codificação ou de representação de uma proposta de solução para o PCC. Assim, os autores, Bortfeldt e Gehring (2001), apresentam uma forma de representação de uma proposta de solução usando a terminologia de conceito de solução (solution concept). Assim, o conceito de solução (a formulação de uma representação de uma proposta de solução ou codificação) para o PCC pode ser formulado da seguinte forma Bortfeldt e Gehring (2001):

- O algoritmo genético híbrido é usado para gerar um plano de carregamento (stowage plan) com uma estrutura tipo camada (layer). As camadas são formadas em sequência e sem superposição.
- A altura e a largura da camada são coincidentes com a altura e a largura do contêiner. A profundidade da camada é determinada pela profundidade da caixa que define a camada.
- O plano de carregamento, isto é, a formação das camadas são geradas usando um algoritmo heurístico chamado de heurística básica que funciona da seguinte forma: (1) No início do processo a heurística básica é usada para gerar soluções iniciais para conformar a população

inicial. Assim, a heurística básica gera uma solução usando o plano de carregamento e esse processo é repetido $npop$ vezes, isto é, o tamanho da população; (2) O operador de recombinação gera uma solução incompleta transferindo algumas camadas completas das soluções geradoras para a solução descendente em construção. Como nem sempre é possível gerar soluções válidas transferindo camadas completas das duas soluções geradoras, então após transferir um número determinado de camadas, deve-se terminar de gerar uma solução válida usando a heurística básica; (3) O operador de mutação também gera uma solução válida com a ajuda da heurística básica. Portanto, a heurística básica trabalha de três maneiras distintas e com algumas modificações em cada caso.

- A heurística básica e os operadores genéticos são projetados de forma que a sequência de camadas do plano de carregamento inicialmente não são interessantes. Por esse motivo, os planos de carregamento são representados internamente no processo através de uma estrutura de dados que não específica a sequência de camadas. A sequência de camadas no contêiner é determinada no passo final.

Portanto, o conceito de solução ou a representação de uma proposta de solução inclui as seguintes estruturas de dados e identificadores Bortfeldt e Gehring (2001):

1. **O contêiner** tem volume vc e dimensões xc de profundidade, yc de largura e zc de altura.

2. **As caixas** representadas por um conjunto de caixas e armazenadas no vetor $BList(b), \quad b = 1, \ldots, nb$, onde nb é o número total de caixas. Um elemento do vetor $BList$, isto é, uma caixa é caracterizada por suas 3 dimensões e seu volume. O vetor $BList$ é classificado em ordem decrescente de acordo com o volume da caixa. Também, o conjunto $BAll$ representa todas as caixas, isto é, $BAll = \{1, \ldots, nb\}$.

3. **As variantes de rotação** das caixas que são alocadas no contêiner devem ter índices de 1 a 6, identificadas de forma adequada. Deve-se observar que existem 6 formas possíveis de alocar uma caixa no contêiner e, portanto, cada forma de alocação deve ter um número identificador.

4. **A alocação de uma caixa** no contêiner deve ser identificado por sua estrutura pl que tem os seguintes componentes: (1) O índice identificador da caixa b, (2) as coordenadas em relação a origem, onde a caixa se encontra alocada, isto é, ox, oy e oz e, (3) o tipo de rotação rv da caixa. A origem das coordenadas é o canto inferior esquerdo do contêiner. As coordenadas y e z são absolutas, mas a coordenada x é relativa à camada.

5. **Uma camada** simples preenchida é representada pela estrutura l que incorpora os seguintes componentes:

   - O volume útil $vutil$ usado pela camada.
   - A profundidade da camada d.
   - O número de caixas npl alocadas na camada.
   - O vetor de alocação das caixas $Pl(i), \;\; i = 1, \ldots, npl$

   Um elemento do vetor $Pl(i)$ da camada l indica a alocação de uma caixa simples, a profundidade da camada é d que é a dimensão no eixo x da caixa que define a camada, $vutil$ é o quociente do volume das caixas alocadas e o volume da camada é igual a $(d.yc.zc)$. Adicionalmente, o operador $B(l)$ indica o conjunto de índices das caixas armazenadas no vetor $Pl(i)$.

6. **O plano de armazenamento** é representado por uma estrutura s e constituído pelos seguintes componentes:

   - O volume v das caixas armazenadas.
   - O número nl de camadas geradas.
   - O conjunto L de camadas geradas.
   - O profundidade $xcfree$ (ainda) disponível no contêiner.
   - O conjunto $Bfree$ de caixas (ainda) disponíveis e que ainda se encontram fora do contêiner.

As camadas de um plano de carregamento são representados por um conjunto de estruturas não ordenado. Os elementos deste conjunto são as camadas representadas na forma indicada no item 5 (a representação de uma camada). Se o plano de carregamento s ainda não foi terminado então a profundidade disponível $xcfree$ e o conjunto de caixas ainda não alocadas (disponíveis) $Bfree$ definem completamente a parcela restante do problema.

Deve-se observar que além da proposta de representação de uma poposta de solução, a parcela fundamental da proposta de Bortfeldt e Gehring (2001) é a heurística básica usada para preencher uma camada. Essa heurística básica é uma modificação da heurística apresentada por Gehring, Menschner e Meyer (1990) que é uma proposta heurística de geração de camadas para o preenchimento do contêiner. Essa heurística básica apresenta dois passos fundamentais na estratégia de geração de camadas:

- Uma camada é iniciada a partir da escolha da caixa base que abre a camada e sua orientação.
- No segundo passo são escolhidas as caixas e a ordem de alocação dessas caixas na camada em construção usando uma estratégia heurística chamada de preenchimento de camada (fill-layer).

Os detalhes sobre o preenchimento de uma camada usando a heurística básica está fora do escopo deste trabalho. Deve-se mencionar apenas que Bortfeldt e Gehring (2001) usam essa heurística básica de 3 formas diferentes: (1) para gerar uma solução que faz parte da população inicial e nesse caso, o plano de carregamento corrente é vazio ao iniciar a geração de uma solução, (2) para gerar uma solução após a recombinação onde apenas um subconjunto de camadas das soluções geradoras é preservado e, portanto, deve-se usar a heurística básica para terminar de gerar uma nova solução e, (3) após a mutação também é necessário o uso da heurística básica para terminar de gerar uma nova solução.

Adicionalmente, Bortfeldt e Gehring (2001) analisa a possibilidade de incorporar determinadas restrições na geração de soluções válidas. Assim, podem ser levadas em conta as seguintes restrições: (1) restrição de orientação onde nem todas as 6 possibilidades de rotação são permitidas, (2) restrição de estabilidade onde a superfície de contato de uma caixa, que não está alocada na superficie do contêiner, deve estar em contato pleno com a superfície da caixa alocada abaixo dela, (3) restrição de empilhamento em que determinadas caixas não podem ser alocadas acima de outras caixas, (4) restrição de peso, em que o peso das caixas alocadas no contêiner não pode ultrapassar a capacidade máxima de peso permitido para transporte no contêiner e (5) restrição de balanço da carga.

A partir de uma análise da proposta de Bortfeldt e Gehring (2001), devem-se realizar as seguintes observações:

- Os autores apresentam uma das primeiras propostas de codificação ou de representação de uma proposta de solução que é consistente com as características específicas do PCC. Basicamente, uma proposta de solução é armazenada em uma estrutura de dados um tanto complexa. Entretanto, deve-se mencionar que uma proposta de solução é formada por um conjunto de camadas e cada camada é formada por um conjunto de caixas que são preenchidas usando uma heurística básica. Essa proposta de codificação ou de representação de uma proposta de solução pode ser considerada uma evolução da proposta de codificação apresentada pelos autores em Gehring e Bortfeldt (1997) para o mesmo problema.
- O algoritmo genético híbrido não muda o preenchimento de uma camada. Assim, o operador de recombinação apenas preserva camadas completas das soluções geradoras e termina de gerar

uma nova solução gerando novas camadas com as caixas ainda disponíveis e usando a heurística básica. A mesma coisa acontece com o operador de mutação onde as camadas com melhor utilização de preenchimento são transferidas para o descendente e a partir dessas camadas é gerada uma nova solução gerando novas camadas com as caixas ainda disponíveis e usando a heurística básica.

- Baseado nas observações anteriores podemos afirmar que a proposta de codificação apresentada por Bortfeldt e Gehring (2001) é uma sequência de camadas e cada camada pode ser gerada usando as caixas ainda disponiveis e a heurística básica.

- A qualidade das soluções encontradas por Bortfeldt e Gehring (2001) são melhores que as encontradas pelo algoritmo genético apresentado pelos mesmos autores em Gehring e Bortfeldt (1997) e comparável com o algoritmo tabu search desenvolvido pelos mesmos autores em Bortfeldt e Gehring (2003).

- Os detalhes adicionais do algoritmo genético estão fora do escopo de análise neste trabalho.

Em geral, a proposta de Bortfeldt e Gehring (2001) pode ser considerado como uma proposta eficiente de otimização do PCC. Uma crítica que pode ser realizada a esse trabalho é a mesma que pode ser realizada à maioria de algoritmos que usam a lógica de geração de uma solução através de uma geração de camadas em sequência. Nesse tipo de lógica de otimização, a escolha da caixa que abre a camada é crítica e também a lógica de geração de camadas em sequência não permite gerar soluções que poderiam ser geradas usando a geração de arranjos (multicamadas). Em outras palavras, o espaço de busca é menor que o espaço de busca quando geramos uma solução usando arranjos (multicamadas).

3.2.4 *Algoritmo GRASP de Leite (2007)*

O trabalho de Leite (2007) propõe um algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*), voltada para o PCC composto por duas fases, que encontra boas soluções.

Inicialmente, pretende-se descrever as variáveis envolvidas no algoritmo. Leite (2007) define o conjunto $C = \{b_1, b_2, ..., b_m\}$, onde $b_1, b_2, ..., b_m$ são as caixas, representadas por suas características: *comprimento, largura e altura*, que irão compor o preenchimento do contêiner, que na

fase de construção da solução será carregado do seu fundo para sua entrada, tentando sempre manter uma superfície plana.

A seguir é apresentado o procedimento heurístico referenciado em Leite (2007) para a resolução do PCC, através da metaheurística GRASP.

Na fase de construção do algoritmo GRASP de Leite (2007), constrói-se uma solução iterativamente elemento por elemento. A cada iteração, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordem pré-determinado. O processo de seleção é baseado em uma função adaptativa gulosa g: C $\mapsto$ R, que estima o parâmetro de seleção aleatoriamente, a partir de um subconjunto restrito formado pelos melhores elementos da lista de candidatos, que recebe o nome de *lista restrita de candidatos* (LRC). Esta técnica permite que diferentes soluções sejam geradas a cada iteração GRASP.

Um padrão de preenchimento será obtido tomando-se *n* elementos do conjunto *C*, sendo $n \leq m$, construindo um subconjunto restrito $LRC = \{b_1, b_2, ..., b_n\}$ formado pelas caixas de maior volume que compõem a lista de candidatos e a partir dos itens deste conjunto, procura-se estabelecer uma sequência de preenchimento do contêiner, camada por camada. Uma camada é uma seção do comprimento do contêiner na altura *H* e largura *L* completa, que irá preencher o contêiner, construindo camadas ao longo do seu comprimento *C* e combinando espaços vazios entre camadas para aumentar a utilização de espaço.

O tamanho da LRC é controlado por um parâmetro $\alpha \in [0, 1]$, onde para $\alpha = 1$ tem-se um comportamento guloso do algoritmo e para $\alpha = 0$, um comportamento aleatório. Assim, o parâmetro α regula o grau de miopia e aleatoriedade na fase de construção. Esse é o principal parâmetro a ser ajustado no algoritmo GRASP, pois se a cardinalidade de LRC for pequena, menor será o espaço de solução examinado e, consequentemente, a probabilidade de escapar de um ótimo local diminuirá.

A fase de busca local está baseada na noção de vizinhança. A função *N*, a qual depende da estrutura do problema tratado, relaciona a cada solução viável s sua vizinhança $N(s)$. Cada solução s' $\in N(s)$ é chamada de vizinho de s. Em linhas gerais, esta fase, começa de uma solução obtida pela fase de construção GRASP e navega pelo espaço de pesquisa passando de uma solução para outra, que seja sua vizinha, em busca de uma melhor solução.

Para definição do problema de preenchimento de contêiner, considera-se um conjunto de m itens (caixas retangulares). Para cada caixa *i*, caracterizada pelo comprimento c_i, largura l_i e altura h_i, tem-se uma quantidade q_i de caixas, para todo $i \in I = 1, 2, 3, ..., m$. Será considerado também um contêiner retangular, tendo como dimensões internas: comprimento C, largura L e altura H.

A lista de candidatos C é formada pelas caixas em ordem decrescente, de acordo com o seu valor de volume calculado. A escolha da caixa a ser utilizada é feita de forma aleatória em uma das duas listas construídas, LRC ou LRC1, onde LRC contém as caixas de maior volume da lista de candidatos, determinadas pela faixa selecionada de acordo com o parâmetro α que define a cardinalidade da lista, e LRC1 contém as caixas restantes, isto é, as caixas menores que não foram selecionadas para a lista LRC.

Realizado o preenchimento da camada atual, é feita a atualização das quantidades das caixas utilizadas e das listas restritas.

Em cada camada preenchida, passa-se pela fase de melhoria, verificando quais colunas construídas poderão passar por duas etapas: melhoria da altura e/ou do comprimento. Nestas etapas, verifica-se o espaço ocioso e se faz uma busca mais detalhada pela caixa que melhor preenche este espaço.

Na fase de melhoria da altura, verifica-se o espaço ocioso acima das caixas do mesmo tipo e cria-se o espaço vazio a ser preenchido. É feito este processo sucessivamente a cada tipo de caixa diferente escolhido na camada.

Na fase de melhoria do comprimento, verifica-se o espaço ocioso a frente das caixas de mesmo tipo cuja dimensão de comprimento seja menor que a dimensão de comprimento da camada e cria-se o espaço livre a ser preenchido.

O algoritmo GRASP de Leite (2007) tem como critérios de parada:

- Se a quantidade total de caixas é zerada, ou seja, se não houver mais caixas a serem carregadas.
- Se o número de iterações chegar ao máximo.

Algoritmo GRASP de Leite (2007).

- Entra com os dados
- Calcula o volume das caixas.
- Define o número de iterações.
- Calcula o número total de caixas não carregadas.

- Cria a lista C posicionadas em ordem decrescente, de acordo com o volume de cada tipo de caixa. Lista = 1.

Fase de Construção:

- Critério de parada: enquanto tiver caixas não carregadas e espaço a ser preenchido.
- Identifica a caixa não carregada com maior volume.
- Cria as listas de candidatos LRC e LRC1.

Camada Principal:

Lista = 1, significa carregamento de uma nova camada, ou seja, carregamento do comprimento do contêiner.

- Verifica quais os tipos de caixa que cabem no espaço a ser preenchido.
- Escolher uma caixa aleatoriamente de LRC. Se em LRC não houver nenhuma caixa, escolher uma caixa aleatoriamente de LRC1.
- Verifica se a quantidade de caixas é suficiente para completar uma coluna.
  - Se for suficiente, empacotar a quantidade de colunas completas que forem possíveis.
  - Senão empacotar a quantidade de caixas disponíveis.
- Atualiza a quantidade de cada tipo de caixa disponível.

Espaço Residual de Largura do Contêiner:

- Verifica se tem espaço disponível a ser preenchido e se cabe algum tipo de caixa.
- Verifica na lista LRC, se existe alguma caixa que preencha o espaço ocioso.
  - Se existir, verificar se a quantidade é suficiente para completar uma coluna completa. Senão carrega a quantidade de caixas disponíveis.
  - Carrega a quantidade de colunas completas que forem possíveis de serem alocadas.
- Senão passa para a lista LRC1 e verifica se existe alguma caixa que preencha o espaço ocioso.

  - Verifica qual tipo de caixa melhor preenche o comprimento do espaço ocioso.
  - Se existir, verificar se a quantidade é suficiente para completar uma coluna completa. Senão, carrega a quantidade de caixas disponíveis.
  - Carrega a quantidade de colunas completas que forem possíveis alocar.

- Atualiza a quantidade de cada tipo de caixa disponível.

Espaço Residual de Altura do Contêiner:

- Criar o volume ocioso acima das caixas das colunas preenchidas.

$$\text{volVazio} = \text{restAltura} * \text{largCol} * \text{compCol}$$

- Verificar, nas listas restritas, se existe alguma caixa cujo volume ocupe o volume ocioso acima das caixas e se o restante da altura é menor ou igual a menor dimensão das caixas.
- Se existir escolher um tipo de caixa que ocupe o máximo do comprimento do espaço residual a ser preenchido.
  - Carregar quantas colunas completas forem possíveis.
  - Atualizar a quantidade de caixas disponíveis.
- Senão, voltar para a Camada Principal e criar uma nova camada.

Resultados Computacionais.

Os resultados obtidos com o algoritmo GRASP de Leite (2007), foram testados com os dados reais de George e Robinson (1980) e demonstram ser satisfatórios, observando que não existe menção explícita de que exista espaço residual de comprimento do contêiner onde seja possível alocar mais caixas e também não é detalhado como é feito o carregamento de cada camada.

Para melhor avaliação do algoritmo GRASP Leite (2007) gera outros 9 dados aleatoriamente para testes, cujas dimensões e quantidades admitem valores também aleatorios. Dentro de todos os dados gerados, tem-se que DA1 a DA4 mantém o padrão dos dados de George e Robinson (1980) utilizando 8 tipos de caixas. Já os dados de DA5 a DA8 são gerados 12 tipos de caixas e por fim os dados de DA9 são gerados 20 tipos de caixas.

Quanto aos exemplos testados por Leite (2007), os resultados apresentados foram abaixo de 90% do volume de caixas carregadas em relação ao volume do contêiner. Neste trabalho apresentamos melhores resultados que os de Leite (2007) e de George e Robinson (1980).

Capítulo 4

O Algoritmo Heurístico Construtivo Aplicado ao Problema de Carregamento de Contêiner

4.1 Introdução

Neste Capítulo apresentamos um algoritmo heurístico construtivo (AHC) para o PCC. A principal diferença entre o AHC descrito no Capítulo 2 e os outros AHCs apresentados na literatura especializada, como o algoritmo de George e Robinson (1980), é que no AHC apresentado neste trabalho desenvolvido o conceito de arranjo em contraposição de camada usado na, maioria.

Uma camada é formada usando apenas uma caixa como base. Para formar uma camada devemos escolher uma caixa que representa a base na formação da camada. Assim, uma vez escolhida uma caixa de base para iniciar uma camada, então a profundidade da caixa define a profundidade da camada. Portanto, todas as caixas que devem fazer parte dessa camada devem ter uma profundidade igual ou menor que a profundidade da caixa usada como base.

Após escolher uma caixa da base para iniciar o preenchimento de uma nova camada, então a camada pode ser preenchida adicionando novas caixas. Uma possibilidade é iniciar o preenchimento das caixas seguindo a largura do contêiner e depois preenchendo a altura. Outra possibilidade é preencher primeiro as caixas na altura. Nesse contexto aparece o conceito de torre. Assim, uma torre é formada por um conjunto de caixas, uma acima da outra, até preencher todo o espaço de altura possível do contêiner. Na formação de uma torre de caixas existem duas propostas em relação às restrições no processo de alocação. Uma proposta, mais tradicional, usada por exemplo no trabalho de George e Robinson (1980), exige que a superfície de contato de uma caixa deve ser menor ou igual da caixa que se encontra debaixo dela e, portanto, garantindo uma superfície de contato plena e estável. Outra

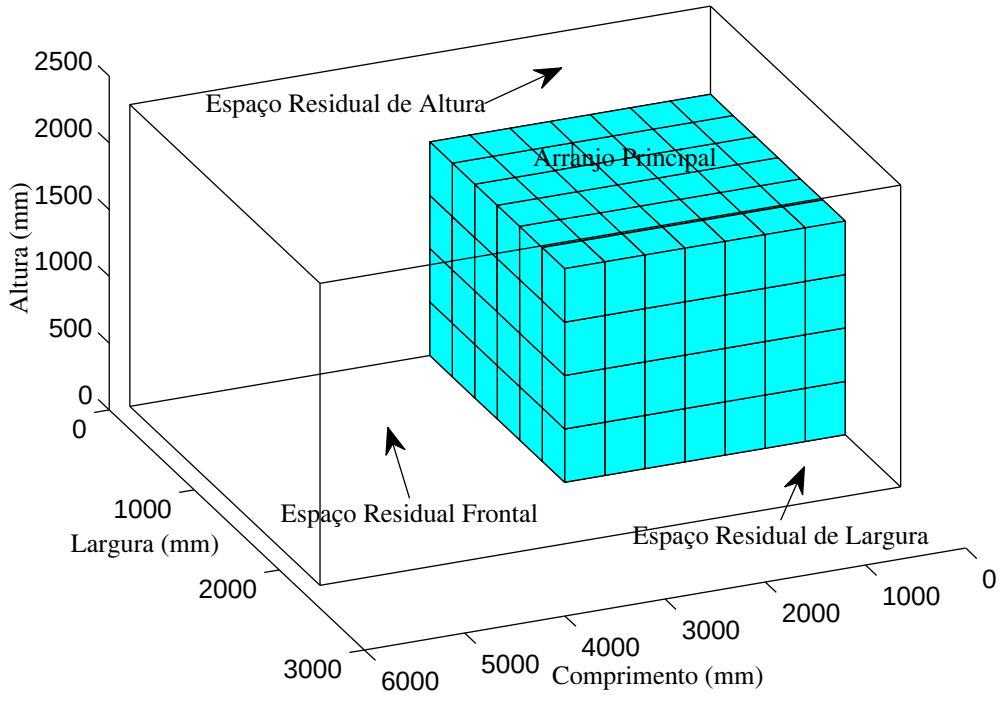
proposta, apresentada por pesquisadores como Pisinger (2002), não apresenta a exigência anterior e, portanto, a superfície de contato não necessariamente apresenta contato pleno. Nesse caso, como sugere Pisinger (2002), a estabilidade pode ser garantida alocando material adequado no espaço livre. Essa restrição de superfície de contato está presente não somente na formação de torres, mas na alocação das caixas de uma camada. Neste trabalho usamos a restrição imposta por trabalhos como o de George e Robinson (1980), mas também podemos usar a proposta de Pisinger (2002). Portanto, em cada teste devemos deixar claramente especificado o tipo de restrição usado.

Em contraposição ao conceito de camada, neste trabalho apresentamos o conceito de arranjo. Um arranjo pode ser constituído por uma ou várias camadas. Assim, se o arranjo possui apenas uma camada, as duas estratégias de preenchimento de caixas seriam iguais. A ideia fundamental de usar a proposta de arranjo é que pode ser gerado um arranjo com caixas do mesmo tipo, de forma que o arranjo formado por um conjunto de caixas, forme uma estrutura retangular perfeita. Assim, esse arranjo perfeito formado por um conjunto de caixas iguais, apresenta perda nula de volume no espaço usado do contêiner. Adicionalmente, essa estrutura retangular perfeita permite que o espaço residual de altura e o espaço residual de largura sejam preenchidos por caixas com perda mínima de volume. Obviamente, esse tipo de AHC se torna especialmente eficiente se existem poucos tipos de caixas com muitas caixas de cada tipo.

Para ilustrar a proposta usamos a Figura 15 onde aparece uma proposta de arranjo principal formado por várias camadas de caixas do mesmo tipo. Assim, usando a Figura 15 definimos os seguintes conceitos:

- **Arranjo principal:** Um arranjo principal é formado por uma ou várias camadas de caixas que formam uma estrutura retangular perfeita. Um arranjo principal sempre é formado a partir do canto inferior esquerdo do contêiner e junto ao arranjo principal montado no passo imediatamente anterior. O arranjo principal pode ser formado por caixas do mesmo tipo ou caixas de vários tipos, mas com a exigência de que o arranjo forme uma estrutura retangular perfeita, isto é, um arranjo principal pode ser considerado como uma caixa grande de dimensões conhecidas. A altura do arranjo principal deve ser menor que a altura do contêiner e o espaço livre exatamente acima do arranjo principal é chamado de espaço residual de altura. A largura do arranjo principal deve ser menor ou igual que a largura do contêiner e o espaço livre que fica ao lado direito do arranjo principal e com altura igual à altura do contêiner é chamado de espaço residual de largura. Finalmente, o espaço ainda não usado no contêiner e onde podem ser montados novos arranjos principais é chamado de espaço residual frontal. Em resumo, o arranjo principal

Figura 15: Arranjo Principal.



Fonte: Informações da pesquisa da autora.

pode ser considerado como sendo uma caixa retangular de dimensões $D_p \times W_p \times H_p$, isto é, D_p de profundidade, W_p de largura e H_p de altura para um contêiner com dimensões $D \times W \times H$.

- **Espaço residual de altura:** O espaço residual de altura é o espaço disponível no contêiner exatamente acima de um arranjo principal. Assim, relacionado com o arranjo principal especificado anteriormente, o espaço residual de altura é um espaço retangular com dimensões $D_p \times W_p \times (H - H_p)$. Deve-se observar que o espaço residual de altura pode ser considerado como um minicontêiner de dimensões $D_p \times W_p \times (H - H_p)$ e, portanto, nesse espaço podem ser alocadas caixas usando estratégias eficientes ou até mesmo usando um algoritmo de otimização exato já que se trata de um problema muito mais simples que o problema do contêiner completo.

- **Espaço residual de largura:** O espaço residual de largura é o espaço disponível no contêiner exatamente ao lado direito de um arranjo principal e com uma altura igual à altura do contêiner. Assim, relacionado com o arranjo principal especificado anteriormente, o espaço residual de

largura é um espaço retangular com dimensões $D_p \times (W - W_p) \times H$. Deve-se observar que o espaço residual de largura pode ser considerado também como um minicontêiner de dimensões $D_p \times (W - W_p) \times H$ e, portanto, nesse espaço podem ser alocadas caixas usando estratégias eficientes ou até mesmo usando um algoritmo de otimização exato já que também se trata de um problema muito mais simples que o problema do contêiner completo.

- **Espaço residual frontal:** É o espaço ainda disponível no contêiner e onde podem ser montados novos arranjos principais. Assim, seja D_c o espaço corrente e já usado por arranjos principais no preenchimento de caixas no contêiner, então o espaço residual frontal é um espaço retangular livre com dimensões $(D - D_c) \times W \times H$.

4.2 Algoritmo Heurístico Construtivo Aplicado ao Problema de Carregamento de Contêiner

O AHC é descrito em vários módulos para facilitar a apresentação. Inicialmente, é mostrado o algoritmo global e depois a forma de preenchimento dos tipos de arranjos existentes no processo de construção.

O algoritmo heurístico principal assume a seguintea passos:

1. Identificar os tipos de caixas ainda disponíveis, o número de caixas de cada tipo, o volume total disponível correspondente a cada tipo de caixa e o volume total disponível correspondente a todas as caixas.

2. Se o volume total correspondente a todas as caixas pode se alocado em uma única camada então ir ao passo 5 e usar a estratégia de preenchimento da última camada. Em caso contrário, montar o arranjo principal usando o tipo de caixa que apresenta o maior volume total ainda disponível e ir ao passo 3.

3. Montar o espaço residual de altura.

4. Montar o espaço residual de largura e voltar ao passo 1.

5. Montar a última camada usando a estratégia de preenchimento da última camada.

A seguir detalhamos a estratégia para cada passo do algoritmo principal.

No passo 2 existe a necessidade de identificar, se as caixas restantes podem ser alocadas em uma única camada e consideremos que o volume total de todas as caixas ainda disponíveis é igual a V_{res} (volume do espaço residual). Para realizar essa verificação, comparamos a menor dimensão de todas as caixas ainda disponíveis e identificamos o maior valor entre elas que denominamos D_r. Seja o volume $V_r = D_r \times W \times H$. Assim, se o volume V_r for maior que o volume V_{res}, então as caixas restantes podem ser alocadas em uma única camada.

A montagem do arranjo principal do passo 2, deve levar em conta os seguintes critérios:

- A idéia central é montar um arranjo principal formado por caixas do mesmo tipo formando uma estrutura retangular perfeita de tamanho máximo. Entretanto, deve-se verificar se existem caixas para preencher o volume do espaço residual de altura e o volume do espaço residual de largura.

- Após montar o candidato a arranjo principal, deve-se verificar se existem caixas com dimensões adequadas para preencher o volume residual de altura e o volume residual de largura.

- No caso em que existam caixas que permita montar um arranjo principal com várias camadas, deve-se verificar a possibilidade de preencher as caixas maximizando a altura e a largura do arranjo principal e também se deve analisar a possibilidade de deixar uma caixa a menos na altura e na largura. Assim, para cada tipo de caixa a ser analisado e após escolher a profundidade existem quatro possibilidades de formar o arranjo principal: (1) usando as caixas de forma a usar a altura máxima e a largura máxima, (2) usando as caixas de forma a usar a altura máxima e na largura deixar uma caixa a menos, (3) usando as caixas de forma a usar a largura máxima e na altura deixar uma caixa a menos, e (4) usando as caixas de forma a deixar uma caixa a menos na altura e uma caixa a menos na largura. A opção escolhida é aquela que permite usar da melhor forma possível os espaços residuais de altura e largura.

- No caso, em que existam caixas que não permitem montar uma camada completa do arranjo principal, então deve-se analisar duas possibilidades: (1) montar uma camada priorizando diminuir o espaço residual de altura e (2) montar duas camadas tentando aumentar a superfície de contato do arranjo principal.

- Caso exista um arranjo, em que o espaço residual seja maior que 5% e não seja possível alocar nenhuma caixa, então existe a evidência de que o arranjo montado é ineficiente. Essa

observação é válida na montagem de caixas no arranjo principal, no arranjo residual de altura e no arranjo residual lateral.

4.2.1 Montagem do Arranjo Principal

A montagem do arranjo principal é fundamental para o desempenho eficiente do AHC. Assim, para a montagem do melhor arranjo principal devemos montar uma estratégia eficiente e analisando várias alternativas. Também devemos lembrar que o arranjo principal é formado por um conjunto de caixas de forma que todas essas caixas, formam uma estrutura retangular perfeita. A montagem do arranjo principal é realizado usando o seguinte esquema:

1. O tipo de caixa escolhido é aquele que tem o maior volume total disponível. Em caso de empate se escolhe o tipo de caixa com maior volume individual.

2. Caso exista um número de caixas suficientes para preencher uma camada completa então montar o arranjo principal usando a estratégia 1. Em caso contrário usar a estratégia 2.

Estratégia 1: Esta estratégia é usada quando existem caixas suficientes para preencher pelo menos uma camada completa. Nesta estratégia usamos 24 alternativas diferentes de preechimento e escolhemos quatro delas como candidatas a arranjo principal. Posteriormente, deve-se realizar o preenchimento do arranjo residual de altura e do arranjo residual de largura de cada uma dessas quatro propostas candidatas. Finalmente, o arranjo completo deve ser o melhor dessas quatro propostas.

Em resumo, uma vez escolhido o tipo de caixa para montar o arranjo principal, devemos gerar quatro propostas de solução e escolher a melhor delas.

As 24 propostas de arranjo principal são geradas escolhendo cada uma das dimensões da caixa na profundidade (três dimensões) e para cada opção de profundidade existe a opção de escolha das outras duas dimensões da caixa na altura e na largura (2 dimensões). Assim, se usamos as caixas de forma a preencher o máximo número de caixas na altura e na largura, então existem 6 propostas de arranjo principal para um tipo de caixa selecionado. Entretanto, para cada tipo de profundidade escolhida foi proposta a montagem de quatro arranjos principais: (a) usando o preenchimento máximo na altura e na largura, (b) usando o preenchimento máximo na altura, mas deixando uma caixa a menos na largura, (c) usando o preenchimento máximo na largura e deixando uma caixa a menos na altura e,

(d) deixando uma caixa a menos na altura e na largura. Assim, existem 24 combinações possíveis para gerar um arranjo principal. Para escolher as quatro candidatos a arranjo principal escolhemos a melhor de cada tipo de proposta, isto é, dos tipos (a), (b), (c) e (d).

Para identificar a proposta de arranjo principal mais promissora usamos uma proposta heurística. Para cada proposta de arranjo principal encontramos, o volume do arranjo residual de altura (V_{ra}), o volume do arranjo residual da largura (V_{rl}) e o volume de todas as caixas que podem ser alocadas no arranjo residual de altura e no arranjo residual lateral (V_{rc}). Assim, a melhor proposta de arranjo principal (uma de cada tipo (a), (b), (c) e (d)) é aquela que apresenta o maior valor de I_r:

$$I_r = \frac{V_{rc}}{V_{ra} + V_{rl}}$$

Para ilustrar a implementação da Estratégia 1 usamos o exemplo DA2 mostrado na Tabela 16 e no Apêndice A.

Tabela 16: Conjunto DA2 - Dados aleatórios com 453 caixas

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)		m^3	m^3
1	820	282	366	92	0,0846	7,7863
2	1332	179	209	19	0,0498	0,9468
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	140	0,0542	7,5881
7	428	280	386	89	0,0463	4,1170
8	356	347	100	27	0,0124	0,3335

Fonte: Informações da pesquisa da autora.

De acordo com a Estratégia 1 e os dados mostrados na Tabela 16 podemos concluir que o primeiro arranjo principal deve ser montado usando as caixas de tipo 1 que tem volume total corrente de $7,7863m^3$. A Tabela 17 mostra os 24 candidatos a arranjo principal. Verificando as 24 propostas a estratégia 1 deve escolher as 4 propostas seguintes:

1. Forma de preenchimento tipo (a): Proposta No. 13 com arranjo formado por cinco camadas e 80 caixas alocadas.

2. Forma de preenchimento tipo (b): Proposta No. 6 com arranjo formado por duas camadas e 80 caixas alocadas.

3. Forma de preenchimento tipo (c): Proposta No. 3 com arranjo formado por duas camadas e 70 caixas alocadas.

4. Forma de preenchimento tipo (d): Proposta No. 8 com arranjo formado por duas camadas e 70 caixas alocadas.

Tabela 17: Alternativas de arranjo principal

N	Prof.	Larg.	Alt.	Forma	Caixas no	v_{ra}	v_{rl}	V_{rc}	I_r
	(mm)	(mm)	(mm)		Arranjo	m^3	m^3	m^3	
1	820	282	366	(a)	84	0,2104	0,9715	2,677	2,265
2	820	282	366	(b)	72	0,1804	2,0172	20,499	9,328
3	820	282	366	(c)	70	1,3953	0,9715	20,668	8,732
4	820	282	366	(d)	90	1,1940	2,0172	18,9758	3,937
5	820	366	282	(a)	48	0,0009	0,7416	——	——
6	820	366	282	(b)	80	0,0150	1,5055	25,5774	16,8220
7	820	366	282	(c)	84	1,0336	0,1483	10,1328	8,5731
8	820	366	282	(d)	70	0,8613	1,5055	24,7311	10,4491
9	366	282	820	(a)	84	2,6920	1,3009	25,9159	6,4906
10	366	282	820	(b)	84	2,6920	3,1512	25,9159	4,4352
11	366	282	820	(c)	91	13,5343	2,8186	26,5084	1,6210
12	366	282	820	(d)	90	13,3856	6,7526	26,4237	1,3121
13	366	820	282	(a)	80	0,0150	2,4660	25,5774	10,3092
14	366	820	282	(b)	88	0,0165	12,8896	26,2545	2,0343
15	366	820	282	(c)	84	1,0336	2,9592	25,9159	6,4906
16	366	820	282	(d)	91	1,1198	15,2331	26,5084	1,6210
17	282	820	366	(a)	84	0,2104	2,6601	25,9159	9,0284
18	282	820	366	(b)	90	0,2256	13,5427	26,4237	1,9192
19	282	820	366	(c)	90	1,7940	3,4201	26,4237	5,0678
20	282	820	366	(d)	90	1,7940	16,2512	26,4237	1,4643
21	282	366	820	(a)	84	2,6920	0,1785	25,9159	9,0284
22	282	366	820	(b)	90	2,8843	2,3298	26,4237	5,0678
23	282	366	820	(c)	90	13,3856	0,3826	26,4237	1,9192
24	282	366	820	(d)	90	13,3856	4,6596	26,4237	1,4643

Fonte: Informações da pesquisa da autora.

Estratégia 2: Esta estratégia é usada quando não existem caixas suficientes para montar um arranjo completo. Neste caso a proposta fundamental consiste em formar um arranjo principal em que as dimensões da largura e da altura do arranjo principal sejam os mais próximos possíveis. Adicionalmente, neste caso devem ser encontrados 12 candidatas de arranjo principal. Esses 12 candidatos a arranjo principal são os seguintes:

1. Seis candidatos a arranjo principal usando uma estratégia semelhante com a estratégia 1 e apenas uma camada, isto é, escolhendo cada uma das dimensões da caixa na profundidade (três dimensões) e para cada opção de profundidade existe a opção de escolha das outras duas dimensões da caixa na altura e na largura (duas dimensões).

2. Seis candidatos a arranjo principal usando a mesma lógica do passo anterior, mas usando duas camadas.

Dos 12 candidatos a arranjo principal, o algoritmo escolhe 3 deles para completar o processo, com o preenchimento do espaço residual de altura e do espaço residual de largura. Esses 3 candidatos são escolhidos usando a seguinte heurística:

- O arranjo principal que apresenta as dimensões mais próximas às dimensões de um cubo, isto é, o arranjo que tem as três dimensões mais próximas possíveis.
- Dentre os arranjos que usam apenas uma camada escolher aquela em que as dimensões de altura e largura são as mais próximas possíveis.
- Dentre os arranjos que usam duas camadas escolher aquela em que as dimensões de altura e largura são as mais próximas possíveis.

Para determinar a proximidade entre as dimensões de um arranjo principal escolhemos como base a dimensão de menor valor e usando esse valor, calculamos o desvio porcentual das outras dimensões. Assim, o arranjo mais adequado é aquele que apresenta o menor desvio porcentual.

4.2.2 *Montagem no Espaço Residual de Altura*

Para o preenchimento do espaço residual de altura devemos levar em conta os seguintes critérios:

- O espaço residual de altura é na verdade um minicontêiner de dimensões reduzidas.
- É um espaço, em que geralmente o volume disponível é pequeno e onde podem ser alocadas apenas um número reduzido de tipos de caixas. Assim, em um caso extremo, este espaço residual pode ser preenchido usando enumeração completa. Essa afirmação é particularmente válida nos tipos de problemas, em que existem poucos tipos de caixas e muitas caixas do mesmo tipo.
- O preenchimento de caixas deve ser na seguinte sequência: inicialmente, deve-se preencher a profundidade, depois a altura e finalmente a largura do contêiner.
- A proposta anterior, mostra que após preencher as caixas que entram na profundidade, então existe um espaço residual de altura, um espaço residual frontal e um espaço residual de largura.

- Neste trabalho apresentamos a proposta de preenchimento, em que o espaço residual frontal deve ser minimizado e, portanto, não deve ser usado.

- Em resumo, para preencher o espaço residual de altura, após abrir uma camada na profundidade, existe disponivel um espaço residual frontal (que sempre deve ser minimizado e, portanto, nunca usado), um espaço residual de altura e um espaço residual de largura. Portanto, a camada em profundidade deve ser completada preenchendo as caixas no espaço residual de altura. No preenchimento desse espaço residual de altura deve ser seguida a ideia de formação de torre, isto é, acima de uma caixa deve ser alocada outra caixa com superfície de contato menor que a caixa que serve de base. Entretanto, deve-se observar que, se as caixas iguais estão em sequência formando a camada, então a superfície de contato deve ser a soma das superfícies de contato de todas as caixas com a mesma altura. O espaço residual de largura é tratado como se fosse um novo espaço residual de altura.

1. Identificar os tipos de caixas $k = 1, \ldots, K$ que podem ser alocadas no espaço residual de altura.

2. Seja k o tipo de caixa escolhido para iniciar um arranjo no espaço residual de altura. Formar 3 arranjos candidatas usando cada dimensão da caixa na largura e que define a largura da camada.

3. Para cada dimensão escolhida na largura, montar o arranjo na profundidade e posteriormente completar o preenchimento na altura. As caixas escolhidas para formar o arranjo em profundidade devem ter largura menor ou igual que a largura da camada.

4. Na formação da camada em profundidade devem ser escolhidas as dimensões de forma a minimizar o espaço residual de largura da camada.

5. Na alocação das caixas no espaço residual de altura da camada na profundidade, devem ser alocadas caixas em que a superficie de contato da caixa, a ser alocada deve ser menor ou igual que a superficie de contato da caixa alocada abaixo dela.

6. Repetir os passos 1 a 5 para preencher uma nova camada em profundidade. O processo termina quando for verificado que no espaço residual lateral não é possível alocar as caixas disponíveis ou, quando não existem mais caixas com dimensões adequadas para serem alocadas.

O processo mostrado anteriormente deve ser realizado começando com cada tipo de caixa disponível k e, portanto, podem ser geradas K propostas de solução. A arranjo final do espaço residual de altura deve ser o melhor das K propostas de solução encontradas.

4.2.3 *Montagem do Espaço Residual de Largura*

Para o preenchimento do espaço residual de largura devemos usar a mesma estratégia usada no preenchimento do espaço residual de altura.

4.2.4 *Montagem da Última Camada*

Para o preenchimento da última camada devemos usar a mesma estratégia usada para o preenchimento do espaço residual de altura, mas com as seguintes particularidades.

1. A sequência de preenchimento deve ser iniciada com o preenchimento da altura e depois na largura.

2. A profundidade do arranjo deve ser montada com uma caixa ou simultaneamente com duas caixas.

3. Devem ser geradas todas as alternativas de solução, iniciando a profundidade da camada com uma ou duas caixas (caixas do mesmo tipo ou de dois tipos diferentes). Finalmente, deve ser escolhida a melhor proposta de solução.

4.2.5 *Exemplo Ilustrativo*

Para ilustrar o AHC usamos o exemplo DA2 cujos dados são mostrados no apêndice e na Tabela 16.

Na montagem das caixas indicamos as dimensões usando a ordem de profundidade, largura e altura. Assim, por exemplo, se uma caixa tipo 1 foi alocada na forma $820 \times 366 \times 282$ significa que a dimensão 820 mm está na profundidade, a dimensão 366 na largura e a dimensão 282 está na altura do contêiner.

Nas Figuras 16 a 24 e nas Tabelas 18 a 26 apresentamos o carregamento completo de DA2, passo a passo, com a heurística proposta neste trabalho, após o carregamento de todas as 453 caixas e mostrando o espaço residual de $83mm$.

Tabela 16: Conjunto DA2 - Dados aleatórios com 453 caixas

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)		m^3	m^3
1	820	282	366	92	0,0846	7,7863
2	1332	179	209	19	0,0498	0,9468
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	140	0,0542	7,5881
7	428	280	386	89	0,0463	4,1170
8	356	347	100	27	0,0124	0,3335

Fonte: Informações da pesquisa da autora.

1. **Primeiro arranjo principal:**

   Escolhemos as caixas do tipo 1 que têm volume total corrente de $7,7863m^3$.

   O melhor arranjo principal seria na forma $820 \times 366 \times 282$. Assim, o arranjo principal assume a seguinte distribuição: $1 \times 6 \times 7$, isto é, o arranjo principal está formado por 42 caixas do tipo 1 e uma camada.

   - Espaço residual de altura: $820 \times 2196 \times 287$.

     Neste caso pode ser usado esse espaço residual para carregar 6 caixas do tipo 1 na seguinte estrutura: $820 \times 366 \times 282$, isto é, $1 \times 6 \times 1 = 6$ caixas do tipo 1.

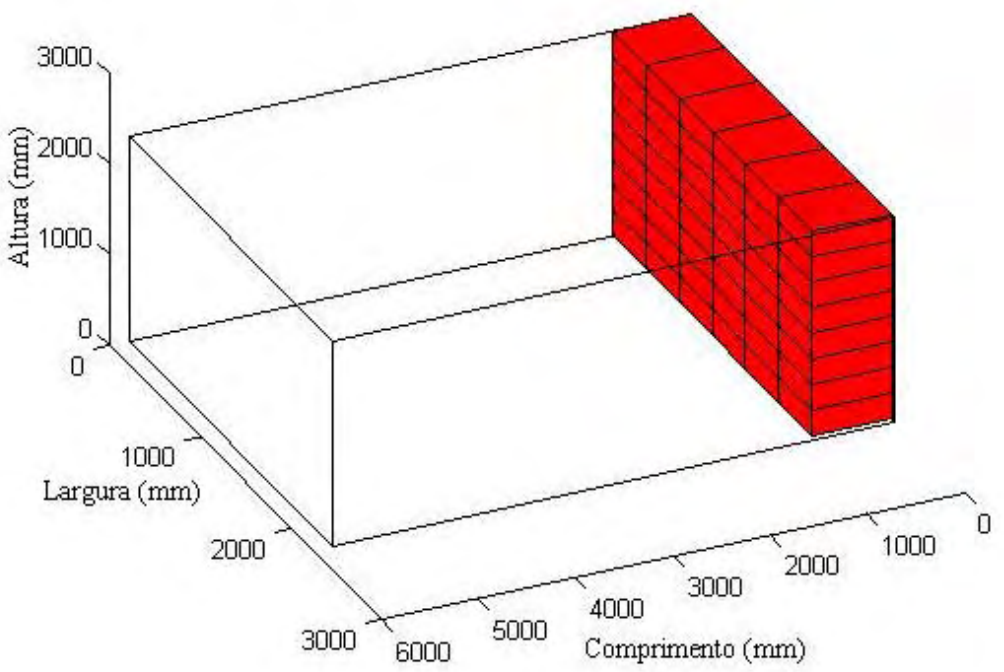
   - Espaço residual de largura: $820 \times 4 \times 2261$.

     Neste caso pode não ser usado esse espaço residual Não existe possibilidade de alocar caixas no espaço residual.

   Resumo do primeiro arranjo principal:

   - 48 caixas do tipo 1 alocadas.
   - Volume ocupado: 48(0,084634) = 4,0608 m^3.
   - Volume usado: $0,820 \times 2,236 \times 2,261 = 4,145589\ m^3$.
   - Taxa de ocupação: 97,9547 %.
   - Profundidade usada: 820 mm.

Figura 16: Conjunto DA2: Carregamento do primeiro arranjo.



Fonte: Informações da pesquisa da autora.

Na Figura 16 apresentamos o carregamento do primeiro arranjo, com 48 caixas.

Na Tabela 18 apresentamos os tipos e quantidades de caixas que não foram carregadas.

Tabela 18: Conjunto DA2: Caixas restantes após o primeiro arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	44	0,0846	3,7239
2	1332	179	209	19	0,0498	0,9468
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	140	0,0542	7,5881
7	428	280	386	89	0,0463	4,1170
8	356	347	100	27	0,0124	0,3335

Fonte: Informações da pesquisa da autora.

2. **Segundo arranjo principal:**

   Escolhemos as caixas do tipo 6 que têm volume total corrente de $7,5881m^3$.

   O melhor arranjo principal seria na forma $495 \times 339 \times 323$. Assim, o arranjo principal assume a seguinte distribuição: $3 \times 6 \times 6 = 108$, isto é, o arranjo principal é formado por 108 caixas do tipo 6 e três camadas.

   - Espaço residual de altura: $1485 \times 2034 \times 323$.

     Neste caso pode ser usado esse espaço residual para carregar 18 caixas do tipo 6 com a seguinte estrutura. $495 \times 339 \times 323$, isto é, $3 \times 6 \times 1 = 18$ caixas do tipo 6.

   - Espaço residual de largura: $1485 \times 202 \times 2261$.

     Neste caso pode ser usado esse espaço residual para alocar 7 caixas do tipo 2 com a seguinte estrutura: $209 \times 179 \times 1332$, isto é, $7 \times 1 \times 1 = 7$ caixas do tipo 2.

     Existe ainda um espaço residual de altura no espaço residual lateral.

     O espaço residual de altura apresenta as dimensões: $1463 \times 179 \times 929$. Nesse espaço podem ser carregados 8 caixas do tipo 8 na seguinte estrutura: $356 \times 100 \times 347$, isto é, $4 \times 1 \times 2 = 8$ caixas do tipo 8.

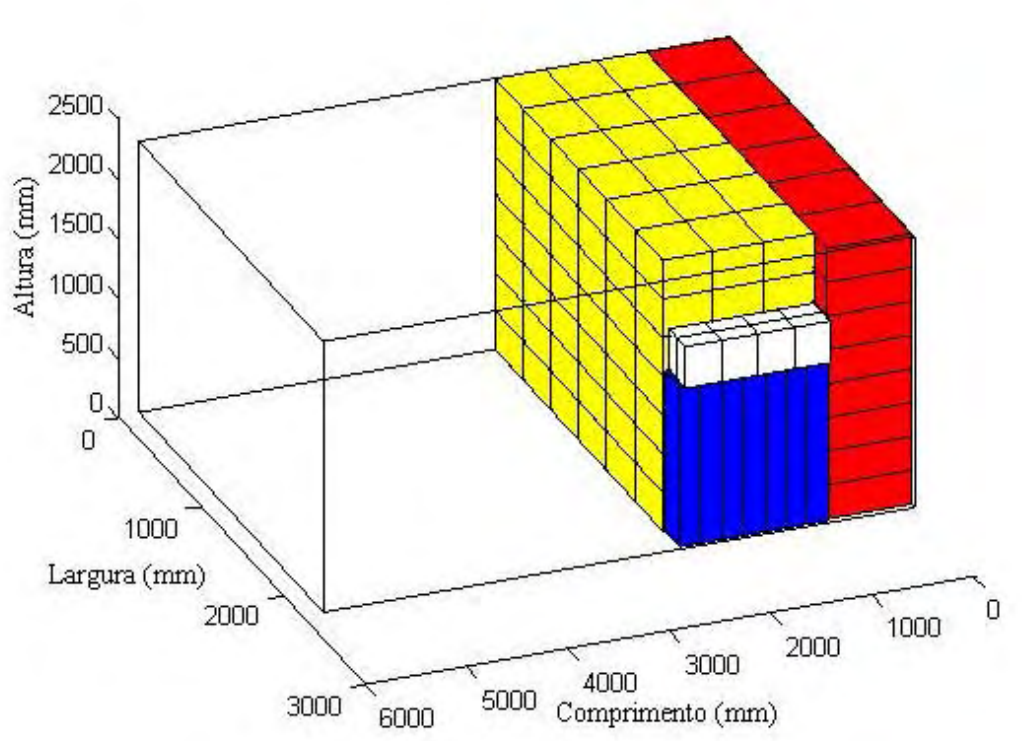
   Resumo do segundo arranjo principal:

   - 126 caixas do tipo 6 alocadas.
   - 7 caixas do tipo 2 alocadas.
   - 8 caixas do tipo 8 alocadas.
   - Volume ocupado: 126(0,0542) + 7(0,049831) + 8(0,012353) = 7,276934 m^3.
   - Volume usado: $1,485 \times 2,236 \times 2,261 = 7,5076\ m^3$.
   - Taxa de ocupação: 96,9281 %.
   - Profundidade usada: 2305 mm.

   Na Figura 17 apresentamos o carregamento do segundo arranjo, com 141 caixas.

   Na Tabela 19 apresentamos os tipos e quantidades de caixas que não foram carregadas.

Figura 4.1: Conjunto DA2: Carregamento do segundo arranjo.



Fonte: Informações da pesquisa da autora.

Tabela 19: Conjunto DA2: Caixas restantes após o segundo arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	44	0,0846	3,7239
2	1332	179	209	12	0,0498	0,5979
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	14	0,0542	0,7588
7	428	280	386	89	0,0463	4,1170
8	356	347	100	19	0,0124	0,2347

Fonte: Informações da pesquisa da autora.

3. **Terceiro arranjo principal:**

   Escolhemos as caixas do tipo 7 que têm volume total corrente de $4,1170m^3$.

O melhor arranjo principal seria na forma $386 \times 428 \times 280$. Assim, o arranjo principal assume a seguinte distribuição: $2 \times 5 \times 7 = 70$, isto é, o arranjo principal é formado por 70 caixas do tipo 7 e duas camadas.

- Espaço residual de altura: $772 \times 2140 \times 301$.

  Neste caso pode ser usado esse espaço residual para alocar 10 caixas do tipo 7 com a seguinte estrutura: $386 \times 428 \times 280$, isto é, $2 \times 5 \times 1 = 10$ caixas do tipo 7.

Resumo do terceiro arranjo principal:

- 80 caixas do tipo 7 alocadas.
- Volume ocupado: 80(0,0463) = 3,7007 m^3.
- Volume usado: $0,772 \times 2,236 \times 2,261 = 3,9029\ m^3$.
- Taxa de ocupação: 94,8182 %.
- Profundidade usada: 3077 mm.

Na Figura 18 apresentamos o carregamento do terceiro arranjo, com 80 caixas.

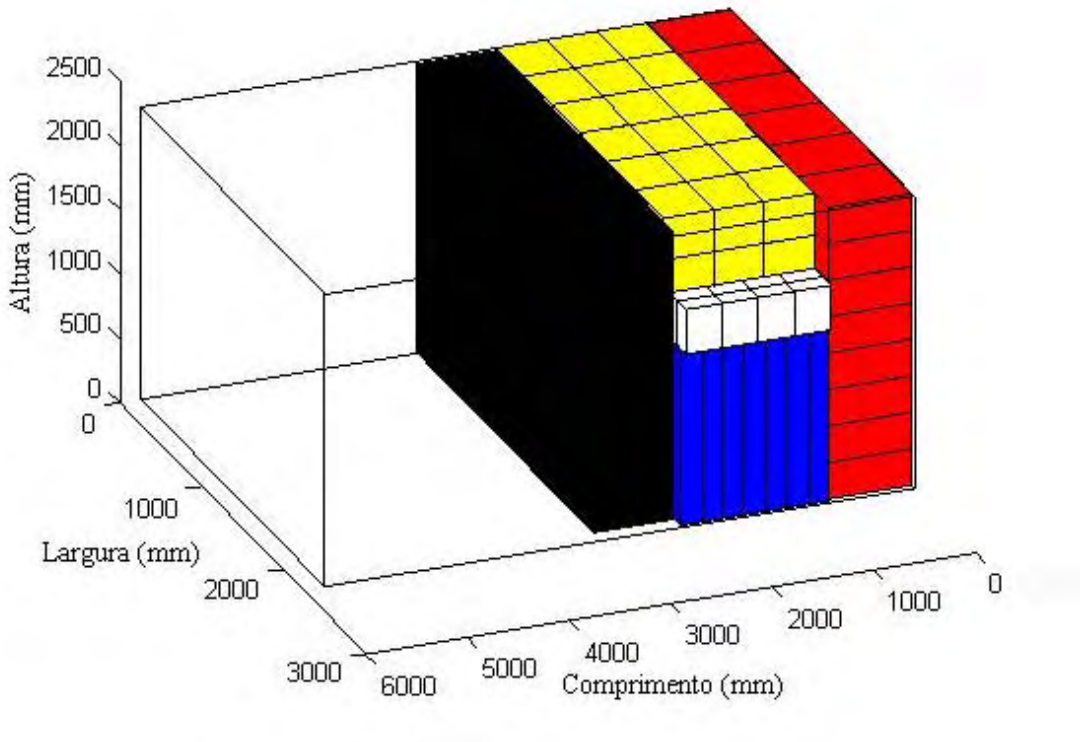
Na Tabela 20 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

Tabela 20: Conjunto DA2: Caixas restantes após o terceiro arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	44	0,0846	3,7239
2	1332	179	209	12	0,0498	0,5979
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	14	0,0542	0,7588
7	428	280	386	9	0,0463	0,4163
8	356	347	100	19	0,0124	0,2347

Fonte: Informações da pesquisa da autora.

Figura 18: Conjunto DA2: Carregamento do terceiro arranjo.



Fonte: Informações da pesquisa da autora.

4. **Quarto arranjo principal:**

   Escolhemos as caixas do tipo 1 que têm volume total corrente de $3,7239m^3$.

   O melhor arranjo principal seria na forma $366 \times 820 \times 282$. Assim, o arranjo principal assume a seguinte distribuição: $2 \times 2 \times 8 = 32$, isto é, o arranjo principal é formado por 32 caixas do tipo 1 e duas camadas.

   - Espaço residual de largura: $732 \times 596 \times 2261$.

     Neste caso pode ser usado esse espaço residual para alocar inicialmente 8 caixas do tipo 1 com a seguinte estrutura: $366 \times 282 \times 820$, isto é, $2 \times 2 \times 2 = 8$ caixas do tipo 1.

     Existe um espaço residual de altura de $732 \times 564 \times 621$.

     Podem ser alocadas 12 caixas tipo 8 com a seguinte estrutura: $356 \times 347 \times 100$, isto é, $2 \times 1 \times 6 = 12$ caixas do tipo 8.

     Existe espaço residual de largura, isto é, ao lado das 12 caixas do tipo 8.

Espaço residual de largura: $712 \times 217 \times 621$.

No espaço residual de largura são alocadas 4 caixas do tipo 8 na forma $356 \times 100 \times 347$, isto é, $2 \times 2 \times 1 = 4$ caixas do tipo 8.

Resumo do quarto arranjo principal:

- 40 caixas do tipo 1 alocadas.
- 16 caixas do tipo 8 alocadas.
- Volume ocupado: 40(0,084634) + 16(0,0124) = 3,5830 m^3.
- Volume usado: $0,732 \times 2,236 \times 2,261 = 3,7007\ m^3$.
- Taxa de ocupação: 96,8198 %.
- Profundidade usada: 3809 mm.

Na Figura 19 apresentamos o carregamento do quarto arranjo, com 56 caixas.

Na Tabela 21 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

Tabela 21: Conjunto DA2: Caixas restantes após o quarto arranjo.

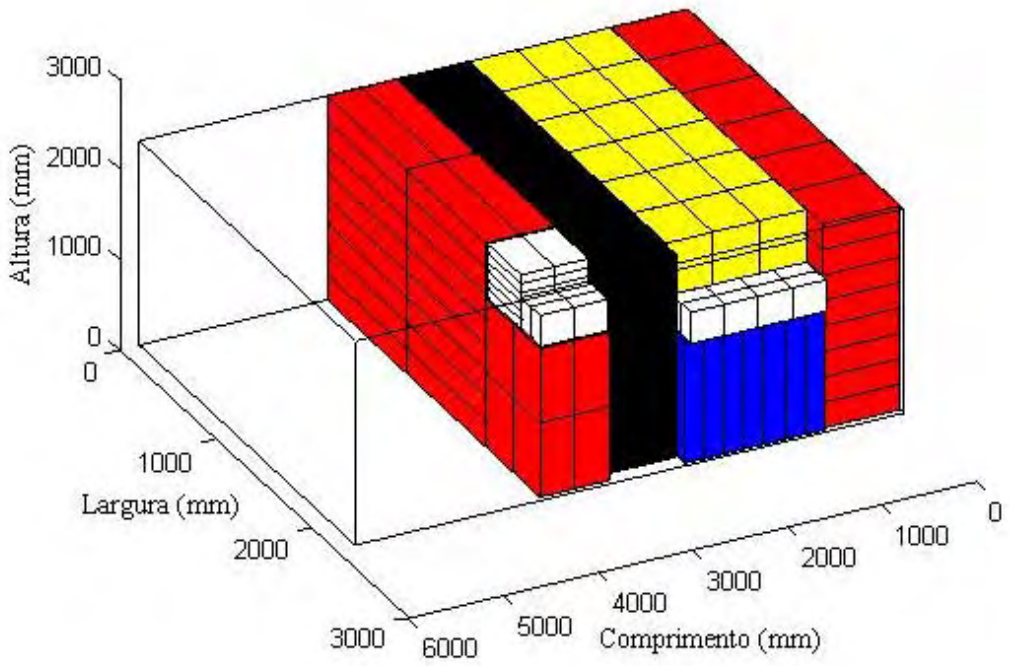
i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	4	0,0846	0,3385
2	1332	179	209	12	0,0498	0,5979
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	43	0,0761	3,2712
6	495	323	339	14	0,0542	0,7588
7	428	280	386	9	0,0463	0,4163
8	356	347	100	3	0,0124	0,037059

Fonte: Informações da pesquisa da autora.

5. **Quinto arranjo principal:**

Escolhemos as caixas do tipo 5 que têm volume total corrente de $3,2712 m^3$.

Figura 19: Conjunto DA2: Carregamento do quarto arranjo.



Fonte: Informações da pesquisa da autora.

O melhor arranjo principal seria na forma $340 \times 838 \times 267$. Assim, o arranjo principal assume a seguinte distribuição: $2 \times 2 \times 8 = 32$, isto é, o arranjo principal é formado por 32 caixas do tipo 5 e duas camadas.

- Espaço residual de altura: $680 \times 1676 \times 125$.

  No espaço residual de altura são alocadas as 3 caixas do tipo 8 na forma $356 \times 100 \times 347$ com a distribuição $1 \times 3 \times 1 = 3$ caixas do tipo 8.

  Existe um espaço residual lateral que não pode ser usado $680 \times 635 \times 125$, isto é, não tem mais caixas.

- Espaço residual de largura: $680 \times 560 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 8 caixas tipo 5 na seguinte estrutura: $340 \times 267 \times 838$, isto é, $2 \times 2 \times 2 = 8$ caixas do tipo 5.

  Existe um espaço residual de altura de $680 \times 534 \times 585$.

Podem ser alocadas 2 caixas tipo 6 na seguinte estrutura: $339 \times 495 \times 323$, isto é, $2 \times 1 \times 1 = 2$ caixas do tipo 6.

Resumo do quinto arranjo principal:

- 40 caixas do tipo 5 alocadas.
- 3 caixas do tipo 8 alocadas.
- 2 caixas do tipo 6 alocadas.
- Volume ocupado: 40(0,076074) + 3(0,012353) + 2(0,054201) = 3,188421 m^3.
- Volume usado: $0,680 \times 2,236 \times 2,261 = 3,437805\ m^3$.
- Taxa de ocupação: 92,7458 %.
- Profundidade usada: 4489 mm.

Na Figura 20 apresentamos o carregamento do quinto arranjo, com 45 caixas.

Na Tabela 22 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

Tabela 22: Conjunto DA2: Caixas restantes após o quinto arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	4	0,0846	0,3385
2	1332	179	209	12	0,0498	0,5979
3	1404	168	191	31	0,0451	1,3966
4	913	329	320	12	0,0961	1,1534
5	838	267	340	3	0,0761	0,2283
6	495	323	339	12	0,0542	0,6504
7	428	280	386	9	0,0463	0,4163

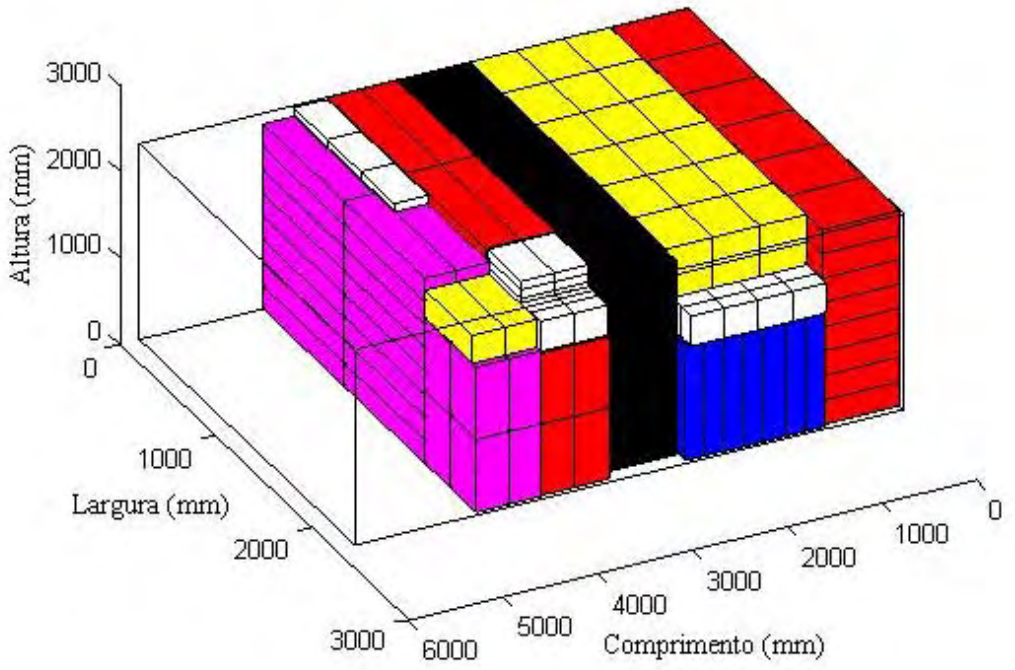
Fonte: Informações da pesquisa da autora.

6. **Sexto arranjo principal:**

   Escolhemos as caixas do tipo 3 que têm volume total corrente de $1,3966 m^3$.

   O melhor arranjo principal seria na forma $168 \times 191 \times 1404$. Assim, o arranjo principal assume a seguinte distribuição: $2 \times 11 \times 1 = 22$, isto é, o arranjo principal é formado por 22 caixas do tipo 3 e duas camadas.

Figura 20: Conjunto DA2: Carregamento do quinto arranjo.



Fonte: Informações da pesquisa da autora.

- Espaço residual de altura: $336 \times 2101 \times 857$.

  No espaço residual de altura são alocadas as 8 caixas do tipo 3 na forma $168 \times 1404 \times 191$, isto é, $2 \times 1 \times 4 = 8$ caixas do tipo 3.

  Existe ainda um espaço residual de largura de $336 \times 697 \times 857$, isto é ao lado das 8 caixas do tipo 3.

  Neste caso pode ser usado esse espaço residual para alocar inicialmente 2 caixas do tipo 6 na seguinte estrutura: $323 \times 339 \times 495$, isto é, $1 \times 2 \times 1 = 2$ caixas do tipo 6.

  Existe um espaço residual de altura de $323 \times 678 \times 362$, isto é acima das 2 caixas do tipo 6.

  Pode ser alocada 1 caixa do tipo 6 na seguinte estrutura: $323 \times 495 \times 339$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 6.

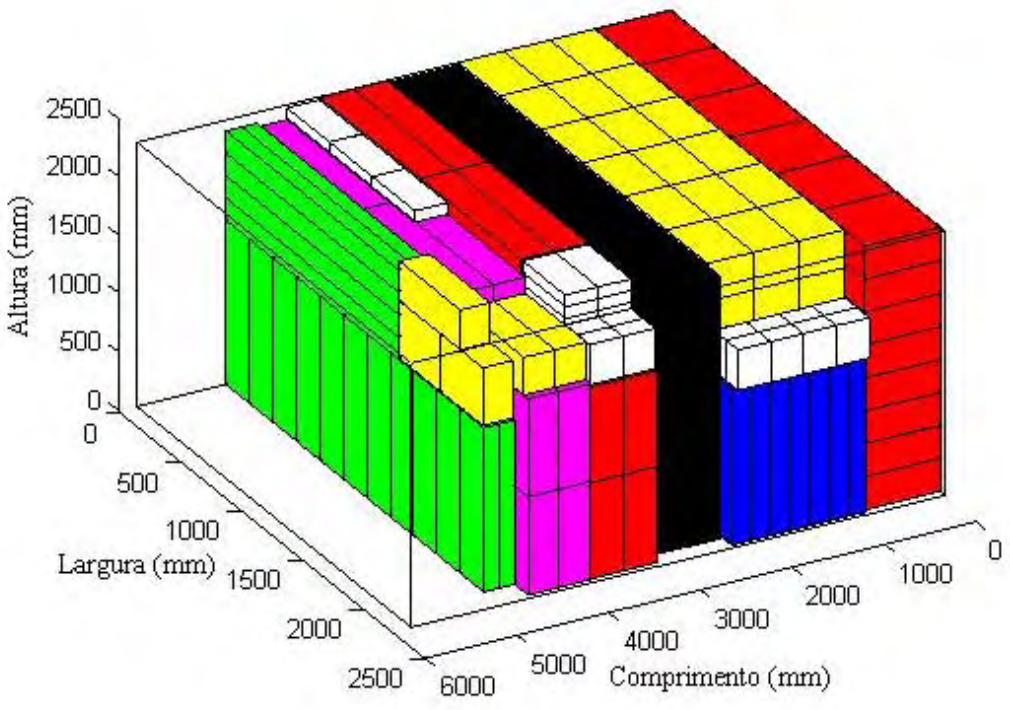
- Espaço residual de largura: $336 \times 135 \times 2261$, onde não existe possibilidade de carregar caixas.

Resumo do sexto arranjo principal:

- 30 caixas do tipo 3 alocadas.
- 3 caixas do tipo 6 alocadas.
- Volume ocupado: 30(0,045052) + 3(0,054201) = 1,514163 m^3.
- Volume usado: $0,336 \times 2,236 \times 2,261 = 1,69868\ m^3$.
- Taxa de ocupação: 89,1376 %.
- Profundidade usada: 4825 mm.

Na Figura 21 apresentamos o carregamento do sexto arranjo, com 33 caixas.

Figura 21: Conjunto DA2: Carregamento do sexto arranjo.



Fonte: Informações da pesquisa da autora.

Na Tabela 23 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

Tabela 23: Conjunto DA2: Caixas restantes após o sexto arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	4	0,0846	0,3385
2	1332	179	209	12	0,0498	0,5979
3	1404	168	191	1	0,0451	0,0451
4	913	329	320	12	0,0961	1,1534
5	838	267	340	3	0,0761	0,2283
6	495	323	339	9	0,0542	0,4878
7	428	280	386	9	0,0463	0,4163

Fonte: Informações da pesquisa da autora.

7. **Sétimo arranjo principal:**

   Escolhemos as caixas do tipo 4 que têm volume total corrente de $1,1534m^3$.

   O melhor arranjo principal seria na forma $320 \times 913 \times 329$. Assim, o arranjo principal assume a seguinte distribuição: $1 \times 2 \times 5 = 10$, isto é, o arranjo principal é formado por 10 caixas do tipo 4 e uma camada.

   - Espaço residual de altura: $320 \times 1826 \times 616$.

     No espaço residual de altura são alocadas as 2 caixas do tipo 4 na forma $320 \times 913 \times 329$, isto é, $1 \times 2 \times 1 = 2$ caixas do tipo 4.

     Existe ainda um espaço residual de altura de $320 \times 1826 \times 287$, isto é, acima das 2 caixas do tipo 4.

     Pode ser alocado 1 caixa tipo 2 na seguinte estrutura: $209 \times 1332 \times 179$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 2.

     Não existe possibilidade de alocar caixas no espaço residual de altura de $209 \times 1332 \times 108$.

     Existe um espaço residual de largura de $320 \times 494 \times 287$. Neste caso pode não ser usado esse espaço residual. Não existe possibilidade de alocar caixas no espaço residual.

   - Espaço residual de largura: $320 \times 410 \times 2261$.

     Neste caso pode ser usado esse espaço residual para alocar 2 caixas do tipo 1 com a seguinte estrutura: $282 \times 366 \times 820$, isto é, $1 \times 1 \times 2 = 2$ caixas do tipo 1.

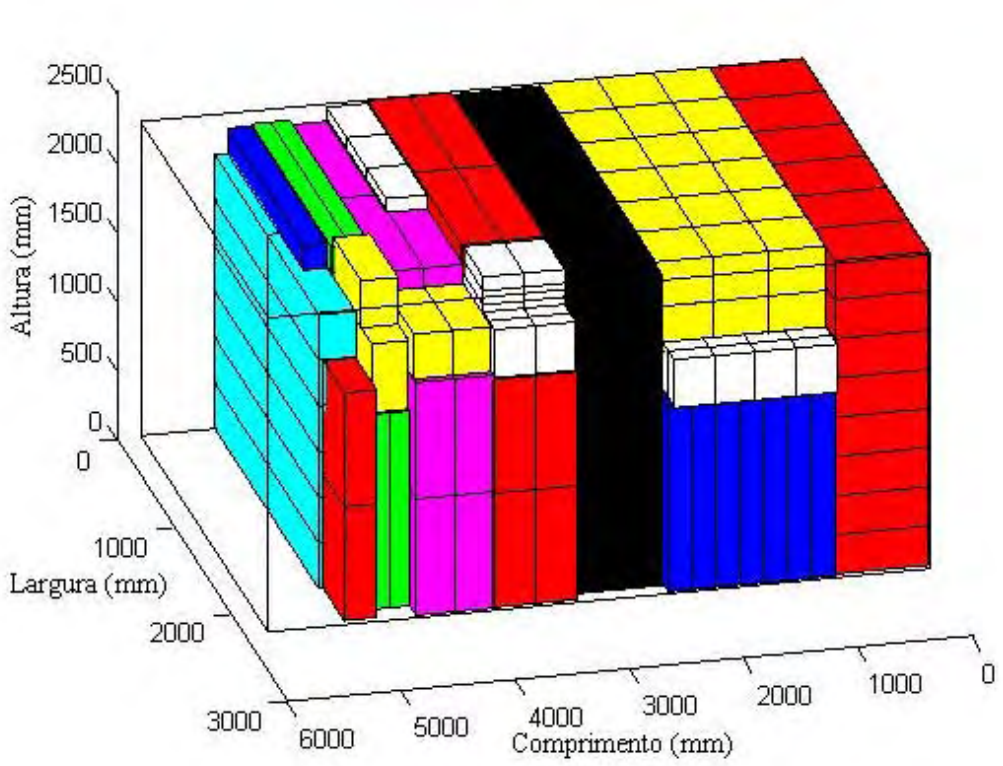
     Existe um espaço residual de altura de $282 \times 366 \times 621$. Não é possível alocar caixas no espaço residual de altura.

Resumo do sétimo arranjo principal:

- 12 caixas do tipo 4 alocadas.
- 1 caixa do tipo 2 alocada.
- 2 caixas do tipo 1 alocadas.
- Volume ocupado: 12(0,096121) + 1(0,049831) + 2(0,084634) = 1,279971 m^3.
- Volume usado: $0,320 \times 2,236 \times 2,261 = 1,617791\ m^3$.
- Taxa de ocupação: 79,1184 %.
- Profundidade usada: 5145 mm.

Na Figura 22 apresentamos o carregamento do sétimo arranjo, com 15 caixas.

Figura 22: Conjunto DA2: Carregamento do sétimo arranjo.



Fonte: Informações da pesquisa da autora.

Na Tabela 24 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

Tabela 24: Conjunto DA2: Caixas restantes após o sétimo arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	2	0,0846	0,169268
2	1332	179	209	11	0,0498	0,548141
3	1404	168	191	1	0,0451	0,0451
5	838	267	340	3	0,0761	0,2283
6	495	323	339	9	0,0542	0,4878
7	428	280	386	9	0,0463	0,4163

Fonte: Informações da pesquisa da autora.

8. **Oitavo arranjo principal:**

   Escolhemos as caixas do tipo 2 que têm volume total corrente de $0,548141m^3$.

   O melhor arranjo principal seria na forma $179 \times 1332 \times 209$. Assim, o arranjo principal assume a seguinte distribuição: $1 \times 1 \times 9 = 9$, isto é, o arranjo principal é formado por 9 caixas do tipo 2 e uma camada.

   - Espaço residual de altura: $179 \times 1332 \times 380$.

     No espaço residual de altura é alocada 1 caixa do tipo 2 com a forma $179 \times 1332 \times 209$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 2.

   - Espaço residual de largura: $179 \times 904 \times 2261$.

     Neste caso pode ser usado esse espaço residual para alocar 1 caixa do tipo 2 com a seguinte estrutura: $179 \times 209 \times 1332$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 2.

     O espaço residual de largura apresenta as seguintes dimensões: $179 \times 695 \times 2261$. Nesse espaço pode ser carregada 1 caixa tipo 3 na seguinte estrutura: $168 \times 191 \times 1404$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 3.

     Existe um espaço residual de largura de $179 \times 504 \times 2261$. Não é possível alocar caixas no espaço residual.

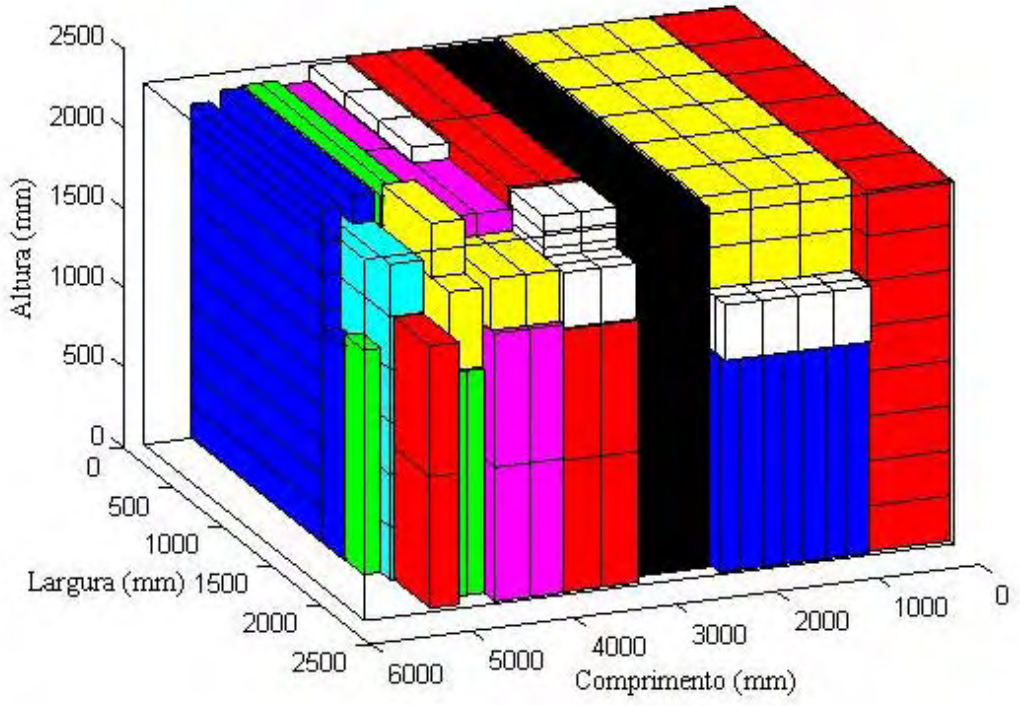
   Resumo do oitavo arranjo principal:

   - 11 caixas do tipo 2 alocadas.
   - 1 caixa do tipo 3 alocada.

- Volume ocupado: 11(0,049831) + 1(0,045052) = 0,593193 m^3.
- Volume usado: $0,320 \times 2,236 \times 2,261 = 0,904952\, m^3$.
- Taxa de ocupação: 65,5497 %.
- Profundidade usada: 5324 mm.

Na Figura 23 apresentamos o carregamento do oitavo arranjo, com 12 caixas.

Figura 23: Conjunto DA2: Carregamento do oitavo arranjo.



Fonte: Informações da pesquisa da autora.

Na Tabela 25 apresentamos os tipos e quantidades de caixas que não foram carregadas, para iniciar um novo arranjo.

9. **Última camada:**

   Escolhemos as caixas do tipo 7 que têm volume total corrente de $0,4163 m^3$.

   O melhor arranjo principal seria na forma $386 \times 428 \times 280$. Assim, o arranjo principal assume a seguinte distribuição: $1 \times 1 \times 8 = 8$, isto é, o arranjo principal é formado por 8 caixas do tipo 7 e uma camada.

Tabela 25: Conjunto DA2: Caixas restantes após o oitavo arranjo.

i	c_i	l_i	h_i	q_i	v_i	vt_i
	(mm)	(mm)	(mm)	m^3	m^3	
1	820	282	366	2	0,0846	0,169268
5	838	267	340	3	0,0761	0,2283
6	495	323	339	9	0,0542	0,4878
7	428	280	386	9	0,0463	0,4163

Fonte: Informações da pesquisa da autora.

- Espaço residual de largura: $386 \times 1808 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar 1 caixa do tipo 7 na seguinte estrutura: $386 \times 428 \times 280$, isto é, $1 \times 1 \times 1 = 1$ caixa do tipo 7.

  Existe um espaço residual de altura com dimensões $386 \times 428 \times 1981$.

  No espaço residual de altura são alocadas as 2 caixas do tipo 1 com a forma $366 \times 282 \times 820$ com a distribuição $1 \times 1 \times 2 = 2$ caixas do tipo 1.

  O espaço residual de altura apresenta as dimensões: $366 \times 282 \times 341$ onde não é possível alocar caixas.

  Existe ainda um espaço residual de largura com dimensões $386 \times 146 \times 1981$, onde não pode ser alocada nenhuma caixa.

- Espaço residual de largura: $386 \times 1380 \times 2261$.

  Neste caso pode ser usado esse espaço residual para alocar as 3 caixas tipo 5 na seguinte estrutura: $340 \times 838 \times 267$, isto é, $1 \times 1 \times 3 = 3$ caixas do tipo 5.

  Espaço residual de altura: $340 \times 838 \times 1460$.

  No espaço residual de altura são alocadas 4 caixas do tipo 6 com a forma $339 \times 495 \times 323$ com a distribuição $1 \times 1 \times 4 = 4$ caixas do tipo 6.

  Existe ainda um espaço residual de largura com dimensões $340 \times 343 \times 1460$

  Nesse espaço podem ser alocadas 2 caixas do tipo 6 com a seguinte estrutura: $339 \times 323 \times 495$, isto é $1 \times 1 \times 2 = 2$ caixas do tipo 6.

- Finalmente existe um espaço residual de largura com dimensões $386 \times 542 \times 2261$. Nesse espaço podem ser alocadas as 3 caixas do tipo 6 com a seguinte estrutura: $339 \times 495 \times 323$, isto é, $1 \times 1 \times 3 = 3$ caixas do tipo 6.

Resumo do último arranjo principal:

- 2 caixas do tipo 1 alocadas.
- 3 caixas do tipo 5 alocadas.
- 9 caixas do tipo 6 alocadas.
- 9 caixa tipo 7 alocada.
- Volume ocupado: 2(0,084634) + 5(0,0761) + 9(0,0542) + 9(0,0463) = 1,454268 m^3.
- Volume usado: $0,386 \times 2,236 \times 2,261 = 1,95146\ m^3$.
- Taxa de ocupação: 74,5221 %.

Ao final do processo temos os seguintes resultados:

- Profundidade usada: 5710 mm.
- Volume residual de profundidade: 83 mm.
- Taxa de ocupação total: 92,1210 %.

Deve-se observar que em todos os testes realizados usando o algoritmo GRASP de Leite (2007) não foram possíveis alocar todas as caixas. No melhor resultado reportado em Leite (2007) ficaram de fora 15 caixas (12 de tipo 1, 2 de tipo 4 e 1 de tipo 5) representando um volume de 1,2835 m^3.

Finalmente, a Tabela 26 mostra o volume ocupado em cada iteração e a eficiência de preenchimento que é a relação entre o volume ocupado dividido pelo volume usado.

Da Tabela 26 podemos verificar que nas 4 primeiras iterações a eficiência de preenchimento foi muito elevada e essa eficiência diminui nas últimas 3 iterações. Esse resultado era esperado já que nas últimas iterações existem menos possibilidades e frequentemente os espaços residuais não podem ser preenchidos. Por esse motivo uma estratégia interessante podem ser evitar usar as caixas com dimensões menores nas iterações iniciais.

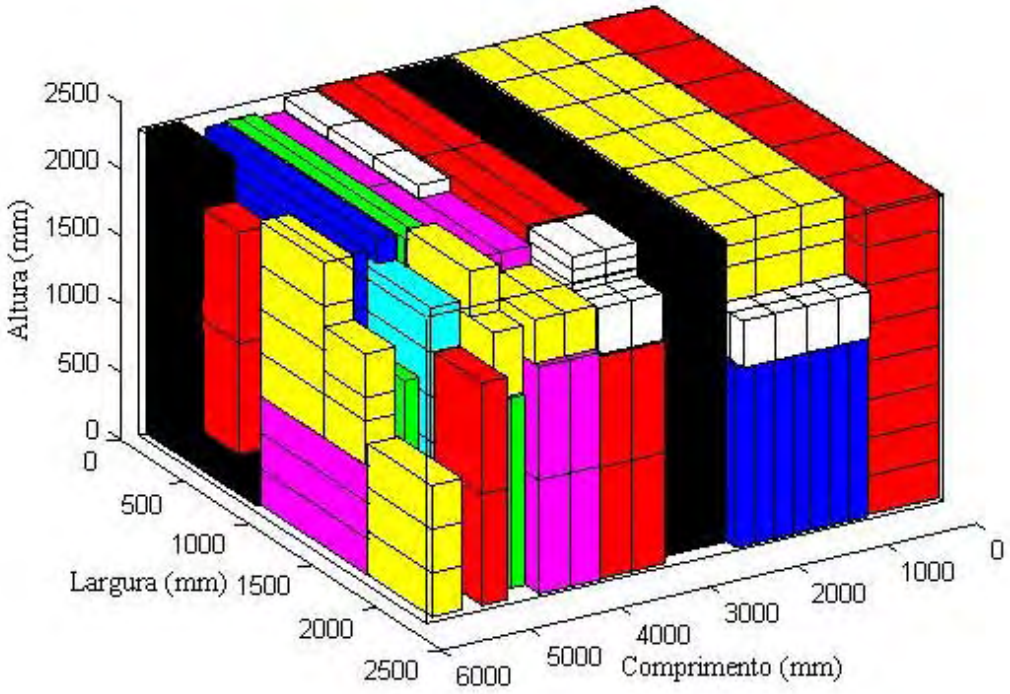
Na Figura 24 apresentamos o carregamento completo da instância de DA2 do contêiner, com todas as 453 caixas.

Tabela 26: Conjunto DA2: Eficiência de preenchimento por iteração

No	Volume usado	Eficiência
	m^3	(%)
1	4,0608	97,9547
2	7,2769	96,9281
3	3,7007	94,8182
4	3,5830	96,8198
5	3,1884	92,7458
6	1,5142	89,1376
7	1,27997	79,1184
8	0,5932	65,5497
9	1,4543	74,5221

Fonte: Informações da pesquisa da autora.

Figura 24: Conjunto DA2: Carregamento completo, em nove arranjos.



Fonte: Informações da pesquisa da autora.

4.3 Detalhes de Implementação Computacional

O algoritmo heurístico proposto foi testado para caixas heterogêneas e um único container. O programa foi elaborado em Fortran e os testes foram feitos em um PC Dell Processador Core 2

duo/2GB e memória RAM 2GB, com sistema operacional Windows XP.

Para testes foram usados várias instâncias do problema cujos dados se encontram disponíveis de forma completa na literatura especializada. Entretanto, esses dados se encontram no Apêndice. Analisando os dados das instâncias usadas em testes, podemos verificar que todos eles são fracamente heterogêneos ou fortemente heterogêneos. Adicionalmente, existem dois tipos de dados com relação ao volume total das caixas disponíveis: (1) instâncias em que o volume total das caixas disponíveis é menor que o volume do contêiner e, (2) instâncias em que o volume total das caixas disponíveis é maior que o volume do contêiner. No primeiro tipo de instâncias geralmente o aproveitamento no preenchimento é relativamente menor (em torno de 90%), quando comparados com as instâncias em que o volume de caixas disponíveis é maior que o volume do contêiner. Neste tipo de caso o aproveitamento no preenchimento varia de 92 a 96%.

Para mostrar o desempenho do preenchimento das caixas no contêiner usamos o conceito de taxa de ocupação que é definida da seguinte maneira:

$$\text{Taxa de ocupação} = \frac{\text{Volume ocupado pelas caixas}}{\text{volume usado do contêiner}}$$

Quando sobram caixas no preenchimento do contêiner então o volume usado no contêiner é o próprio volume total do contêiner. Entretanto, quando todas as caixas foram alocadas no contêiner, então o volume usado no contêiner V_{us} é o seguinte:

$$V_{us} = W_{us} \times L \times H$$

onde W_{us} é o comprimento usado no contêiner no preenchimento das caixas.

4.4 Testes Usando o Algoritmo Heurístico Construtivo

O AHC proposto foi testado em 13 instâncias do problema:

- Nove instâncias apresentadas por Leite (2007), chamados de DA1, DA2, DA3, DA4, DA5, DA6, DA7, DA8 e DA9;
- Uma instância apresentada por George e Robinson (1980), denominado de GR;
- Duas instâncias apresentadas por Rodrigues (2005), denominadas de Ro1 e Ro2;

- Uma instância apresentada por Pisinger (2002), chamada de Pis;

Em todas as 13 instâncias foram utilizados apenas um único contêiner tipo I (29,287m^3), tipo II (7,290m^3), tipo III (31,211m^3) e tipo IV (33,130m^3) e caixas heterogêneas. As instâncias de cada teste utilizado neste trabalho pode estão no Apêndice.

Leite (2007) gerou as nove instâncias de forma aleatoria para avaliar o desempenho do algoritmo GRASP desenvolvido, essas instâncias apresentam 8, 12 e 20 tipos de caixas cujas dimensões de comprimento, largura, altura e quantidade foram gerados de forma aleatória dentro de um intervalo previamente especificado.

Na Tabela 27 apresentamos os resultados do AHC desenvolvido neste trabalho e a Tabela 28 apresentamos, para cada instância, a melhor solução encontrada pelo algoritmo GRASP de Leite (2007), utilizando o contêiner tipo I (ver Apêndice) e para as instâncias DA1, DA2, DA3 e DA4 em que existem oito tipos de caixas diferentes e um total de 306, 453, 679 e 471 caixas, respectivamente. Nos dados gerados por Leite (2007), existem instâncias (DA1, DA2 e DA3) em que o volume total de caixas (15,919m^3, 26,5929m^3 e 26,205m^3) é menor que o volume do contêiner I e existe uma instância (DA4) em que o volume total de caixas (31,940m^3) ultrapassa o volume do contêiner.

Nas Tabelas 27 e 28 também apresentamos o resultado de teste da instância de George e Robinson (1980), que segundo Cecilio (2003) trata-se de um exemplo de uma situação real, envolvendo um contêiner padrão ISO série 2 com o volume total de caixas (26.323824m^3) menor que o volume do contêiner I. Esta instância é comparado com os resultados encontrados por Leite (2007).

Tabela 27: Resultados do AHC

Instâncias	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas								Tempo (seg.)
				1	2	3	4	5	6	7	8	
DA1	0	86,81	2166	0	0	0	0	0	0	0	0	$<$ 0,01
DA2	0	92,12	83	0	0	0	0	0	0	0	0	$<$ 0,01
DA3	0	92,89	213	0	0	0	0	0	0	0	0	$<$ 0,01
DA4	90	94,13	60	1	24	41	0	0	16	0	8	$<$ 0,01
GR	0	90,62	47	0	0	0	0	0	0	0	0	$<$ 0,01

Fonte: Informações da pesquisa da autora.

As instâncias DA5, DA6, DA7 e DA8, geradas por Leite (2007), apresentam 12 tipos de caixas diferentes e um total de 614, 785, 661 e 458 caixas respectivamente. Nas instâncias DA5, DA6 e DA7, o volume total de todas as caixas é de 24,636m^3, 27,372m^3 e 24,637m^3 respectivamente e,

Tabela 28: Melhores resultados obtidos em Leite (2007)

Instâncias	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas								Tempo (seg.)
				1	2	3	4	5	6	7	8	
DA1	0	67,98	- - -	0	0	0	0	0	0	0	0	0,05
DA2	15	86,42	- - -	12	0	0	2	1	0	0	0	0,36
DA3	0	85,06	- - -	0	0	0	0	0	0	0	0	0,03
DA4	105	82,95	- - -	0	0	2	8	2	16	0	77	0,06
GR	0	89,99	- - -	0	0	0	0	0	0	0	0	—-

Fonte: Leite (2007).

portanto, apresentam um volume menor que o volume do contêiner. Por outro lado, a instância DA8 apresenta um volume de 33,275m^3 e, portanto, apresenta um volume que ultrapassa o volume do contêiner I, com 29,287m^3. Os resultados dos testes com essas quatro instâncias são mostrados na Tabela 29 para o AHC desenvolvido neste trabalho e na Tabela 30 são apresentados os melhores resultados encontrados em Leite (2007) usando um algoritmo GRASP.

Tabela 29: Resultados do AHC

Instân-cias	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas												Tempo (seg.)
				1	2	3	4	5	6	7	8	9	10	11	12	
DA5	0	93,77	596	0	0	0	0	0	0	0	0	0	0	0	0	$<$ 0,01
DA6	7	92,59	47	0	0	0	0	0	0	0	0	0	0	0	7	$<$ 0,01
DA7	0	91,81	485	0	0	0	0	0	0	0	0	0	0	0	0	$<$ 0,01
DA8	154	92,79	48	0	0	9	2	5	20	9	20	25	27	37	0	$<$ 0,01

Fonte: Informações da pesquisa da autora.

Nas Tabelas 31 e 32 são mostrados os resultados encontrados para a instância DA9 gerada por Leite (2007) e a instância Pis analisada em Pisinger (2002). Os resultados da Tabela 31 foram obtidos pelo AHC desenvolvido neste trabalho. O resultado da instância DA9 mostrada na Tabela 32 é o melhor resultado encontrado por Leite (2007) usando um algoritmo GRASP e o resultado da instância Pis mostrado na Tabela 32 é o resultado encontrado pela heurística de Pisinger (2002). Ambas instâncias apresentam 20 tipos de caixas diferentes sendo que a instância de Pisinger (2002) (Pis) é altamente heterogênea. A instância de Leite (2007) apresenta um total de 930 caixas, um volume total de todas as caixas igual a 23,678m^3 e usa o contêiner tipo I com volume igual a 29,287m^3

Tabela 30: Melhores resultados obtidos em Leite (2007)

Instân-cias	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas												Tempo (seg.)
				1	2	3	4	5	6	7	8	9	10	11	12	
DA5	0	83,58	- -	0	0	0	0	0	0	0	0	0	0	0	0	0,12
DA6	30	88,87	- -	14	0	0	0	6	0	7	0	0	3	0	0	0,33
DA7	0	84,97	- -	0	0	0	0	0	0	0	0	0	0	0	0	0,09
DA8	65	88,30	- -	0	0	0	37	12	10	0	0	0	2	4	0	0,19

Fonte: Leite (2007).

(ver Apêndice). Por outro lado, a instância de Pisinger (2002) (Pis) apresenta um total de 146 caixas, um volume total 28,664194m^3 e usa o contêiner tipo III com volume igual a 31,211m^3) (ver Apêndice).

Tabela 31: Resultados do AHC

Instâncias	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas											Tem. (seg.)
				1	...	4	...	9	...	11	12	13	...	20	
DA9	0	89,16	540	0	...	0	...	0	...	0	0	0	...	0	$<$ 0,01
Pis	6	85,83	00	0	...	1	...	1	...	2	1	1	...	0	$<$ 0,01

Fonte: Informações da pesquisa da autora.

Tabela 32: Resultados obtidos de Leite (2007) e Pisinger (2002)

Instâncias	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas											Tem. (seg.)
				1	2	3	4	5	6	7	8	9	...	20	
DA9	0	81,90	- -	0	0	0	0	0	0	0	0	0	...	0	0,45
Pis	0	90,79	- -	0	0	0	0	0	0	0	0	0	...	0	—-

Fonte: Leite (2007) e Pisinger (2002).

Finalmente, a Tabela 33 apresenta os resultados encontrados pelo AHC desenvolvido neste trabalho para as duas instâncias (Ro1 e Ro2) apresentadas por Rodrigues (2005), que segundo o autor trata-se de um exemplo com dimensões de caixas reais de produtos eletroeletrônicos, com quantidade, tipo de caixas semelhantes e com o volume total de caixas de (8,232071m^3 e 44,958261m^3)

(no total de 100 e 285 caixas), envolvendo um contêiner tipo II e IV com o volume total de (7.290m^3 e 33.130m^3).

Tabela 33: Resultados do AHC

Instâncias	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas							Tempo
				1	2	3	4	5	6	7	(seg.)
Ro1	20	90,47	26	2	0	2	6	6	2	2	$<$ 0,01
Ro2	90	90,91	44	0	0	13	14	30	27	6	$<$ 0,01

Fonte: Informações da pesquisa da autora.

4.5 Análise Crítica dos Resultados Encontrados

O AHC proposto neste trabalho demonstra que se trata de uma técnica heurística construtiva eficiente para a resolução do PCC.

Segundo a analise do AHC e a comparação com os resultados da literatura especializada, em Leite (2007), Pisinger (2002) e Cecilio (2003), podemos concluir:

- **DA1:** O AHC apresentou (distribuídos em seis arranjos principais) o melhor resultado no percentual de volume ocupado em relação ao volume do contêiner sendo de 86,81%, quando comparado ao resultado mostrado em (LEITE, 2007) de 67,98%. O AHC demonstra ser mais eficiente principalmente, pelo espaço residual de 2166mm no comprimento do contêiner, sendo possível alocar qualquer tipo caixa de DA1, porém no trabalho de (LEITE, 2007), não existe menção explícita de que exista espaço residual no comprimento e que seja possível alocar caixas adicionais. O AHC carregou todas as caixas, de volume total de 15,919m^3, e volume do contêiner carregado de 18,3367m^3.

- **DA2:** Neste caso mostrado passo a passo em nove arranjos principais, no exemplo ilustrativo, deste trabalho tem-se que o AHC obteve o percentual de volume ocupado em relação ao volume do contêiner de 92,12% em relação ao resultado de 86,42%, encontrado com o algoritmo GRASP de (LEITE, 2007) temos uma diferença de 5,7%. Na profundidade do contêiner de 83mm, não usado não é possível alocar nenhum tipo de caixa, porém o AHC carregou todas

as 453 caixas disponíveis dentro do contêiner, enquanto que nos resultados encontrados por (LEITE, 2007) foram alocadas apenas 438 caixas, deixando de fora 12 caixas tipo 1, 2 caixas tipo 4 e 1 caixa tipo 5.

- **DA3:** No AHC o percentual de volume ocupado em relação ao volume do contêiner é de 92,89%, distribuídos em 8 arranjos principais e todas as caixas diponíveis alocadas, restando um espaço residual de 213mm, podendo ser carregados ainda quase todas as caixas de DA3. Em (LEITE, 2007) o volume utilizado do contêiner foi de 85,06% e alocadas todas as 679 caixas disponíveis, porém não é mencionado o tamanho do espaço residual no comprimento do contêiner. O AHC carregou todas as caixas, de volume total de (26,205m^3), e volume do contêiner carregado de (28,2102m^3).

- **DA4:** Fazendo uma análise comparativa, o resultado do AHC proposto mostra a melhor solução de todos os testes realizados neste trabalho: 94,13% de volume ocupado do contêiner, distribuído em 7 arranjos principais, em um total de 381 caixas carregadas e um espaço residual muito pequeno. O resultado encontrado por (LEITE, 2007), apresenta um percentual do volume de 82,95% e no total de 366 caixas carregadas. Podemos dizer que o AHC carrega 15 caixas a mais do que o algoritmo de (LEITE, 2007), a diferença do percentual de volume ocupado em relação ao volume do contêiner é de 11% e o AHC encontra um valor altíssimo de desempenho neste caso. Este teste é descrito passo a passo, arranjo por arranjo no apêndice deste trabalho.

- **GR:** O AHC apresentou (distribuídos em oito arranjos principais) o melhor resultado no percentual de volume ocupado em relação ao volume do contêiner sendo de 90,6173%, quando comparado ao resultado mostrado em (LEITE, 2007) de 89,99%. O AHC demonstra ser mais eficiente carregando todas as caixas, de volume total de (26,3238m^3), e volume do contêiner carregado de (29,0495m^3). Lembrando que na literatura, (GEORGE; ROBINSON, 1980) apresenta o percentual de volume ocupado de 89,74% (deixando 1 caixa tipo 7); (CECILIO, 2003) apresenta 89,88% de percentual de volume ocupado e (LEITE, 2007) apresenta 89,99%.

- **DA5:** Dos 9 testes abordados por (LEITE, 2007), o DA5 foi o segundo melhor resultado encontrado pelo AHC proposto, com o percentual de volume de 93,77%, distribuídos em 10 arranjos principais e um consideravel espaço residual de 596mm no comprimento do contêiner, o AHC mostra uma melhor solução quando comparada com o resultado encontrado em (LEITE, 2007), cujo o percentual de volume ocupado é de 83,58%. Tanto o AHC quanto (LEITE, 2007) car-

regam totas as 614 caixas disponíveis, porém no teste feito pelo AHC é possível alocar mais caixas de todos os 12 tipos de caixas de DA5.

- **DA6:** O AHC apresentou, distribuídos em doze arranjos principais, o melhor resultado no percentual de volume ocupado em relação ao volume do contêiner sendo de 92,59%, quando comparado ao resultado mostrado em (LEITE, 2007) de 88,87%. O AHC demonstra ser mais eficiente apesar de deixar de fora 7 caixas tipo 12. O AHC carrega 778 caixas de 12 tipos diferentes e restando um espaço residual pequeno de $47mm$, comparando com as 755 caixas alocadas de (LEITE, 2007).

- **DA7:** Neste caso o resultado encontrado pelo AHC proposto, com o percentual de volume de 91,81%, distribuídos em 10 arranjos principais e um considerável espaço residual de $485mm$ no comprimento do contêiner, o AHC mostra uma melhor solução quando comparado com o resultado encontrado em (LEITE, 2007), cujo percentual de volume ocupado é de 84,97%. Tanto o AHC quanto (LEITE, 2007) carregam totas as 661 caixas disponíveis, porém no teste feito pelo AHC é possível alocar mais caixas de todos os 12 tipos de caixas de DA7.

- **DA8:** Fazendo uma análise comparativa, o resultado do AHC proposto apresenta 92,79% de volume ocupado do contêiner, distribuídos em 4 arranjos principais, em um total de 304 caixas carregadas e um respaço residual muito pequeno. O resultado encontrado na literatura por (LEITE, 2007), cujo percentual do volume é de 88,30% e no total de 395 caixas carregadas. Podemos dizer que apesar do AHC carregar 91 caixas a menos do que o algoritmo de (LEITE, 2007), a diferença do percentual de volume ocupado em relação ao volume do contêiner é de 4,49% maior para o AHC.

- **DA9:** Dos 9 testes abordados por (LEITE, 2007), o DA9 foi o segundo resultado abaixo de 90% encontrado pelo AHC proposto, com o percentual de volume de 89,16%, distribuídos em 15 arranjos principais e um considerável espaço residual de $540mm$ no comprimento do contêiner, o AHC mostra uma melhor solução quando comparado com o resultado encontrado em (LEITE, 2007), cujo o percentual de volume ocupado é de 81,90%. Tanto o AHC quanto (LEITE, 2007) carregam totas as 930 caixas disponíveis, porém no resultado apresentado pelo AHC é possível alocar mais caixas de todos os 20 tipos de caixas de DA9.

- **Pis:** Neste teste apresentado com os dados de (PISINGER, 2002), não se mostrou muito satisfatório. O AHC proposto apresenta 85,8339% de volume ocupado do contêiner, distribuídos

em 11 arranjos principais, em um total de 140 caixas carregadas e nenhum respaço residual. O resultado encontrado na literatura por (PISINGER, 2002), cujo percentual do volume é de 90,79% e no total de 146 caixas carregadas. Neste caso o AHC demonstra ser menos eficiente deixando de fora: 1 caixa tipo 04, 1 caixa tipo 09, 2 caixas tipo 11, 1 caixa tipo 12 e 1 caixa tipo 13.

- **Ro1:** O AHC apresentou (distribuídos em seis arranjos principais) o melhor resultado no percentual de volume ocupado em relação ao volume do contêiner sendo de 90,4733%, quando comparado ao resultado mostrado em (RODRIGUES, 2005) de 83,20%. O AHC demonstra ser mais eficiente carregando 80 caixas de um total de 100 caixas, obtendo o volume total de caixas carregadas de 6,5955m^3.

- **Ro2:** Neste caso o resultado encontrado pelo AHC proposto, mostrou-se satisfatório. O AHC proposto apresenta 90,913% de volume ocupado do contêiner, distribuídos em 3 arranjos principais, em um total de 195 caixas carregadas e 44mm de respaço residual. O resultado encontrado na literatura por (RODRIGUES, 2005), cujo percentual do volume é de 70,19%. Neste caso o AHC demonstra ser eficiente, apesar de carregar um grande número de caixas, deixa de fora: 13 caixas tipo 03, 14 caixas tipo 04, 30 caixas tipo 05, 27 caixas tipo 06 e 6 caixas tipo 07.

4.6 Conclusões Parciais

Os resultados obtidos pelo AHC demonstra ser uma técnica eficiente na resolução do PCC levando em conta que se trata de uma heurística construtiva de tipo gulosa. Os resultados do AHC testados em 13 instâncias da literatura e cujos dados se encontram no Apêndice, podem ser considerados eficientes levando em conta a natureza desse tipo de algoritmo. O desempenho do AHC também é interessante porque supera a metaheurística GRASP desenvolvida em Leite (2007). Entretanto, deve-se observar que os algoritmos heurísticos construtivos geralmente não encontram soluções ótimas de instâncias grandes de problemas complexos.

Os resultados apresentados mostram que é possível empacotar as caixas heterogêneas com uma boa porcentagem de volume ocupado dentro do contêiner, quase sempre com 90%, e mesmo para

os casos onde não cabiam todas as caixas, foi possível observar um bom aproveitamento do espaço, alcançando o objetivo proposto, que é de otimizar o espaço ocioso dentro de um contêiner.

Dos resultados experimentais obtidos, conclue-se que é interessante a utilização do AHC para o PCC, demonstrando que a ideia do uso do AHC nesse tipo de problema é promissora mesmo que ainda se tenha espaço para novos refinamentos. Finalmente, deve-se observar que o AHC apresentado neste trabalho apresenta desempenho pouco eficiente quando se trata de resolver uma instância com caixas fortemente heterogêneas, isto é, muitos tipos de caixas e com um número pequeno de caixas de cada tipo. Nesse tipo de problemas, a chamada Estratégia 1 que gera arranjos com uma ou várias camadas é pouco acionada. Assim, um refinamento possível é melhorar a chamada Estratégia 2. No próximo Capítulo usamos o AHC, desenvolvido neste Capítulo, como base para desenvolver uma metaheurística GRASP. Assim, o AHC faz parte da fase construtiva da metaheurística GRASP.

Capítulo 5

A Metaheurística GRASP Aplicado ao Problema de Carregamento de Contêiner

5.1 Introdução

Neste Capítulo apresentamos o algoritmo GRASP proposto para resolver o PCC. Esse algoritmo foi desenvolvido usando como base a teoria apresentada no Capítulo 2 e o algoritmo heurístico construtivo apresentado no Capítulo 3. A parcela mais importante do algoritmo GRASP é a generalização do AHC apresentado no Capítulo 4 e a formulação de uma fase de melhoria local (busca local).

Os testes apresentados mostram que o algoritmo GRASP desenvolvido trabalha de forma adequada, para as instâncias usadas no Capítulo anterior e o tempo de processamento ainda é pequeno para as instâncias testadas.

Nas seguintes seções são apresentadas as diferentes parcelas do algoritmo GRASP, assim como os resultados encontrados.

5.2 A Fase de Pré-Processamento do Algoritmo GRASP

Na versão desenvolvida neste trabalho, não foi formulada uma fase de pré-processamento do GRASP para resolver o problema de carregamento de contêiner. Geralmente a fase de pré-processamento, tenta identificar características que permitam diminuir o espaço de busca do problema. Assim, para o

PPC não foi possível identificar essas características.

5.3 A Fase Construtiva do Algoritmo GRASP

A fase construtiva do GRASP é apenas uma generalização do AHC apresentado no Capítulo 4. Nesse capítulo, para identificar a proposta de arranjo principal mais promissora usamos uma proposta heurística. Para cada proposta de arranjo principal encontramos, o volume do arranjo residual de altura (V_{ra}), o volume do arranjo residual lateral (V_{rl}) e o volume de todas as caixas que podem ser alocadas no arranjo residual de altura e no arranjo residual lateral (V_{rc}). Assim, a melhor proposta de arranjo principal é aquela que apresenta o maior valor de I_r:

$$I_r = \frac{V_{rc}}{V_{ra} + V_{rl}}$$

O AHC identifica 4 arranjos usando o índice de desempenho I_r. Na fase construtiva, do GRASP generalizamos essa proposta de forma que o arranjo mais promissor seja escolhido usando a lógica de escolha do GRASP. Assim, seja $I_{r,i}$ o índice relacionado com o arranjo i. Nesse contexto, fazem parte da lista reduzida de candidatos RCL todos os arranjos que satisfazem a seguinte relação:

$$I_r^{min} + \alpha(I_r^{max} - I_r^{min}) \leq I_{r,i} \leq I_r^{max} \tag{28}$$

onde I_r^{min} é o índice do arranjo de pior qualidade, I_r^{max} é o índice do arranjo de melhor qualidade e o parâmetro α controla o tamanho da lista reduzida. Deve-se observar que quando $\alpha = 0$, então todos os arranjos formam parte da lista RCL e, portanto, o algoritmo GRASP se torna um processo de otimização totalmente aleatório. Por outro lado, quando $\alpha = 1$, então apenas o melhor arranjo faz parte da lista RCL e, portanto, o algoritmo GRASP da fase construtiva se torna equivalente ao AHC do capítulo anterior. Neste trabalho, apresentamos resultados de testes com valores de $\alpha = 0,8$ (significativamente guloso) e $\alpha = 0,5$ que representa um valor intermediário. Obviamente o número de elementos da lista RCL é variável porque depende do valor de α escolhido e dos valores de I_r^{min} e I_r^{max} que variam em cada passo da fase construtiva do algoritmo GRASP. Essa forma de encontrar os elementos da lista reduzida RCL é usada na estratégia 1 e na estratégia 2 do AHC apresentado no capítulo anterior.

Uma vez encontrado os elementos da lista RCL, então o elemento de RCL (arranjo escolhido) é escolhido de forma aleatória.

5.4 A Fase de Melhoria Local do Algoritmo GRASP

A fase de melhoria local é iniciada após o final da fase construtiva em que existe disponível uma solução formada por um conjunto de arranjos principais. Neste Capítulo chamamos de **arranjo principal** ao conjunto de caixas que formam o arranjo principal e também ao conjunto de caixas que foram alocadas no espaço residual de altura e no espaço residual de largura relacionados com o arranjo principal. Assim, podemos considerar uma proposta de solução para o problema de carregamento de contêiner como sendo formada por um conjunto de arranjos principais.

A ideia central da fase de melhoria local é aproveitar os melhores arranjos principais encontrados na fase construtiva e, após a seleção dos melhores arranjos principais, refazer o preenchimento das caixas restantes usando o AHC desenvolvido no Capítulo 4 após pequenas mudanças.

Supor que a fase construtiva terminou, gerando uma solução formada por p arranjos principais. Nesse contexto, a fase de melhoria local do algoritmo GRASP para o problema de carregamento de contêiner possui os seguintes passos:

1. Montar os dados de entrada constituídos por p arranjos principais, a taxa de ocupação de cada arranjo principal e as caixas alocadas em cada arranjo principal. Escolher uma taxa mínima t_{min} para preservar um arranjo principal.

2. Montar uma solução parcial formada pelos arranjos principais gerados na fase construtiva e que apresentam uma taxa de ocupação maior que t_{min}.

3. Todos os outros arranjos são desfeitos e, deve-se recalcular o espaço em profundidade disponível, assim como os tipos de caixas não alocadas, a quantidade e o volume total de cada tipo de caixa.

4. Com as caixas remanescentes é encontrada uma nova solução usando o AHC do Capítulo 4 sendo que na geração dos novos arranjos principais devem ser geradas cinco soluções diferentes e preservar a melhor delas. Cada uma dessas cinco soluções são geradas escolhendo as melhores alternativas no preenchimento dos espaços residuais de largura e de altura.

5.5 Detalhes de Implementação Computacional

O algoritmo GRASP proposto foi testado para caixas heterogêneas e um único container e usando as mesmas instâncias utilizadas nos testes do AHC apresentado no Capítulo anterior. O algoritmo foi programado em Fortran e os testes foram feitos em um PC Dell Processador Core 2 duo/2GB e memória RAM 2GB, com sistema operacional Windows XP.

Os resultados obtidos com o algoritmo GRASP, foram testados com o parâmetro α assumindo os valores $\alpha = 0.5$ e $\alpha = 0.8$. Como critério de parada, caso não ocorra o preenchimento total das caixas dentro do contêiner, estabelecemos o número máximo de iterações determinado para fins de testes de 1000 iterações. Assim, a cada iteração executada pelo algoritmo é registrada a solução que apresente, a melhor porcentagem de espaco ocupado no contêiner.

5.6 Testes Usando o Algoritmo GRASP

O algoritmo GRASP proposto foi testado nas 13 instâncias, usadas nos testes do AHC (apresentadas em 4.4): DA1, ... , DA9, GR, Ro1, Ro2 e Pis.

Na Tabela 34, apresentam-se os resultados do algoritmo GRASP desenvolvido neste trabalho e para um valor de $\alpha = 0,8$, para as instâncias DA1, DA2, DA3 e DA4. Adicionalmente, a Tabela 35 mostram-se os mesmos resultados, mas usando um valor de $\alpha = 0,5$.

Nas Tabelas 36 e 37 são mostrados os resultados encontrados pelo algoritmo GRASP para as instâncias DA5, DA6, DA7 e DA8, para os valores de $\alpha = 0,8$ e $\alpha = 0,5$, respectivamente.

Nas Tabelas 38 e 39 são mostrados os resultados encontrados pelo algoritmo GRASP desenvolvido neste trabalho para as instâncias DA9 e Pis, para os valores de $\alpha = 0,8$ e $\alpha = 0,5$.

Finalmente, as Tabelas 40 e 41 apresentam os resultados encontrados pelo algoritmo GRASP, para as duas instâncias apresentadas em Rodrigues (2005) (Ro1 e Ro2) e para valores de $\alpha = 0,8$ e $\alpha = 0,5$.

Tabela 34: Resultados do GRASP com $\alpha = 0.8$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas								Tempo (seg.)
				1	2	3	4	5	6	7	8	
DA1	0	91,11	2337	0	0	0	0	0	0	0	0	0,96
DA2	0	92,49	106	0	0	0	0	0	0	0	0	0,38
DA3	0	93,97	277	0	0	0	0	0	0	0	0	0,80
DA4	94	95,28	31	1	26	17	35	0	9	3	3	0,19
GR	0	91,74	117	0	0	0	0	0	0	0	0	0,37

Fonte: Informações da pesquisa da autora.

Tabela 35: Resultados do GRASP com $\alpha = 0.5$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas								Tempo (seg.)
				1	2	3	4	5	6	7	8	
DA1	0	91,11	2337	0	0	0	0	0	0	0	0	0,87
DA2	0	93,70	179	0	0	0	0	0	0	0	0	0,31
DA3	0	93,97	277	0	0	0	0	0	0	0	0	0,72
DA4	93	95,46	3	1	32	33	20	0	3	1	3	0,17
GR	0	92,39	157	0	0	0	0	0	0	0	0	0,34

Fonte: Informações da pesquisa da autora.

Tabela 36: Resultados do GRASP com $\alpha = 0.8$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas												Tempo (seg.)
				1	2	3	4	5	6	7	8	9	10	11	12	
DA5	0	93,51	582	0	0	0	0	0	0	0	0	0	0	0	0	1,54
DA6	0	93,47	1	0	0	0	0	0	0	0	0	0	0	0	0	0,38
DA7	0	92,86	545	0	0	0	0	0	0	0	0	0	0	0	0	1,80
DA8	148	95,11	16	16	3	9	2	3	13	6	30	25	3	30	8	0,14

Fonte: Informações da pesquisa da autora.

5.7 Análise Crítica dos Resultados Encontrados

O algoritmo GRASP proposto demonstra que se trata de uma técnica bastante eficiente para a resolução do PCC, melhorando todos os testes realizados, em que o percentual de volume ocupado é

Tabela 37: Resultados do GRASP com $\alpha = 0.5$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas												Tempo (seg.)
				1	2	3	4	5	6	7	8	9	10	11	12	
DA5	0	94,55	639	0	0	0	0	0	0	0	0	0	0	0	0	1,20
DA6	0	93,61	9	0	0	0	0	0	0	0	0	0	0	0	0	0,38
DA7	0	93,59	586	0	0	0	0	0	0	0	0	0	0	0	0	1,68
DA8	172	95,74	16	30	3	2	2	3	20	12	30	25	16	21	8	0,15

Fonte: Informações da pesquisa da autora.

Tabela 38: Resultados do GRASP com $\alpha = 0.8$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas										Tem. (seg.)
				1	...	4	...	10	...	12	13	...	20	
DA9	0	92,87	750	0	...	0	...	0	...	0	0	...	0	2,84
Pis	4	87,98	1	1	...	1	...	1	...	1	0	...	0	0,69

Fonte: Informações da pesquisa da autora.

Tabela 39: Resultados do GRASP com $\alpha = 0.5$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Compr. resid. (mm)	Tipo de caixas não alocadas											Tem. (seg.)
				1	...	4	...	10	...	14	...	17	...	20	
DA9	0	92,00	702	0	...	0	...	0	...	0	0	0	...	0	2,32
Pis	4	88,17	2	0	...	1	...	1	...	1	0	1	...	0	0,73

Fonte: Informações da pesquisa da autora.

Tabela 40: Resultados do GRASP com $\alpha = 0.8$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas							Tempo (seg.)
				1	2	3	4	5	6	7	
Ro1	20	90,47	28	2	0	2	6	6	2	2	0,10
Ro2	103	91,57	80	0	0	13	9	34	43	4	0,06

Fonte: Informações da pesquisa da autora.

Tabela 41: Resultados do GRASP com $\alpha = 0.5$

Tipo	No. de caixas não alocadas	Taxa de ocupação (%)	Comprim. residual (mm)	Tipo de caixas não alocadas							Tempo
				1	2	3	4	5	6	7	(seg.)
Ro1	20	90,47	28	2	0	2	6	6	2	2	0,12
Ro2	138	92,53	80	0	28	8	9	34	43	16	0,06

Fonte: Informações da pesquisa da autora.

maior em relação aos resultados encontrados no AHC.

Segundo a análise do algoritmo GRASP proposto, e a comparação com os resultados da literatura especializada e os resultados do AHC, apresentam-se a seguir as conclusões para cada uma das instâncias:

- **DA1:** Instância gerada por Leite (2007) e o volume total das caixas (15,919m^3) é menor que o volume do contêiner em relação ao volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP proposto: 91,11% do percentual de volume ocupado (com 2337mm de espaço residual na profundidade) e todas as caixas foram alocadas. O GRASP demonstra ser mais eficiente principalmente, pelo espaço residual no comprimento do contêiner, sendo possível alocar qualquer tipo caixa de DA1. No trabalho de Leite (2007), não existe menção explícita de que exista espaço residual no comprimento do contêiner e a melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 67,98%, de percentual de volume ocupado e todas as caixas foram carregadas.

- **DA2:** Instância gerada por Leite (2007) e o volume total das caixas (26,593m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP proposto: 93,70% (com 179mm de espaço residual na profundidade do contêiner) e todas as caixas foram alocadas. O GRASP demonstra ser mais eficiente, sendo possível alocar mais caixas. A Melhor solução encontrada pelo algoritmo GRASP de (Leite (2007): 86,42% e deixando para fora 15 caixas não carregadas (12 caixas tipo 1, 2 caixas tipo 4 e 1 caixa tipo 5).

- **DA3:** Instância gerada por Leite (2007) e o volume total das caixas (26,205m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP proposto: 93,97% (com 277mm de espaço residual na profundidade do contêiner) e todas as caixas foram alocadas. A Melhor solução encontrada pelo algoritmo GRASP de Leite (2007) : 85,06% e todas as

caixas foram carregadas.

- **DA4:** Instância gerada por (LEITE, 2007) e o volume total das caixas (31,940m^3) é maior que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 95,46% (com 3mm de espaço residual na profundidade) e restando 93 caixas não alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 82,95% e restando 105 caixas não carregadas.

- **GR:** O teste com a instância de (GEORGE; ROBINSON, 1980) : Instância apresentada por (GEORGE; ROBINSON, 1980) e o volume total das caixas (26,324m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 92,39% (com 157mm de espaço residual na profundidade) e todas as caixas foram alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 89,99% e todas as caixas foram carregadas.

- **DA5:**Instância gerada por (LEITE, 2007) e o volume total das caixas (24,636m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 94,55% (com 639mm de espaço residual na profundidade) e todas as caixas foram alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 83,58% e todas as caixas foram carregadas.

- **DA6:** Instância gerada por (LEITE, 2007) e o volume total das caixas (27,117m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 93,61% (com 9mm de espaço residual na profundidade) e todas as caixas foram alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 88,87% e restando 30 caixas não carregadas.

- **DA7:** Instância gerada por (LEITE, 2007) e o volume total das caixas (24,637m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 93,59% (com 586mm de espaço residual na profundidade) e todas as caixas foram alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 84,97% e todas as caixas foram carregadas.

- **DA8:** Instância gerada por (LEITE, 2007) e o volume total das caixas (33,275m^3) é maior que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 95,74% (com 16mm de espaço residual na profundidade) e restando 172 caixas não alocadas. Melhor

solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 88,30% e restando 65 caixas não carregadas.

- **DA9:** Instância gerada por (LEITE, 2007) e o volume total das caixas (23,678m^3) é menor que o volume do contêiner (29,287m^3). Melhor solução encontrada pelo GRASP: 92,87% (com 750mm de espaço residual na profundidade) e todas as caixas foram alocadas. Melhor solução encontrada pelo algoritmo GRASP de (LEITE, 2007) : 81,90% e todas as caixas foram carregadas.

- **Pis:** Instância gerada por (PISINGER, 2002) e o volume total das caixas (28,664m^3) é menor que o volume do contêiner (31,211m^3). Melhor solução encontrada pelo GRASP: 88,17% (com 2mm de espaço residual na profundidade) e restando 4 caixas não alocadas. Melhor solução encontrada pelo algoritmo de (PISINGER, 2002): 90,79% e todas as caixas foram carregadas.

- **Ro1:** Instância gerada por (RODRIGUES, 2005) e o volume total das caixas (8,232m^3) é maior que o volume do contêiner (7,290m^3). Melhor solução encontrada pelo GRASP: 90,47% (com 28mm de espaço residual na profundidade) e restando 20 caixas não alocadas. (RODRIGUES, 2005), utiliza 4 funções objetivas, por isso não é possível uma comparação.

- **Ro2:** Instância gerada por (RODRIGUES, 2005) e o volume total das caixas (44,958m^3) é maior que o volume do contêiner (33,132m^3). Melhor solução encontrada pelo GRASP: 92,53% (com 80mm de espaço residual na profundidade) e restando 138 caixas não alocadas. (RODRIGUES, 2005), utiliza 4 funções objetivas, por isso não é possível uma comparação.

5.8 Resultados de Sensibilidade do Algoritmo GRASP

Na Tabela 42 apresentamos o melhor resultado do algoritmo GRASP desenvolvido neste trabalho, para α variando de $0,1$ a $0,9$ e o número de iterações fixa em 500, levando em consideração a taxa de ocupação de volume ocupado dentro do contêiner.

Na Tabela 43 apresentamos o melhor resultado do algoritmo GRASP desenvolvido neste trabalho, para α fixo em $0,2$ e o número de iterações variando de 500 a 100000.

Tabela 42: Resultados do GRASP para 500 iterações

	Valores de alfa									Taxa do
Instâncias	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	melhor alfa
DA1	91,11	91,11	90,58	90,30	91,11	91,11	91,11	91,11	86,86	91,11
DA2	92,37	93,14	92,94	92,44	92,44	92,49	92,64	92,49	91,30	93,14
DA3	93,68	93,97	93,97	93,97	93,85	93,93	93,97	93,97	93,97	93,97
DA4	95,80	95,73	95,63	95,69	95,47	95,47	95,28	95,28	92,89	95,80
GR	92,89	92,81	93,06	92,43	92,26	92,38	91,81	91,74	91,41	93,06
DA5	94,09	93,96	94,54	93,51	93,96	94,20	94,09	93,51	91,71	94,54
DA6	93,59	93,59	93,62	93,68	94,06	93,57	93,19	93,47	93,37	94,06
DA7	93,23	93,30	93,59	93,30	93,27	93,73	93,73	92,86	91,29	93,73
DA8	95,77	95,68	96,04	96,24	95,74	96,07	96,16	95,11	92,70	96,24
DA9	91,68	91,95	91,72	92,19	92,10	92,70	92,43	91,99	90,74	92,70
Pis	88,06	88,92	88,66	87,81	87,71	87,26	87,45	87,98	87,38	88,92
Ro1	90,47	88,86	90,47	90,47	90,47	90,47	90,47	90,47	90,47	90,47
Ro2	92,53	92,53	92,53	92,53	92,53	92,53	92,53	91,57	92,21	92,53

Fonte: Informações da pesquisa da autora.

Tabela 43: Resultados do GRASP para $\alpha = 0,2$

	Número de iterações					
Instâncias	500	1000	1500	2000	10000	100000
DA1	91,11	90,30	91,11	91,11	91,11	92,04
DA2	93,14	93,18	92,79	93,07	93,35	94,014
DA3	93,97	94,80	94,80	94,11	94,80	94,97
DA4	95,73	95,67	95,69	95,77	95,83	95,86
GR	92,81	93,07	93,07	93,07	93,07	93,74
DA5	93,96	94,57	93,95	93,95	94,55	94,79
DA6	93,59	93,57	93,61	94,21	93,85	94,41
DA7	93,30	93,48	93,97	93,77	94,41	94,41
DA8	95,68	95,90	95,77	95,98	96,48	96,42
DA9	91,95	92,34	92,34	92,50	92,85	93,22
Pis	88,92	88,63	88,93	88,93	88,93	89,09
Ro1	88,86	89,32	89,32	90,47	90,47	90,47
Ro2	92,53	92,53	92,53	92,53	92,53	92,53

Fonte: Informações da pesquisa da autora.

5.9 Conclusões Parciais

O algoritmo GRASP demonstra ser uma técnica eficiente na resolução do PCC e o algoritmo GRASP proposto, apresenta capacidade de encontrar soluções de melhor qualidade que outras pro-

postas de otimização do mesmo tipo. Os resultados do algoritmo GRASP com as 13 instâncias da literatura, podem ser considerados eficientes levando em conta a natureza desse tipo de problema. Deve-se observar que a metaheurística GRASP desenvolvida neste trabalho supera em todas as instâncias testadas ao algoritmo GRASP desenvolvida em Leite(2007). Na verdade em todas as instâncias testadas, o algoritmo GRASP apenas não é superior no teste realizado com a instância de Pisinger(2002) e, obviamente, comparado com o algoritmo de Pisinger(2002).

Os resultados apresentados mostram que é possível empacotar as caixas heterogêneas com uma boa porcentagem de volume ocupado dentro do contêiner, geralmente com mais de 90% de taxa de ocupação. Outro aspecto importante que pode ser mencionado é o tempo de processamento muito pequeno levando em conta de que estamos resolvendo um problema de planejamento de logística.

Dos resultados experimentais obtidos, conclui-se que é interessante a utilização do algoritmo GRASP especialmente para o caso em que existem poucos tipos de caixas e muitas caixas do mesmo tipo que favorece a lógica de formação de cubóides proposto como sendo muito importante neste trabalho. Pode-se observar que para problemas desse tipo, o algoritmo GRASP apresenta excelente desempenho. or outro lado, quando a instância testada apresenta muitos tipos de caixas e poucas caixas de cada tipo, problema chamado de fortemente heterogêneo, como acontece na instância de Pisinger(2002), então o algoritmo GRASP desenvolvido neste trabalho apresenta algumas limitações de desempenho para obter um resultado mais eficiente.

Capítulo 6

CONCLUSÕES

Neste trabalho foram desenvolvidas duas técnicas de otimização para o problema de carregamento de contêiner, isto é, um algoritmo heurístico construtivo e uma metaheurística GRASP. Os resultados encontrados se mostraram promissores, mas ainda existem várias alternativas de melhoria que podem ser implementadas, para trabalhos futuros, nessas propostas de otimização para melhorar o desempenho dessas estratégias de otimização.

No algoritmo heurístico construtivo foi retomada a estratégia de geração de arranjos em contraposição com da estratégia de geração de camadas que é muito mais popular e muito mais usada entre os pesquisadores que trabalham na otimização do problema de carregamento de contêiner. Um arranjo, na verdade é uma generalização do conceito de camada e, portanto, um arranjo pode ser formado por várias camadas. Complementando o conceito de arranjos também foi retomada a estratégia de geração de cubóides que se encontra muito pouco explorada na literatura especializada. Um cubóide é um paralelepipedo perfeito, mas formado por um conjunto de caixas. Assim, o preenchimento do contêiner proposto neste trabalho sugere inciar o preenchimento usando um cubóide (e portanto gerando um arranjo) em contraposição com as propostas de iniciar o preenchimento com uma caixa (e portanto gerando uma camada). Nesse contexto, concluimos que a lógica de geração de arranjos, iniciada com a formação de cuboides perfeitos representa uma alternativa de otimização promissora para resolver o problema de carregamento de contêiner e, portanto, representa a componente mais consistente do algoritmo heurístico construtivo.

Na proposta do AHC a parcela que deve ser melhorada de forma significativa em trabalhos futuros é a estratégia de preenchimento do espaço residual de altura e do espaço residual de largura do arranjo principal. Os testes experimentais mostram que a estratégia apresentada neste trabalho para essa parcela do algoritmo de otimização não funciona de forma eficiente. Os testes experimentais mostram

também um desempenho pouco eficiente da montagem da última camada. Assim, essa parcela do algoritmo heurístico construtivo pode ser melhorada de forma significativa.

O algoritmo heurístico construtivo, mesmo com as deficiências mencionadas anteriormente, se mostrou relativamente eficiente. Assim, esse algoritmo encontrou melhores resultados que o algoritmo GRASP desenvolvido por Leite (2007) para as instâncias da série DA e para a instância de George e Robinson (1980). Esse algoritmo também encontrou melhores resultados que os encontrados pelo algoritmo génetico desenvolvido por Rofrigues (2006). Entretanto, o resultado de teste com a instância de Pisinger (2002) foi significativamente inferior que o encontrado.

A metaheurística GRASP desenvolvido neste trabalho também apresentou resultados promissores. Esse desempenho pode ser decorrente da qualidade do algoritmo heurístico construtivo usado na fase construtiva. Assim, o GRASP incorpora as qualidades e os defeitos do algoritmo heurístico construtivo usado como base na fase construtiva. Esse algoritmo pode ainda ser melhorado incorporando uma fase de melhoria global em que seriam juntados arranjos de excelente qualidade existentes em soluções consideradas de elite e terminando o preenchimento usando uma heurística construtiva eficiente. Entretanto, esta implementação adicional, assim como uma implementação mais eficiente de outras parcelas do algoritmo GRASP passa por uma implementação de uma melhoria do algoritmo heurítico construtivo. Portanto, em trabalhos futuros, a tarefa inicial e mais importante é implemnetar modificações adicionais no algoritmo heurístico desenvolvido neste trabalho.

A metaheurística GRASP apresentou melhores resultados que o algoritmo heurístico construtivo desenvolvido neste trabalho como era esperado. Portanto, comparado com os resultados encontrados por Leite (2007) e Rodrigues (2005), a metaheurística GRASP apresentou um desempenho superior. Entretanto, o algoritmo GRASP não encontrou a solução encontrada por Pisinger (2002). Deve-se observar que a instância de Pisinger (2002) é altamente heterogênea e, portanto, não favorece a estratégia de geração de cubóides apresentada neste trabalho como parte do algoritmo heurístico construtivo. Adicionalmente, Pisinger (2002) apresenta uma excelente estratégia para o preenchimento do espaço residual de altura e do espaço residual de largura.

REFERÊNCIAS

BORTFELDT, A.; GEHRING, H. **A hybrid genetic algorithm for the container loading problem**. European Journal of Operational Research, Amsterdam, v. 131, n. 1, p. 143-161, 2001.

BORTFELDT, A.; GEHRING, H. **A parallel tabu search algorithm for solving the container loading problem**. Parallel Computing, Amsterdam, v. 29, n. 5, p. 641-662, 2003.

CECILIO, F. O. **Heurísticas para o problema de carregamento de carga dentro de contêineres**. 2003. 112 f. Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal de São Carlos - UFSCAR, São Carlos, 2003.

CHEN, C. S.; LEE, S. M.; SHEN, Q. S. **An analytical model for the container loading problem**. European Journal of Operational Research, Amsterdam, v. 80, n. 1, p. 68-76, 1995.

FEO, T. A.; RESENDE, M. G. C. **A probalistic heuristic for a computationally dificult set convering problem**. Operations Research Letters, Amsterdam, v. 8, p. 67-71, 1989.

FEO, T. A.; RESENDE, M. G. C. **Greedy randomized adaptive search procedures**. Journal of Global Optimization, New York, v. 6, p. 109-133, 1995.

GARFINKEL, R. S.; NEMHAUSER, G. L. **Integer programming**. New York: John Wiley & Sons, 1972.

GEHRING, H.; BORTFELDT, A. **A genetic algorithm for solving the container loading problem**. International Transactions in Operations Research, Oxford, v. 4, n. 516, p. 401-418, 1997.

GEHRING, H.; MENSCHNER, K.; MEYER, M. **A computer based heuristic for packing pooled shipment container**. European Journal of Operational Research, Amsterdam, v. 44, n. 2, p. 277-288, 1990.

GEOFFRION, A. M. **Integer programming by implicit enumeration and balas method**. SIAM Review, Philadelphia, v. 9, n. 2, p. 178-190, 1967.

GEORGE, J. A.; ROBINSON, D. F. **A heuristic for packing boxes into a container**. Computers and Operational Research, Oxford, v. 7, n. 3, p. 147-156, 1980.

GLOVER, F.; KOCHENBERGER, G. (Eds.). **Handbook of metaheuristics**. Boston: Kluwer Academic, 2003.

GOLDBERG, D. E. **Genetic algorithms in search, optimization and machine learning**. Reading: Addison Wesley, 1989.

HE, D, Y.; CHA, J. Z. **Research on solution to complex container loafing problem based on genetic algorithm**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS, 1., 2002, Beijijg. Proceedings New York: IEEE, 2002. v. 1, p. 78-82.

LEITE, E. C. B. R. **GRASP para otimizar o carregamento de um contêiner**. 2007. 73 f. Dissertação (Mestrado em Engenharia de Produção) - Universidade Estadual do Norte Fluminense - UENF, Campos dos Goytacazes, 2007.

LI, H. L.; TSAI, J. F.; HU, N. Z. **A distributed global optimization method for packing problems**. Journal of the Operational Research Society, Tokyo, v. 54, n. 4, p. 419-425, 2003.

LIU, F. F. E.; HISIAO, C. **A three-dimensional pallet loading method for single-size boxes**. Journal of the Operational Research Society, Tokyo, v. 48, p. 726-735, 1997.

MARTELLO, S.; PISINGER, D.; VIGO, D. **The three-dimensional Bin packing problem**. Operations Research, Hanover, v. 48, n. 2, p. 256-267, 2000.

MICHALEWICZ, Z. **Genetic algorithms + data structure = evolution programs**. 2. ed. Berlin: Springer-Verlag, 1994.

MORALES, S. R.; MORABITO, E. R.; WIDMER, J. A. **Otimização do carregamento de produtos paletizados em caminhões**. Gestão e Produção, São Carlos, v. 4, v. 2, p. 234-250, 1997.

MORABITO, R.; ARENALES, M. N. **Abordagens para o problema de carregamento de contêiners**. Pesquisa Operacional, Rio de Janeiro, v. 17, n. 1, p. 29-56, 1997.

MORABITO, R.; ARENALES, M. **An, and/or graph approach to the container loading problem**. International Transactions in Operational Research, Oxford, v. 1, n. 1, p. 59-73, 1994.

MOURA, A.; OLIVEIRA, J. F. **A grasp approach to the container-loading problem**. IEEE Intelligent Systems, New York, v. 20, n. 4, p. 50-57, 2005.

PISINGER, D. **Heuristics for the container loading problem**. European Journal of Operational Research, Amsterdam, v. 141, n. 2, p. 382-392, 2002.

RAIDL, G. R. **A weight-coded genetic algorithm for the multiple container packing problem**. In: ACM SYMPOSIUM ON APLLIED COMPUTING, 14., 1999, San Antonio. Proceedings San Antonio: SAC, 1999. v. 1, p. 596-603.

RODRIGUES, L. L. **Um algoritmo genético para solução de carregamento de container**. 2005. 105 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Rio de Janeiro - UFRJ, Rio de Janeiro, 2005.

SILVA, J. L. C. S.; SOMA, N. Y. **Um algoritmo polinomial para o problema de empacotamento de contêineres com estabilidade estática da carga**. Pesquisa Operacional, Rio de Janeiro, v. 23, n. 1, p. 79-98, 2003.

VENDRAMINI, E. **Otimização do problema de carregamento de container usando uma metaheurística eficiente**. 2007. 103 f. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia, Universidade Estadual Paulista, Ilha Solteira, 2007.

Apêndice A

DADOS DOS SISTEMAS TESTADOS

Tipos de contêiner:

- Contêiner tipo I:

  Neste caso temos um contêiner de 5793 mm de comprimento, 2236 mm de largura e 2261 mm de altura e, portanto, com um volume de 29,287 m^3.

- Contêiner tipo II:

  Neste caso temos um contêiner de 5000 mm de comprimento, 1080 mm de largura e 1350 mm de altura e, portanto, com um volume de 7,290 m^3.

- Contêiner tipo III:

  Neste caso temos um contêiner de 5900 mm de comprimento, 2300 mm de largura e 2300 mm de altura e, portanto, com um volume de 31,211 m^3.

- Contêiner tipo IV:

  Neste caso temos um contêiner de 5899 mm de comprimento, 2388 mm de largura e 2352 mm de altura e, portanto, com um volume de 33,13 m^3.

Tabela 44: Dados do exemplo de George e Robinson (1980)

i	c_i	l_i	h_i	q_i
	(mm)	(mm)	(mm)	
1	785	139	273	400
2	901	185	195	160
3	901	195	265	40
4	1477	135	195	40
5	614	480	185	8
6	400	400	135	16
7	264	400	400	80
8	385	365	290	40

Fonte: George e Robinson (1980).
Total de caixas: 784 Volume total: 26,323824 m^3.
Contêiner tipo I.

Tabela 45: Dados do exemplo de Rodrigues (2005)

i	c_i	l_i	h_i	q_i	p_i	v_i
	(mm)	(mm)	(mm)		(kg)	R$
1	477	460	360	20	14,1	1253
2	419	563	358	25	9,6	626
3	419	563	358	20	10,0	704
4	419	563	358	15	10,6	791
5	419	563	358	10	10,6	1018
6	456	473	382	5	13,7	1488
7	445	441	335	5	10,7	1096

Fonte: Rodrigues (2005).
Total de caixas: 100 Volume total: 8,232071m^3.
Contêiner tipo II.

Tabela 46: Dados do exemplo de Rodrigues (2005)

i	c_i	l_i	h_i	q_i	p_i	v_i
	(mm)	**(mm)**	**(mm)**		**(kg)**	**R$**
1	**500**	**382**	**183**	**55**	**14,1**	**1253**
2	**500**	**382**	**156**	**30**	**9,6**	**626**
3	**593**	**869**	**696**	**68**	**10,0**	**704**
4	**576**	**898**	**696**	**26**	**10,6**	**791**
5	**563**	**358**	**419**	**45**	**10,6**	**1018**
6	**563**	**358**	**419**	**43**	**13,7**	**1488**
7	**356**	**358**	**419**	**18**	**10,7**	**1096**

Fonte: Rodrigues (2005).
Total de caixas: 285 Volume total: 44,958261 m^3.
Contêiner tipo IV.

Tabela 47: Conjunto DA1 - Dados aleatórios com 306 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**334**	**248**	**299**	**75**
2	**1033**	**300**	**234**	**18**
3	**1479**	**104**	**114**	**34**
4	**1372**	**349**	**215**	**2**
5	**722**	**275**	**228**	**33**
6	**1430**	**197**	**294**	**30**
7	**730**	**355**	**267**	**53**
8	**1423**	**299**	**166**	**61**

Fonte: Leite (2007).
Total de caixas: 306 Volume total: 15,919 m^3.
Contêiner tipo I.

Tabela 48: Conjunto DA2 - Dados aleatórios com 453 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**820**	**282**	**366**	**92**
2	**1332**	**179**	**209**	**19**
3	**1404**	**168**	**191**	**31**
4	**913**	**329**	**320**	**12**
5	**838**	**267**	**340**	**43**
6	**495**	**323**	**339**	**140**
7	**428**	**280**	**386**	**89**
8	**356**	**347**	**100**	**27**

Fonte: Leite (2007).
Total de caixas: 453 Volume total: 26,5929 m^3.
Contêiner tipo I.

Tabela 49: Conjunto DA3 - Dados aleatórios com 679 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**912**	**152**	**192**	**53**
2	**961**	**321**	**337**	**20**
3	**367**	**372**	**355**	**153**
4	**378**	**312**	**114**	**73**
5	**1250**	**138**	**170**	**68**
6	**846**	**147**	**199**	**139**
7	**1198**	**210**	**222**	**101**
8	**1333**	**200**	**169**	**72**

Fonte: Leite (2007).
Total de caixas: 679 Volume total: 26,205 m^3.
Contêiner tipo I.

Tabela 50: Conjunto DA4 - Dados aleatórios com 471 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**1081**	**356**	**339**	**97**
2	**590**	**182**	**390**	**68**
3	**457**	**386**	**238**	**42**
4	**1228**	**105**	**257**	**58**
5	**1324**	**107**	**112**	**18**
6	**705**	**373**	**287**	**79**
7	**505**	**197**	**133**	**26**
8	**983**	**264**	**286**	**83**

Fonte: Leite (2007).
Total de caixas: 471 Volume total: 31,940 m^3.
Contêiner tipo I.

Tabela 51: Conjunto DA5 - Dados aleatórios com 614 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**362**	**132**	**119**	**78**
2	**1188**	**227**	**287**	**88**
3	**1249**	**141**	**219**	**12**
4	**391**	**252**	**226**	**34**
5	**388**	**251**	**211**	**7**
6	**819**	**173**	**215**	**53**
7	**368**	**130**	**262**	**7**
8	**1299**	**247**	**231**	**100**
9	**985**	**248**	**290**	**36**
10	**1191**	**131**	**221**	**48**
11	**516**	**174**	**170**	**94**
12	**642**	**122**	**283**	**57**

Fonte: Leite (2007).
Total de caixas: 614 Volume total: 24,636 m^3.
Contêiner tipo I.

Tabela 52: Conjunto DA6 - Dados aleatórios com 785 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**897**	**249**	**231**	**100**
2	**1203**	**247**	**190**	**50**
3	**882**	**238**	**122**	**75**
4	**813**	**141**	**250**	**59**
5	**476**	**248**	**265**	**6**
6	**1120**	**107**	**164**	**70**
7	**777**	**214**	**285**	**77**
8	**937**	**287**	**147**	**82**
9	**1231**	**113**	**168**	**49**
10	**841**	**150**	**274**	**99**
11	**433**	**228**	**158**	**74**
12	**630**	**225**	**257**	**44**

Fonte: Leite (2007).
Total de caixas: 785 Volume total: 27,372 m^3.
Contêiner tipo I.

Tabela 53: Conjunto DA7 - Dados aleatórios com 661 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**807**	**203**	**398**	**28**
2	**1245**	**380**	**192**	**79**
3	**816**	**179**	**392**	**38**
4	**672**	**211**	**235**	**80**
5	**863**	**146**	**145**	**96**
6	**358**	**346**	**252**	**6**
7	**745**	**114**	**132**	**72**
8	**396**	**181**	**141**	**37**
9	**349**	**331**	**146**	**53**
10	**549**	**237**	**148**	**41**
11	**472**	**310**	**357**	**100**
12	**728**	**113**	**300**	**31**

Fonte: Leite (2007).
Total de caixas: 661 Volume total: 24,637 m^3.
Contêiner tipo I.

Tabela 54: Conjunto DA8 - Dados aleatórios com 458 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**814**	**102**	**192**	**32**
2	**534**	**104**	**155**	**13**
3	**1040**	**399**	**384**	**79**
4	**1288**	**368**	**368**	**73**
5	**443**	**181**	**385**	**17**
6	**442**	**208**	**389**	**20**
7	**881**	**181**	**188**	**12**
8	**1044**	**286**	**113**	**48**
9	**323**	**365**	**222**	**32**
10	**721**	**214**	**241**	**29**
11	**303**	**255**	**295**	**86**
12	**654**	**109**	**124**	**17**

Fonte: Leite (2007).
Total de caixas: 458 Volume total: 33,275 m^3.
Contêiner tipo I.

Tabela 55: Conjunto DA9 - Dados aleatórios com 930 caixas

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**696**	**299**	**107**	**10**
2	**384**	**139**	**208**	**94**
3	**1139**	**231**	**154**	**49**
4	**886**	**127**	**284**	**52**
5	**773**	**155**	**204**	**95**
6	**1057**	**204**	**206**	**2**
7	**1298**	**213**	**128**	**43**
8	**501**	**213**	**196**	**86**
9	**1260**	**154**	**205**	**9**
10	**1254**	**236**	**157**	**12**
11	**1241**	**101**	**173**	**2**
12	**935**	**282**	**211**	**53**
13	**542**	**245**	**294**	**69**
14	**594**	**134**	**220**	**98**
15	**914**	**101**	**199**	**30**
16	**733**	**213**	**151**	**57**
17	**634**	**120**	**105**	**59**
18	**531**	**108**	**261**	**17**
19	**427**	**252**	**216**	**85**
20	**664**	**206**	**109**	**8**

Fonte: Leite (2007).
Total de caixas: 930 Volume total: 23,678 m^3.
Contêiner tipo I.

Tabela 56: Dados do exemplo de Pisinger (2002).

i	c_i	l_i	h_i	q_i
	(mm)	**(mm)**	**(mm)**	
1	**850**	**600**	**260**	**5**
2	**360**	**690**	**280**	**7**
3	**680**	**260**	**460**	**6**
4	**710**	**1000**	**1000**	**5**
5	**280**	**310**	**420**	**6**
6	**1090**	**290**	**510**	**8**
7	**940**	**520**	**430**	**9**
8	**320**	**1060**	**1130**	**6**
9	**520**	**1150**	**450**	**13**
10	**290**	**680**	**440**	**9**
11	**420**	**410**	**1150**	**12**
12	**970**	**590**	**480**	**8**
13	**1060**	**530**	**400**	**5**
14	**1050**	**680**	**320**	**3**
15	**1070**	**430**	**1100**	**3**
16	**390**	**620**	**800**	**6**
17	**290**	**560**	**730**	**7**
18	**310**	**470**	**420**	**10**
19	**390**	**410**	**1140**	**8**
20	**1010**	**380**	**400**	**10**

Fonte: Pisinger (2002).
Total de caixas: 146 Volume total: 28,596342 m^3.
Contêiner tipo III.