GRASP PARA OTIMIZAR O CARREGAMENTO DE UM CONTÊINER

ETELVIRA CRISTINA BARRETO RANGEL LEITE

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE - UENF

CAMPOS DOS GOYTACAZES - RJ

MAIO DE 2007

GRASP PARA OTIMIZAR O CARREGAMENTO DE UM CONTÊINER

ETELVIRA CRISTINA BARRETO RANGEL LEITE

"Dissertação apresentada ao Centro de Ciência e Tecnologia, da Universidade Estadual do Norte Fluminense, como parte das exigências para obtenção do título de Mestre em Engenharia de Produção".

Orientador: Prof. Geraldo Galdino de Paula Júnior

CAMPOS DOS GOYTACAZES - RJ
MAIO DE 2007

GRASP PARA OTIMIZAR O CARREGAMENTO DE UM CONTÊINER

ETELVIRA CRISTINA BARRETO RANGEL LEITE

"Dissertação apresentada ao Centro de Ciência e Tecnologia, da Universidade Estadual do Norte Fluminense, como parte das exigências para obtenção do título de Mestre em Engenharia de Produção".

Aprovada em 25 de maio de 2007.

Comissão Examinadora:

_____
Prof. **Euclydes Vieira Neto** (ISECENSA)

_____
Prof. **Carlos Leonardo Ramos Póvoa** (UENF)

_____
Prof. **Rodrigo Tavares Nogueira** (UENF)

_____
Prof. **Geraldo Galdino de Paula Júnior** (UENF) – Orientador

FICHA CATALOGRÁFICA

Preparada pela Biblioteca do **CCT** / **UENF** **05/2008**

Leite, Etelvira Cristina Barreto Rangel
GRASP para otimizar o carregamento de um contêiner / Etelvira Cristina Barreto Rangel Leite. – Campos dos Goytacazes, 2007.
xi, 73 f. : il.
Dissertação (Mestrado em Engenharia de Produção) -- Universidade Estadual do Norte Fluminense Darcy Ribeiro. Centro de Ciência e Tecnologia. Laboratório de Engenharia de Produção. Campos dos Goytacazes, 2007.
Orientador: Geraldo Galdino de Paula Júnior.
Área de concentração: Pesquisa operacional
Bibliografia: f. 55-58
1. Corte e empacotamento 2. Contêiner 3. Otimização 4. GRASP I. Universidade Estadual do Norte Fluminense Darcy Ribeiro. Centro de Ciência e Tecnologia. Laboratório de Engenharia de Produção II. Título

CDD 658.4034

DEDICATÓRIA

Meu esposo Francismário,
meus filhos Daniel e Lucas,
e a meus pais.

AGRADECIMENTOS

Ao Senhor, meu Deus, pela oportunidade que me concede de alcançar mais uma vitória em minha vida.

> "*Mas para mim, bom é aproximar-me de Deus; ponho a minha confiança no Senhor Deus, para anunciar todas as suas obras*." [Sal 73:28]

> *"Porque dEle, e por meio dEle, e para Ele são todas as coisas. A Ele, pois, a glória eternamente. Amém."* [Rom 11:36]

Ao Professor Geraldo Galdino de Paula Júnior pela confiança depositada no desenvolvimento deste projeto e pela amizade que compartilhamos.

A minha amiga Cibelle Degel pelo apoio e confiança em mim.

Ao meu esposo, aos meus filhos e aos meus pais pelo amor, carinho, paciência, apoio incondicional, incentivo e presença durante todos esses anos.

A toda minha família pelo apoio e palavras de incentivo.

E finalmente a todos que, direta ou indiretamente, contribuíram na realização desta dissertação.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CEAC	Cortes e Empacotamento Assistidos por Computador
DDR	*Double Data Rating*
GLS	*Guided Local Search*
GRASP	*Greedy Randomized Adaptive Search Procedure*
INPE	Instituto Nacional de Pesquisas Espaciais
ISO	*International Standardization Organization*
ITA	Instituto Tecnológico de Aeronáutica
LRC	Lista Restrita de Candidatos
RAM	*Random Access Memory*
UENF	Universidade Estadual do Norte Fluminense
UFSCAR	Universidade Federal de São Carlos
USP	Universidade de São Paulo

RESUMO

Otimizar a ocupação de um espaço tridimensional semelhante a um contêiner pode ser visto como um caso especial do problema de corte. Desta forma esse problema pode ser entendido como aquele em que pequenas unidades são empacotadas em uma unidade maior. Além disso, as operações de empacotamento podem ser modeladas como um problema de otimização combinatória onde o ótimo de uma função objetivo é procurado. Os itens requeridos podem ser combinados dentro de um contêiner de várias maneiras, sendo respeitado um grupo de restrições semelhantes àquelas no processo de corte. Essas combinações são chamadas planos de corte. O número de possíveis planos de corte é, na prática, muito alto, demandando técnicas bem elaboradas para determinar o plano ótimo. Além de tratar com os principais fatores que contribuem para otimizar a ocupação de um espaço tridimensional, esta dissertação enumera as principais características para a classificação do problema, bem como as restrições-chave com ele associadas. Do lado da solução a pesquisa aplica um estudo baseado no algoritmo GRASP para procurar a melhor forma de preencher um contêiner. Os principais fatores que influenciaram a GRASP foram a cardinalidade da lista de candidatos, a função gulosa e o procedimento de busca local. O algoritmo proposto foi testado através de um software desenvolvido especificamente para os experimentos computacionais contidos nesta dissertação, e os resultados obtidos foram relatados junto com sua análise.

Palavras-chave: Corte e Empacotamento, Contêiner, Otimização, GRASP.

ABSTRACT

Optimizing the occupation of a three-dimensional space like a container can be seen as a special case of the cutting stock problem. In this way this problem may be viewed as one in wich smaller units are packed into a larger one. Moreover, the packaging operations can be modeled as a combinatorial optimization problem where the optimum of an objective function is sought. The items requested can be combined within the container in countless ways, being respected a group of constraints like those in the cutting stock process. These combinations are called cutting plans. The number of possible cutting plans is, in practice, very high, demanding techniques well elaborated to determine the optimum plan. Besides dealing with the main factors that contributed to optimizing the occupation of a three-dimensional space, this dissertation enumerates the main characteristics for the classification of the problem as well as the physical restriction-keys associated with it. On the solution side the research applies a study based on the GRASP algorithm to seek the best possible completion of a container. The main factors influencing the GRASP were the cardinality of the candidates' list, the greedy function and the procedure for local search. The proposed algorithm was tested through a software developed specifically to the computational experiments contained in this dissertation and the results obtained were reported along with its analysis.

Keywords: Cut and Packing, Container, Optimization, GRASP.

1 – INTRODUÇÃO

1.1 – Problema da Pesquisa

A otimização da ocupação de espaços tridimensionais pode ser vista como um caso especial dos problemas de corte e empacotamento (MORABITO; ARENALES, 1994), fator importante nas atividades logísticas de movimentação, armazenagem e transporte de produtos, implicando, por exemplo, a redução de custos destas atividades, bem como a redução do tempo de carregamento e descarregamento de contêineres. Segundo Cecílio e Morabito (2004), na prática, o problema aparece em dois casos: (*i*) quando uma combinação de contêineres deve ser escolhida para transportar uma dada carga, e (*ii*) quando o maior volume de uma dada carga deve ser escolhido para ser transportado em um único contêiner.

Numa indústria, cortes e empacotamento constituem componentes importantes na formação do custo final dos produtos; logo, qualquer redução de custos é sempre bem-vinda para as empresas e indústrias. Porém, trata-se de um problema de otimização combinatória NP-difícil que, na prática, é extremamente difícil de ser resolvido por algoritmos exatos para a obtenção da solução ótima, uma vez que esta requer muito tempo de processamento computacional. Para contornar essa restrição de tempo utilizam-se heurísticas, que nem sempre garantem a solução ótima, mas, provavelmente, garantam solução viável e de boa qualidade, com baixo esforço computacional, conforme destacam Constantino e Gomes Júnior (2002).

Problemas de natureza combinatória na área de corte e empacotamento são objetos de pesquisa desde a década de 60, como se pode constatar em estudos de Gilmore e Gomory (1961). Também Poldi e Arenales (2003) chamam a atenção para o fato de que a importância técnica e econômica das aplicações práticas e a dificuldade de desenvolver métodos de soluções eficazes que garantam poder encontrar boas soluções em tempos computacionais razoáveis têm motivado muitas pesquisas na área.

Verifica-se que, contrariamente ao Empacotamento Unidimensional e ao Bidimensional, o problema de Empacotamento Tridimensional começou a ser pesquisado, mais intensivamente, há pouco mais de dez anos. Conforme especificado em Silva e Soma (2003), há quantidade restrita de trabalhos publicados relatando tais problemas de empacotamentos tridimensionais.

A problemática desta dissertação tem como objeto de pesquisa a otimização dos processos industriais para ocupação de espaços tridimensionais, usando uma variação da metodologia de empacotamento tridimensional de base retangular. A questão é: como arranjar o maior volume possível de caixas, de baixa densidade, dentro de um único contêiner, de modo que a perda em forma de não preenchimento seja mínima?

1.2 – Classificação dos Problemas de Corte

Problemas de Corte e Empacotamento aparecem na literatura sob uma terminologia bastante diferenciada como, por exemplo, problema de corte de estoque, problema de empacotamento de bins, problema de carregamento de paletes, contêineres, etc. Tais problemas, segundo Poldi e Arenales (2003), podem ser classificados de acordo com as dimensões relevantes do objeto a ser cortado.

Segundo a tipologia de Dyckhoff (1990), os problemas de corte e empacotamento são classificados conforme as características geométricas e combinatórias dos objetos, procurando-se integrar as várias características a fim de facilitar o relacionamento entre elas. As principais características e seus respectivos tipos, mais comumente citados, são denotados com símbolos.

No estabelecimento da tipologia, as características foram agrupadas em quatro critérios básicos, descritos a seguir, usados para classificar os problemas:

1. Dimensionalidade:

Está relacionada ao número de dimensões do objeto que são relevantes durante o processo de corte, tais como:

- (1) o problema é unidimensional, quando apenas uma das dimensões do objeto é relevante no processo do corte. Exemplo: corte de barras de alumínio, aço, etc.

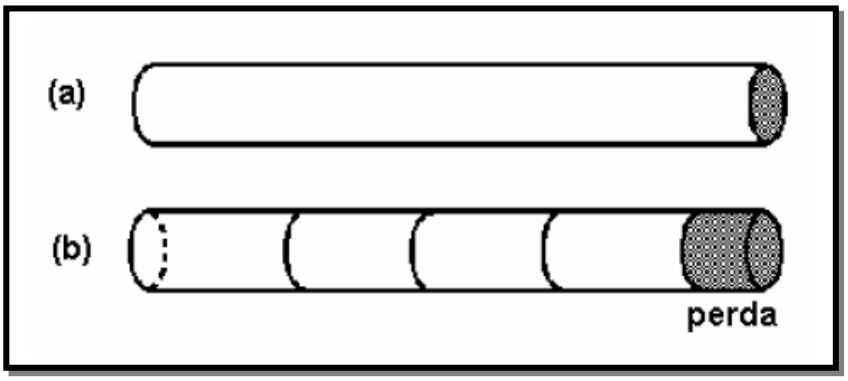

Figura 1 – (a) Objeto (barra) a ser cortado; (b) Objeto cortado produzindo 4 itens e uma perda.
Fonte: POLDI e ARENALES. 2003

- (2) o problema é bidimensional, quando duas dimensões do objeto são relevantes no processo de corte. Exemplo: corte de chapas retangulares de madeira, chapas de aço, etc.

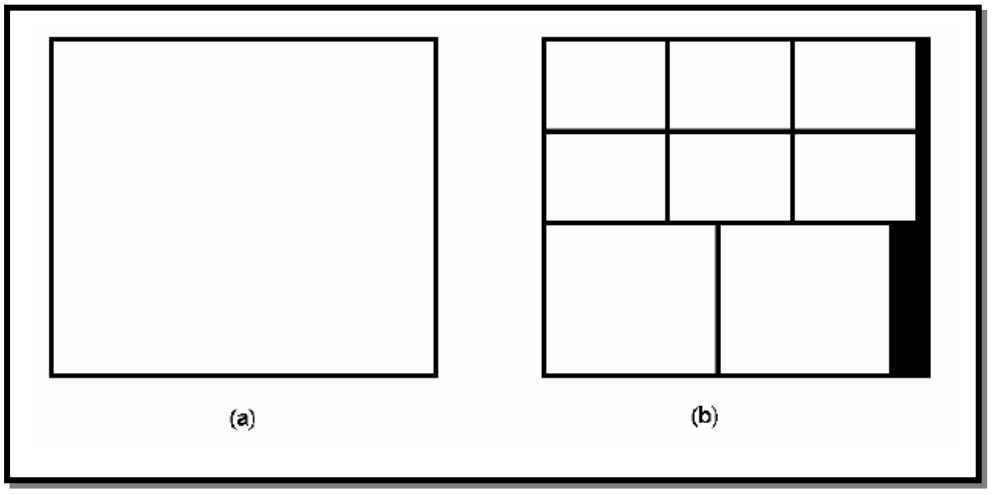

Figura 2 – (a) Objeto (chapa) a ser cortado; (b) Objeto cortado.
Fonte: POLDI e ARENALES, 2003

- (3) o problema é tridimensional, quando três dimensões do objeto são relevantes no processo do corte. Exemplo: corte de espumas para fabricação de colchões, travesseiros, etc. Suas aplicações mais interessantes ocorrem na solução de um problema contrário ao de corte, o chamado problema de empacotamento, que consiste, basicamente, em empacotar unidades pequenas dentro de uma unidade grande, de tal forma que um certo objetivo seja otimizado.

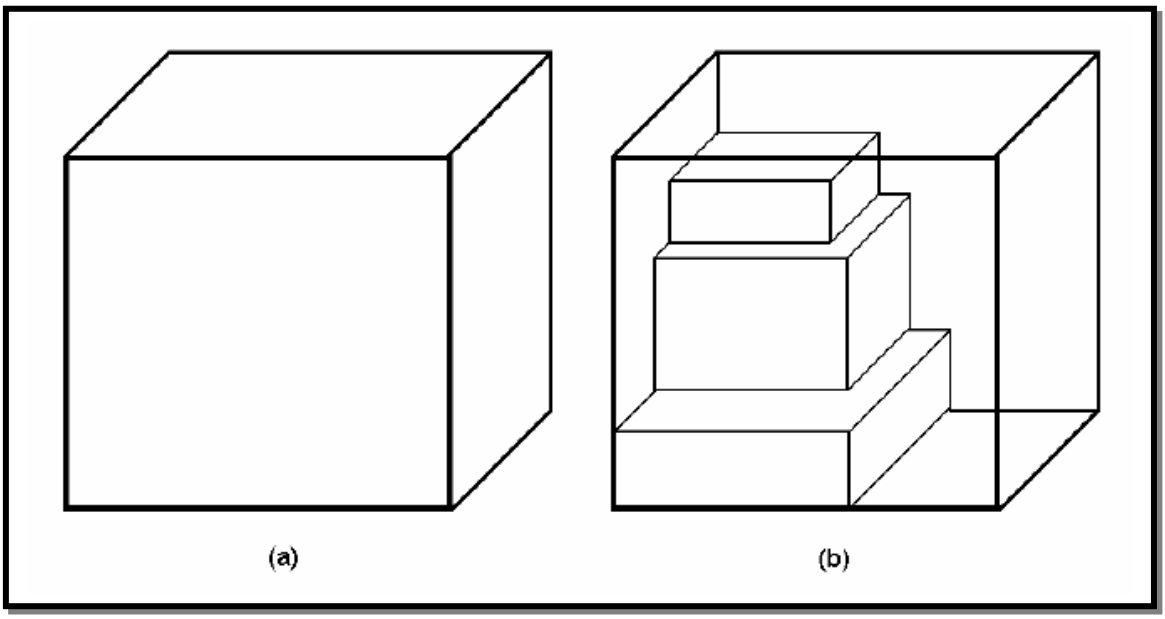

Figura 3 – (a) Contêiner; (b) Exemplo de empacotamento.
Fonte: POLDI e ARENALES, 2003

- (*n*) o problema é n-dimensional, ou multidimensional, quando n dimensões (*n* >3) do objeto são relevantes no processo de corte. Exemplo: empacotamento de caixas de comida em fornos para cozimento (neste caso, o tempo de cozimento representa a quarta dimensão), no problema de alocação de tarefas.

Além das classes citadas anteriormente, pode-se incluir a referida como *n ½* - dimensional quando *n+1* dimensões do objeto são relevantes no processo de corte e uma delas é “variável”. Exemplo: no corte de rolos de tecidos, o objeto possui uma largura fixa, porém seu comprimento é variável. Note que duas dimensões são relevantes no processo de corte, porém este difere do problema bidimensional (ou do unidimensional). Neste caso, o problema é dito ser do tipo *1 ½* - dimensional.

2. Tipo de alocação:

Indica que os itens a serem produzidos serão combinados, respeitando-se restrições associadas ao objeto. Itens e objetos podem ser selecionados de acordo com as seguintes possibilidades de combinação:

- (B) alguns itens são atribuídos a todos os objetos.

  Exemplos: problema da mochila, problema do carregamento de pálete.

- (V) todos os itens são atribuídos a alguns objetos.

  Exemplos: problema do carregamento de veículos, problema do corte de estoque, problema do balanceamento de uma linha de montagem, problema de alocação de memória, problema de alocação de tarefas.

3. Variedade dos objetos:

É um atributo relacionado ao tipo e aparência dos objetos. A variedade é representada da seguinte maneira:

- (O) apenas um objeto.

  Exemplos: problema da mochila, problema do carregamento de pálete.

- (I) objetos de mesmo formato, tamanho e orientação.

  Exemplos: problema do carregamento de veículos, problema de alocação de memória.

- (D) objetos de vários formatos, tamanhos e orientações.

  Exemplo: problema de corte envolvendo partes de objetos (resíduos) de períodos anteriores.

4. Variedade dos itens:

É um atributo relacionado ao tipo e aparência dos itens. Quanto à variedade, os itens podem ser classificados como:

- (F) poucos itens de diferentes aparências (formatos, tamanhos e orientações).

  Exemplo: problema do carregamento de veículos.

- (M) muitos itens de muitas aparências diferentes.

  Exemplo: problema da alocação de memória.

- (R) muitos itens, porém, pouca variedade de tipos de itens.

  Exemplo: problema do corte de estoque bidimensional.

- (C) itens congruentes (possuem formatos e tamanhos iguais).

  Exemplo: problema do carregamento de pálete.

Com esses critérios Dyckhoff classificou, de forma consistente e sistemática, os diversos tipos de Problemas de Corte e Empacotamento, agrupando-os em classes, definidas através de uma quádrupla ($\alpha / \beta / \gamma / \delta$), em que α, β, γ e δ correspondem, respectivamente, aos critérios 1, 2, 3 e 4 definidos anteriormente.

A Tabela 1 mostra alguns problemas clássicos de corte e empacotamento encontrados na literatura, com as suas respectivas tipologias.

Tabela 1 – Tipologia de alguns problemas de corte e empacotamento.

Problema	***Tipo***
Mochila clássico	1/B/O/
Mochila multidimensional	/B/O/
Carregamento de palete	2/B/O/C
Carregamento de veículos	1/V/I/F ou 1/V/I/M
Carregamento de contêiner	3/V/I/ ou 3/B/O/
Bin packing clássico	1/V/I/M
Bin packing dual	1/B/O/M
Bin packing bidimensional	2/V/D/M
Cutting stock clássico	1/V/I/R
Cutting stock bidimensional	2/V/I/R
Cutting stock generalizado	1/ / / , 2/ / / ou 3/ / /
Balanceamento de uma linha de montagem	1/V/I/M
Alocação de tarefas em multiprocessador	1/V/I/M
Alocação de memória	1/V/I/M
Câmbio monetário	1/B/O/R
Investimento financeiro em multiperiódicos	n/B/O/

Fonte: VELASCO, 2005

O problema proposto nesta dissertação é classificado como um problema de empacotamento do tipo 3/B/O/, isto é, o problema é tridimensional; o contêiner deve ter seu volume preenchido de forma máxima, mas nem toda carga, necessariamente, será empacotada; consideramos apenas um contêiner.

Na Figura 4, tem um contêiner, que está preenchido com as caixas retangulares de formas idênticas e quantidades variadas, e ambos, contêiner e caixas, apresentam-se com dimensões definidas.

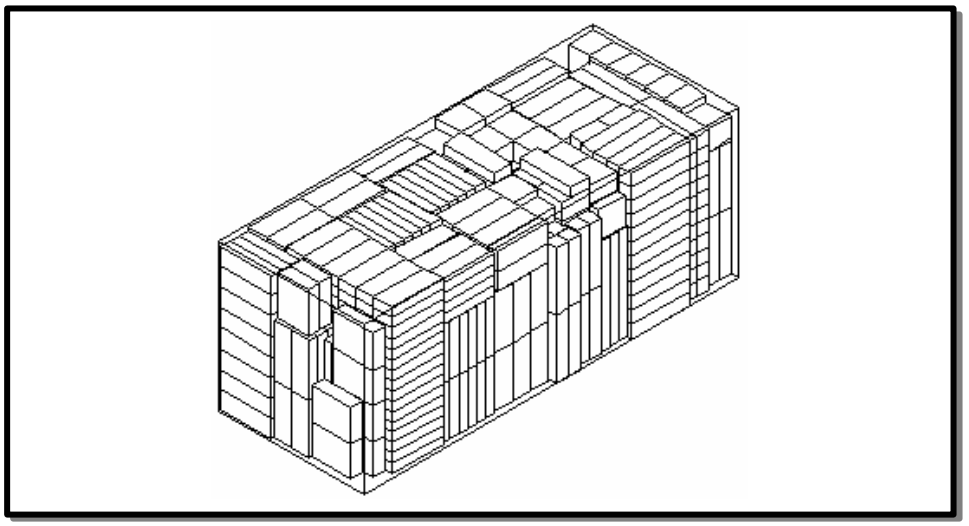

Figura 4 – Empacotamento de caixas no interior de um contêiner.

Fonte: CECÍLIO e MORABITO, 2004

1.3 – Restrições-Chave Físicas

Para preenchimento de um contêiner é necessário que haja uma política que determine a ocupação do espaço interno do contêiner. Nesse sentido, assumem-se restrições-chave físicas associadas a este problema. Existem várias restrições que podem ser encontradas na literatura, como em Morioka e Ronconi (2002). Algumas delas serão descritas a seguir:

Uma delas é a não-sobreposição: as caixas devem ser posicionadas de modo a não ocuparem o mesmo espaço. Como as superfícies de contato não são perfeitas, alguns artigos, como os produzidos por George e Robinson (1980 apud CECÍLIO; MORABITO, 2004), definem folgas entre as caixas para evitar problemas de encaixe. A outra restrição refere-se ao fato de que os itens devem integralmente estar contidos no contêiner, respeitando suas dimensões.

As restrições de orientação, por exemplo, podem ser importantes quando a orientação é fixada (caixas frágeis que não podem ser viradas de ponta cabeça). Outras condições são: a estabilidade (capacidade de evitar que o carregamento movimente-se durante o transporte); agrupamento de itens (itens iguais juntos, ou devido ao fato de serem descarregados juntos); a separação de itens (cujo contato deve ser evitado, como alimentos e remédios); o carregamento completo de itens (como peças de uma mesma máquina); a distribuição de peso dentro do contêiner (distribuição uniforme).

Enfim, identifica-se a complexidade na qual se insere o problema e percebe-se que podem existir várias possibilidades de padrões de empacotamento a fim de se minimizarem as perdas de espaços desocupados dentro do contêiner, visando encontrar soluções viáveis para o problema apresentado.

A Figura 5 apresenta alguns dos padrões a serem produzidos.

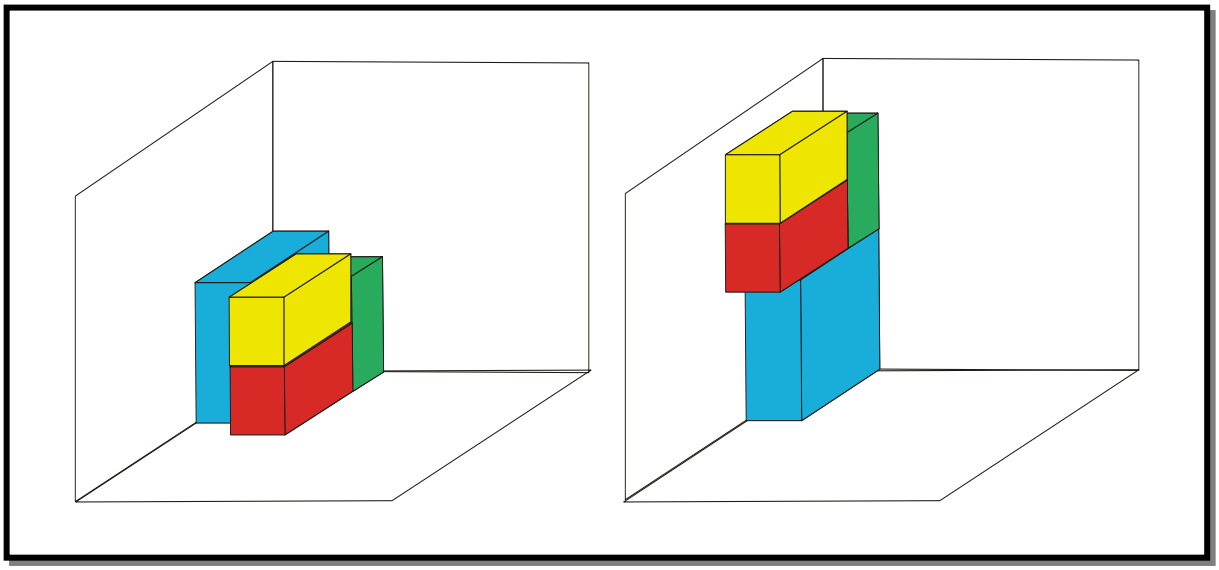

Figura 5 – Padrões de preenchimento no interior de um contêiner.

Fonte: Figura do autor

1.4 – Objetivo

O problema abordado consiste em preencher o contêiner, com a máxima ocupação possível, por itens de faces retangulares, de modo que os espaços vazios no preenchimento sejam mínimos, considerando que os itens são de baixa densidade, para que haja o equilíbrio da carga quando do empilhamento dos itens. Assim, esta dissertação tem como objetivo produzir um algoritmo que encontre boas soluções fundamentadas na GRASP (Greedy Randomized Adaptive Search Procedure) para solução do problema apresentado.

Pretende-se analisar a utilização da técnica de otimização, através de métodos de solução aproximada, como meta-heurística, que se tem destacado nos últimos trabalhos caracterizando-se como métodos eficientes para solução de problemas de otimização.

O propósito do presente estudo é a aplicação de uma metodologia, voltada para o carregamento de um contêiner, mostrando a viabilidade da aplicação prática desta metodologia para obtenção da melhor forma de minimizar as perdas do espaço dentro de um contêiner.

1.5 – Hipótese

Partindo do fato de que o problema de preenchimento de contêiner é de otimização combinatória, e que é necessário encontrar uma solução viável, com baixo esforço computacional, a pesquisa que aqui se apresenta é a implementação de um procedimento baseado na meta-heurística GRASP, que tem como aplicação principal o melhor preenchimento possível de um contêiner, com maior número possível de itens, de modo que a perda em forma de não preenchimento seja mínima quando do empacotamento dos itens.

Serão considerados itens de formas retangulares em todas as faces, de pesos próximos e de baixa densidade para que não haja desequilíbrio; também, os mesmos poderão ser rotacionados, sendo indicada a quantidade de cada item.

A utilização de meta-heurísticas não garante que se encontre a solução ótima de um problema, mas é capaz de retornar solução de qualidade em um tempo adequado. Consiste na aplicação, em cada passo, de uma heurística subordinada, projetada especificamente para o problema particular a ser resolvido. Diversas propostas de meta-heurísticas surgiram, nos últimos anos, impulsionadas pelos problemas. Dentre as meta-heurísticas, a GRASP destaca-se pelos bons resultados obtidos nas aplicações já realizadas e pela facilidade de paralelização e, principalmente, por sua flexibilidade (SILVEIRA e TOSCANI, 1999).

Segundo Vieira Neto (2004), a GRASP tem apresentado resultados bastante satisfatórios para o Problema de Corte Unidimensional, e para o Problema de Corte Bidimensional, conforme ressalta Velasco (2005). Desta forma, a meta-heurística GRASP para solução de Problemas de Empacotamento Tridimensional mostra-se como uma metodologia extremamente promissora.

A heurística proposta construirá soluções viáveis que poderão fornecer padrões de empacotamento para preenchimento de contêiner.

1.6 – Relevância

Esta dissertação visa promover e contribuir para o desenvolvimento científico e tecnológico, bem como para a formação de um aprimoramento do processo produtivo, que necessita otimizar a ocupação dos espaços tridimensionais.

Carregamento de itens em contêineres é uma importante atividade de manipulação em indústrias de fabricação e distribuição. Produtos são embalados em caixas retangulares para proteção e manuseio, sendo estas caixas empilhadas e posicionadas no interior de contêineres para transporte e armazenagem. Esse processo pode ser efetuado de modo manual ou automático. Assim, esse tipo de

problema tem aplicações importantes no planejamento de transporte rodoviário, aéreo, marítimo, etc. (PERIN *et al.*, 2003).

Uma das motivações para a presente pesquisa é a sua relevância acadêmica, uma vez que estará contribuindo para a resolução do problema de carregamento de contêiner. A referida pesquisa trará contribuições para a melhoria dos processos produtivos, bem como poderá tornar viável a automação de alguns processos como, por exemplo, no caso de empresas que precisam fazer entrega de produtos, momento em que a otimização de empacotamento é a essência no que se refere ao número de contêineres necessários para garantir a entrega com menor custo. As empresas poderão se beneficiar com os ganhos advindos do esforço de pesquisa que aqui se propõe.

1.7 – Organização da dissertação

A presente dissertação, no **Capítulo 1** – **Introdução** – esclarece os principais fatores que contribuíram para a proposta de uma discussão sobre as aplicações que demandam otimização da ocupação dos espaços tridimensionais e enumera tanto as características principais para a classificação do problema como as restrições-chave físicas que a ele estão associadas. Também define o objetivo e ressalta a relevância do trabalho realizado, ao mesmo tempo, que expõe a organização da dissertação.

O **Capítulo 2** – **Fundamentação Teórica** – descreve os principais tópicos do referencial teórico construído para a concepção desta pesquisa, estando dividido em sete seções. A primeira delas expõe uma breve revisão bibliográfica acerca de algumas publicações relacionadas ao problema em questão, em ordem cronológica; a segunda parte define o problema de corte e empacotamento; a terceira define a resolução do empacotamento tridimensional; a seguir, a quarta parte ressalta o problema do contêiner como um dos problemas tridimensionais, dando ênfase às suas versões; a quinta seção define algoritmos; a sexta parte define métodos heurísticos e suas diferenças; e a sétima descreve os conceitos teóricos que compõem a meta-heurística GRASP.

O **Capítulo 3** – **Metodologia** – explica o tipo de pesquisa adotado, o universo de pesquisa e a amostra, o tratamento dos dados. Define, também, a modelagem do problema de preenchimento de contêiner e descreve o algoritmo GRASP-3D aplicado ao problema apresentado, apresentando, tanto as variáveis envolvidas como o pseudocódigo.

O **Capítulo 4** – **Resultados Computacionais** – apresenta os resultados obtidos nos testes computacionais com o algoritmo GRASP-3D e a aplicação desta para solução do problema que demanda otimizar a ocupação dos espaços tridimensionais.

O **Capítulo 5** – **Análise dos Resultados e Conclusão** – é dedicado à análise dos resultados obtidos nos testes computacionais com o algoritmo proposto. Além disso, apresenta algumas conclusões constatadas, bem como sugestões para trabalhos futuros.

Também constam do trabalho realizado as **Referências Bibliográficas** com a indicação dos principais textos utilizados na abordagem da problemática, além da apresentação de sugestões de leitura.

2 – FUNDAMENTAÇÃO TEÓRICA

2.1 – Estado da Arte

Problemas de corte começaram a ser estudados por volta de 1940, embora as principais pesquisas tenham surgido, efetivamente, apenas nos anos 60. Segundo os estudos de Poldi e Arenales (2003), as pesquisas na área têm caminhado no sentido de desenvolver técnicas heurísticas adequadas para a resolução de tais problemas.

As pesquisas têm sido realizadas tratando diferentes situações do problema de corte e empacotamento, isto é, são estratégias heurísticas alternativas dedicadas a propostas de soluções, o que indica a importância, tanto acadêmica quanto prática, do problema de corte e empacotamento.

É possível inferir que, a partir do que foi consultado, o problema pode ser modelado segundo uma formulação matemática; porém, nos casos práticos, o número de variáveis e restrições chega à ordem de milhões, conforme assinalam MORABITO e ARENALES (1997). Observa-se, também, que, mesmo para problemas, principalmente de empacotamento tridimensional, a resolução prática torna-se computacionalmente inviável.

De acordo com a pesquisa bibliográfica realizada na literatura especializada na abordagem de corte e empacotamento, diversas variações do problema foram apresentadas e muitas técnicas e algoritmos já foram desenvolvidos. No estudo de tal questão, as variações podem, segundo dados de Klein (2006), ocorrer através de alterações nas restrições, mudanças de objeto e, ainda, podem ser problemas que considerem dimensões diferentes como a unidimensional, bidimensional e tridimensional.

Uma das pesquisas pioneiras na resolução do problema de empacotamento tridimensional é a desenvolvida por George e Robinson (1980), conhecida pela sua simplicidade, flexibilidade de implantação nas situações reais, em relação a outros

métodos descritos na literatura (CECÍLIO; MORABITO, 2004). A abordagem de George e Robinson considera camadas verticais, priorizando-se o carregamento de caixas maiores.

Em Sheithauer (1992), propõe-se um algoritmo aproximado baseado no "forward state strategy" de programação dinâmica. Uma descrição adequada de empacotamento é desenvolvida para a implementação do algoritmo aproximado.

No trabalho desenvolvido por Corcoran e Wainwright (1992) há uma aplicação de Algoritmo Genético, resolvendo o caso de preenchimento de itens por intermédio de faixas bidimensionais.

Em Morabito e Arenales (1994) são revistos métodos descritos pelos autores, tais como os procedimentos em duas etapas de carregamento das caixas em camadas horizontais e em pilhas verticais; a aplicação de técnicas de programação dinâmica e, em particular, revêem-se métodos de busca baseados na representação do espaço de soluções num grafo-E/OU. Algumas dessas abordagens foram implementadas para resolver um exemplo real com 784 caixas, e um exemplo pequeno, mas difícil, com apenas 17 caixas.

Em Bischoff e Ratcliff (1995), é apresentada uma heurística voltada para situações multi-drop, isto é, considera-se um caminhão com vários pontos de descarregamento. O carregamento que vai ser entregue por último é, preferivelmente, alocado ao fundo.

Chen *et al.* (1995) desenvolveram um modelo analítico, no qual se inclui o empacotamento tridimensional de forma indireta. O modelo consiste em selecionar um número de contêineres para empacotar um conjunto dado de itens. Configura-se o citado modelo por ser de programação linear inteira mista zero-um e, mesmo para problemas pequenos, sua resolução prática torna-se computacionalmente inviável.

Uma revisão dos trabalhos publicados nesta área de interesse é apresentada em Dyckhoff et al. (1997) que passa pela história e desenvolvimento dos problemas de corte e empacotamento, além de se relacionarem artigos para uma grande diversidade de problemas abordados pelos autores.

Em Morabito e Arenales (1997) são aplicados os grafos E/OU, com cortes guilhotinados. Nessa proposta, o contêiner original é dividido em duas partes totalmente separadas; essas duas partes são separadas em mais duas e assim sucessivamente até que o espaço abrigue apenas uma caixa, ou seja, torna-se tão pequeno que não consiga abrigar nenhuma outra caixa.

Martello *et al.* (1997) apresentam uma avaliação das recentes técnicas para resolver problemas na versão "*0-1 knapsack*", com ênfase especial na adição de restrição de cardinalidade, programação dinâmica e divisibilidade rudimentar. Resultados computacionais são apresentados comparando todos recentes algoritmos.

Uma adaptação de uma nova metodologia de aproximação para problemas da otimização combinatória, denominada Busca Local Direcionada (GLS) é apresentada em Pisinger *et al.* (1999).

Em Martello *et al.* (2000), tem-se o primeiro algoritmo exato para o problema de empacotamento tridimensional, para o qual uma abordagem *branch-and-bound* foi utilizada. Neste, duas heurísticas são propostas: H1 e H2. A heurística H1 aloca itens dentro de um contêiner de acordo com uma variação do preenchimento por camadas, sugerido em George e Robinson (1980); sendo que os pontos de inserção são pontos de cantos bidimensionais. A heurística H2 faz uma enumeração em árvore, onde os vértices são pontos de cantos tridimensionais. Esta foi implementada e testada, mas devido à grande quantidade de nós gerados pela árvore de busca, não foi possível a execução da mesma na máquina utilizada.

Uma nova heurística é proposta em Pisinger (2002), baseada na aproximação de "wall-building". Propõe-se a decomposição do problema em várias camadas que, novamente, são separadas em várias tiras. O empacotamento pode ser formulado e resolvido como um problema de mochila, com capacidade igual para a largura ou altura do recipiente. A profundidade de uma camada como também a espessura são decididas por uma aproximação "branch-and-bound".

Em Cecílio e Morabito (2001), apresentam-se refinamentos da heurística de George e Robinson (1980), ao preencher o contêiner em camadas e combinar espaços, visando à melhora do empacotamento. Aqueles pesquisadores alteraram

alguns procedimentos destes com a finalidade de aprimorar a solução alcançada, criando duas novas versões da heurística proposta por George e Robinson.

Silva e Soma (2003) apresentam um algoritmo polinomial na quantidade de recursos computacionais utilizados, tendo como maior contribuição considerações sobre a estabilidade estática da carga. O procedimento realizado para o preenchimento dos itens dentro dos bins utiliza o princípio da alocação em pontos de cantos.

Cecílio e Morabito (2004) abordam o problema do carregamento de caixas dentro de contêineres. Heurísticas são propostas para resolver a questão, baseada em extensões e refinamentos da conhecida heurística de George e Robinson. Experimentos computacionais a partir de exemplos citados na literatura, e situações reais de carregamento de contêineres foram realizados, sendo os resultados obtidos comparados com resultados de outros métodos.

Arenales *et al.* (2004), têm como objeto de estudo os problemas de corte e empacotamento, cuja solução é essencial para o planejamento da produção em diversas indústrias. O texto elaborado tem como base notas, apostilas de cursos ou minicursos sobre corte e empacotamento, preparados e ministrados anteriormente pelos autores. No referido trabalho foram reproduzidos diversos trechos dessas notas e/ou apostilas, o que ressalta a importância econômica da pesquisa, aliada à dificuldade de resolução de problemas de corte e empacotamento. Isso vem comprovar o envolvimento da comunidade acadêmica na busca de métodos que levam a soluções eficientes para um problema que, conforme já foi apontado, traz dificuldades para o processo produtivo.

Em CEAC (1999) apresenta um projeto temático na área de corte e empacotamento assistido por computador. Este projeto englobava quatro universidades: USP-São Carlos, UFSCAR-São Carlos, ITA-São José dos Campos e INPE-São José dos Campos, Estado de São Paulo e teve como objetivos: o desenvolvimento de algoritmos para resolução de problemas industriais de natureza combinatória, bem como achar aplicabilidade dos resultados obtidos das pesquisas e integrar pesquisadores de todo o Brasil que já vinham desenvolvendo trabalhos na área de corte e empacotamento. O projeto temático foi responsável pelo desenvolvimento científico e tecnológico bem como a formação de recursos

humanos na área de corte e empacotamento assistido por computador, durante sua execução foram abordados tópicos importantes e desafiantes.

É importante destacar que essas e tantas outras aplicações, que podem utilizar algoritmos de corte e empacotamento, provocaram um aumento significativo nos estudos relacionados a problemas dessa área.

2.2 – Corte e Empacotamento

Segundo Arenales *et al.* (2004), o problema de empacotamento consiste, genericamente, em empacotar unidades pequenas dentro de uma unidade grande, de forma que certo objetivo seja otimizado. Um exemplo desse problema consiste em arranjar o maior volume possível de caixas dentro de um contêiner. Por outro lado, o problema de corte, de forma genérica, consiste em cortar uma unidade grande (objeto), que esteja disponível, para a produção de um conjunto de unidades pequenas (itens) que estão sendo requisitadas. As formas e medidas do objeto e dos itens são bem especificadas.

Cortar unidades maiores em unidades menores, ou empacotar unidades menores dentro de unidades maiores, são problemas idênticos, considerando que um item cortado de certa posição pode ser pensado como alocado àquela posição. Por esse motivo, problemas desta classe são referidos como problemas de corte e empacotamento.

Dependendo dos itens solicitados, pode-se combiná-los dentro de um objeto de inúmeras maneiras, respeitando-se um conjunto de restrições do processo de cortagem. A essas combinações denominamos planos de corte. O plano de corte ótimo é aquele que produz, por exemplo, a menor perda. O número de planos de corte possíveis é, na prática, muito elevado, exigindo que técnicas bem elaboradas sejam desenvolvidas para determinar o plano ótimo.

Segundo Dyckhoff (1990), a estrutura lógica básica dos problemas de corte e empacotamento pode ser facilmente percebida se olharmos os problemas 1, 2 e 3 a seguir.

- Problema 1: O corte de tubos para equipamentos de calefação (Figura 6). Considere que exista um estoque ilimitado de tubos grandes, de tamanho 98cm, usados para a produção de tubos menores, e uma lista de tubos menores, com tamanhos entre 5 e 46cm, que devem ser produzidos para atender a uma demanda semanal (HEICKEN; KÖNG, 1980, *apud* DYCKHOFF, 1990).

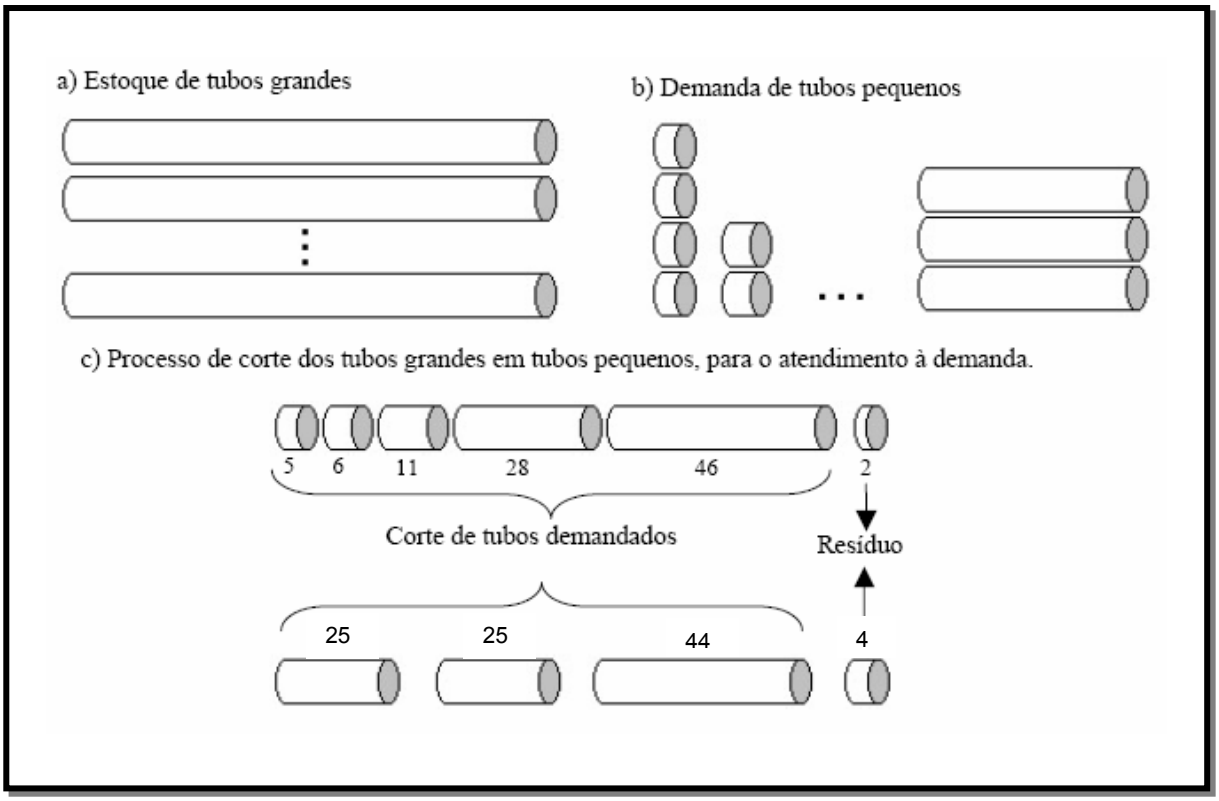

Figura 6 – Corte de tubos em estoque para atendimento à demanda.
Fonte: FIGUEIREDO, 2005

- Problema 2: O carregamento de contêineres (Figura 7). Este tipo de problema também apresenta dois grupos de dados: um estoque de objetos, consistindo de um ou mais contêineres e uma lista de itens que deverão ser alocados dentro do(s) contêiner(es) (GEHRING et al, 1990; HAESSLER; TALBOT, 1990; apud DYCKHOFF, 1990).

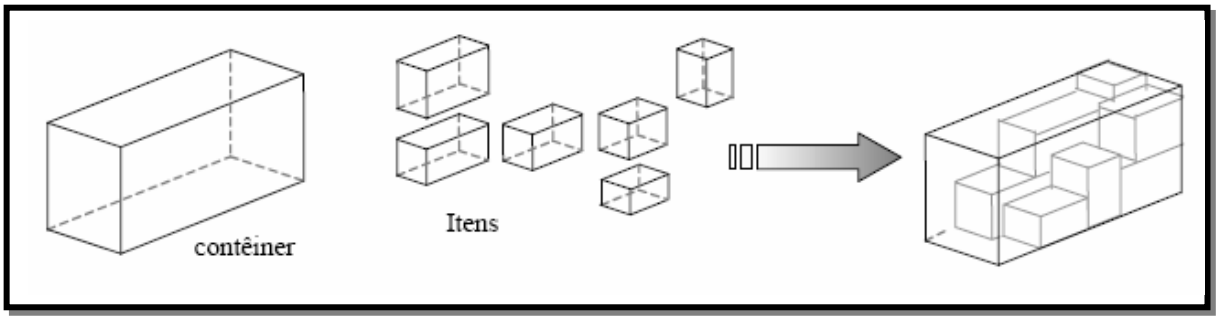

Figura 7 – Problema de carregamento de contêiner.
Fonte: FIGUEIREDO, 2005

- Problema 3: O corte de placas de madeira para a confecção de móveis (Figura 8). Consiste em encontrar a melhor forma de cortar um objeto retangular, para a produção de itens, também retangulares (DYCKHOFF, 1990).

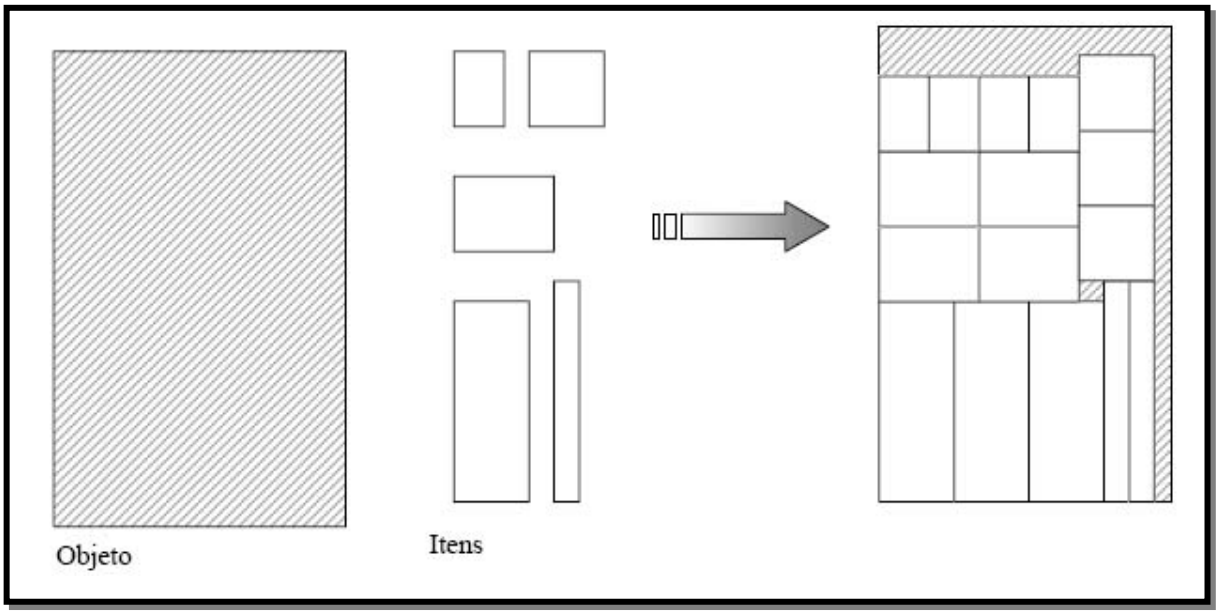

Figura 8 – Problema do corte de estoque bidimensional.
Fonte: FIGUEIREDO, 2005

2.3 – Empacotamento Tridimensional

A definição do problema de empacotamento tridimensional de objetos, com faces retangulares, consiste em empacotar um conjunto de itens em um objeto. Assim, o objetivo é maximizar ou minimizar uma determinada função, conforme afirma Klein (2006).

Neste caso, os produtos deverão ser arranjados em grandes espaços, de tamanhos padronizados previamente projetados, como, por exemplo, caixas de papelão ou madeira, contêineres, etc. Entretanto, esse procedimento introduz um novo estágio – a operação de empacotamento – que nem sempre consegue preencher todos os espaços disponíveis (nas caixas, contêineres, etc), gerando espaços ociosos, os quais serão, conseqüentemente, “armazenados” e/ou “transportados” juntamente com os itens produzidos. Surge então a necessidade de planejar o empacotamento de modo a minimizar os espaços ociosos (ARENALES *et al.*, 2004).

Segundo Silva e Soma (2003), a motivação prática para o estudo do empacotamento tridimensional vem da grande quantidade de aplicações industriais, as quais variam do carregamento de cargas de aviões ao seqüenciamento de tarefas em ambientes multiprocessados.

2.4 – Problema do Contêiner

Klein (2006) ressalta que o problema do contêiner é um dos problemas tridimensionais mais estudados, dando ênfase às suas duas principais versões, a *knapsack* e a *bin packings*. O objetivo de qualquer uma das versões consiste em um conjunto de caixas que deve ser colocado em um contêiner, de forma que se maximize ou minimize uma função. Essa função pode variar de acordo com a definição do problema, mas, em geral, refere-se ao volume empacotado.

Na versão *knapsack* (mochila) existe apenas um contêiner e não existe a garantia de que todos os itens serão empacotados. Cada caixa possui um valor associado e o objetivo é alcançar um valor total máximo.

Na versão *bin packing* o objetivo é empacotar todas as caixas do conjunto, tendo disponível para isso mais de um contêiner. O objetivo é fazer com que o número de contêineres necessários seja o menor possível. Neste caso, os contêineres possuem o mesmo tamanho. Quando os contêineres variam de tamanho, o problema é conhecido como *multi-container loading*.

Além do arranjo geométrico das caixas dentro de cada contêiner, outras restrições devem ser consideradas. Muitas restrições que ocorrem na prática estão sendo adicionadas aos problemas como, por exemplo, a capacidade de peso máximo que o contêiner pode suportar, e a estabilidade do carregamento (que são verificadas movimentando-se o carregamento produzido).

Bischoff (2003) ressalta que importantes avanços têm ocorrido, não apenas na busca de solução para esses problemas, mas também ao se considerarem outros fatores que aumentam a dificuldade de resolução dos problemas.

2.5 – Algoritmos

Conforme Cormen *et al*. (2002), algoritmos são procedimentos computacionais que, a partir de valores de entrada, geram valores de saída desejados, ou seja, são passos necessários para transformar uma determinada entrada em uma saída que atinja o objetivo para o qual foi projetado.

É possível perceber que podem existir inúmeros algoritmos para resolver o mesmo problema, o objetivo é o mesmo, mas os passos para alcançá-lo podem variar, conforme afirma Klein (2006).

A análise de algoritmos, segundo Cormen *et al.* (2002), deve prever as necessidades de recursos dos algoritmos. Com base nos recursos necessários, é possível medir o tempo computacional do algoritmo, para que, a partir dessa

informação, se possa escolher entre os algoritmos que chegam ao resultado desejado.

Segundo Klein (2006), o que realmente é importante em um algoritmo é o crescimento de seu tempo de conclusão em função de seus dados de entrada.

2.6 – Métodos Heurísticos

Os métodos heurísticos, conforme destaca Velasco (2005), são procedimentos que visam à obtenção de soluções de qualidade, num período de tempo razoável, diferindo segundo a estratégia que usam para buscar e construir suas soluções.

As heurísticas de construção têm por objetivo construir uma solução, elemento por elemento. A forma de escolha de cada elemento a ser inserido a cada passo varia de acordo com a função de avaliação adotada, a qual, por sua vez, depende do problema abordado. Nas heurísticas construtivas clássicas, elementos candidatos são geralmente ordenados segundo uma função gulosa, que estima o benefício da inserção de cada elemento, e somente o “melhor” é inserido a cada passo (SOUZA, 2006). Segundo Velasco (2005), as heurísticas construtivas, em muitos casos, são aplicadas para se obter uma solução inicial que poderá ser melhorada e, para isto, emprega-se um artifício de melhoria sobre ela, a fim de encontrar um resultado mais interessante.

As heurísticas de refinamento em problemas de otimização, também chamadas de técnicas de busca local, constituem uma família de técnicas baseadas na noção de vizinhança e consistem em melhorar uma solução, através de modificações em seus elementos (SOUZA, 2006).

Discorrendo sobre este assunto, Velasco (2005) afirma que:

> A busca local começa a partir de uma solução inicial viável, que pode ser obtida por uma heurística construtiva ou produzida aleatoriamente, da qual se gera uma vizinhança de soluções, ou seja, conjunto de soluções obtidas a partir de modificações feitas na solução inicial, visando escolher o melhor vizinho para ser a nova

> solução corrente, seguindo um critério de escolha. [...] Métodos mais flexíveis e de caráter geral, apresentando condições de escapar de ótimos locais, são conhecidas como meta-heurísticas.
>
> As meta-heurísticas são métodos mais eficazes de busca local que possibilitam encontrar soluções melhores. Executam procedimentos de busca em vizinhanças, que evitam a parada prematura em ótimas locais durante a procura de soluções de melhor qualidade, aumentando as chances de se chegar ao ótimo global, podendo até mesmo usar uma estratégia de piorar as soluções, a fim de escapar do ótimo local. Este processo respeita uma seqüência de passos bem definidos, até atingir um critério de parada.

Para Souza (2006), as propostas de meta-heurísticas encontradas na literatura apresentam diferentes características nas suas estruturas, o que distinguem uma das outras.

A seguir é apresentado o procedimento heurístico referenciado nesta dissertação para resolução do problema proposto.

2.7 – GRASP

Proposta por Feo e Resende (1995), a meta-heurística GRASP – *Greedy Randomized Adaptive Search Procedure*, que significa em português, “Procedimento de Busca Adaptativa Gulosa e Aleatória”, é um método iterativo que consiste de duas fases: uma fase de construção, na qual uma solução gulosa e aleatória é gerada, elemento a elemento; e uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as interações realizadas é retornada como resultado do algoritmo de otimização GRASP.

Uma característica especialmente interessante da GRASP é a facilidade com que pode ser implementada. Poucos parâmetros precisam ser fixados e regulados (tamanho da lista de candidato e número de repetições da GRASP), e então, o desenvolvimento pode ser focalizado em implementar em estrutura de dados eficientes para assegurar repetições da GRASP rápidas (FEO; RESENDE, 1995).

O pseudocódigo descrito pela Figura 9 ilustra o algoritmo GRASP.

```
procedimento GRASP ()
        Entrada da instância;
        para ( critério de parada GRASP não for satisfeito ) faça
                ConstruirSoluçãoGulosaAleatória(solução);
                BuscaLocal(solução);
                AtualizarSolução(solução, melhor solução encontrada);
        fim-para;
        retorna (melhor solução encontrada);
fim GRASP.
```

Figura 9 – Algoritmo GRASP.
Fonte: FEO e RESENDE, 1995

Considera-se, para o algoritmo GRASP, uma função objetivo que associa uma solução a um valor real que procuramos maximizar ou minimizar. A linha 1 do pseudocódigo corresponde à entrada de dados. Um conjunto de procedimentos é executado repetidamente entre as linhas 2 e 6, e termina quando algum critério de parada, como número máximo de iterações permitidas sem melhora da função objetivo ou a solução buscada seja encontrada, seja satisfeita. Na linha 3 está a fase de construção da GRASP, enquanto na linha 4 está a fase de busca local. Se uma solução melhorada é encontrada, a solução inicial é atualizada na linha 5, e retorna a melhor solução encontrada na linha 7.

Na fase de construção, uma solução viável é construída 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 ordenação pré-determinado. O processo de seleção é baseado em uma função adaptativa gulosa $g: C \rightarrow R$, que estima o benefício da seleção de cada um dos elementos. A componente probabilística do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos, que recebe

o nome de *lista restrita de candidatos (LRC)*. Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.

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 (VIEIRA NETO, 2004). Assim, o parâmetro α regula o grau de miopia e aleatoriedade da 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 soluções examinado e, conseqüentemente, a probabilidade de escapar de um ótimo local diminuirá. Velasco (2005) observa que uma *LRC* que apresenta uma cardinalidade grande, produz muitas soluções diferentes, aumentando a perspectiva de escapar de um ótimo local indesejado.

A Figura 10 propõe o pseudocódigo de um procedimento de construção da GRASP.

```
procedimento ConstruirSoluçãoGulosaAleatória ()
        solução = { };
        para ( solução construída incompleta ) faça
                CriarLRC(LRC);
                s = ElementoSelecionadoAleatoriamente(LRC);
                solução = solução ∪ {s};
                AdaptarFunçãoGulosa(s);
        fim-para;
fim ConstruirSoluçãoGulosaAleatória.
```

Figura 10 – Fase de Construção da GRASP.
Fonte: FEO e RESENDE, 1995

A solução a ser construída é iniciada na linha 1 do pseudocódigo. Os comandos entre as linhas 2 e 7 são repetidos até que a solução seja construída. Na linha 3, é construída a lista restrita de candidatos. Um candidato da lista restrita é selecionado, aleatoriamente, na linha 4, e é acrescentado na solução na linha 5. E

na linha 6, a função gulosa atualiza as informações de acordo com o elemento incluído.

Segundo Souza (2006), a eficiência da busca local depende da qualidade da solução construída. O procedimento de construção tem um papel importante nessa fase, uma vez que as soluções construídas constituem bons pontos de partida para a busca local, permitindo acelerá-la.

Assim como em muitas técnicas determinísticas, as soluções geradas pela fase de construção da GRASP provavelmente não são localmente ótimas com respeito à definição de vizinhança adotada. Daí a importância da fase de busca local, a qual objetiva melhorar a solução construída (RESENDE, 1998).

A fase de busca local está baseada na noção da vizinhança. A função *N*, a qual depende da estrutura do problema tratado, associa a cada solução viável *s* sua vizinhança *N(s)*. Cada solução *s' ∈ 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.

A Figura 11 propõe o pseudocódigo de um procedimento GRASP para fase de Busca Local considerando uma vizinhança gerada, *N(.)* de *s* (solução atual).

```
procedimento BuscaLocal ()
    para ( s não localmente ótimo ) faça
        Encontrar uma melhor solução s´ ∈ N(s) ;
        se ( f(s´) melhor que f(s) )
            s = s´ ;
        fim-se;
    fim-para;
    retorna ( s como ótimo local para P );
fim BuscaLocal.
```

Figura 11 – Fase de Busca Local da GRASP.
Fonte: FEO e RESENDE, 1995

O conjunto de repetições iniciado na linha 1 do pseudocódigo tem como critério de parada a solução que seja localmente ótima para uma vizinhança. Na linha 2 são efetuadas modificações que transformam uma solução em outra pertencente a sua vizinhança, com intuito de obter a melhor solução encontrada. Caso uma solução encontrada na vizinhança seja melhor que a solução tratada, este vizinho passa a ser a solução corrente na linha 3. E na linha 5, retorna a melhor solução encontrada para o problema *P*. considerando uma vizinhança gerada, *N(.)* de *s* (solução atual).

3 – METODOLOGIA

3.1 – Tipo de Pesquisa

Considerando-se os critérios de classificação de pesquisa propostos por Vergara (2004), quanto aos fins e aos meios, tem-se:

a) Quanto aos fins, trata-se de uma pesquisa descritiva, pois pretende expor as características das metodologias com abordagens para o problema de corte e empacotamento já utilizados; e estabelecer correlações entre variáveis, com natureza já definida, expondo o melhor arranjo entre as mesmas. A pesquisa tem o objetivo de fornecer uma base para explicar a metodologia proposta para solução do problema.

b) Quanto aos meios, trata-se de uma pesquisa, ao mesmo tempo, de laboratório e experimental.

   Classifica-se como pesquisa de laboratório, já que se recorrerá a uma ferramenta computacional no auxílio à pesquisa, já que no campo seria praticamente difícil realizá-la.

   A pesquisa também é experimental, porque será feito uso de procedimentos experimentais que permitirão observar e analisar os resultados obtidos.

3.2 – Universo e Amostra

Esta pesquisa visa solucionar o problema: como arrumar o maior volume possível de caixas dentro de um único contêiner? Como já mencionado anteriormente, as abordagens aplicáveis a este problema vêm sendo amplamente

pesquisadas ao longo dos anos e várias metodologias aplicadas a este foram propostas.

A amostra desta pesquisa baseia-se em um exemplo real introduzido por George e Robinson (1980), referente ao carregamento de um contêiner numa Companhia da Nova Zelândia. Ressalta-se que várias literaturas estudadas para esta pesquisa utilizaram este exemplo, tais como: Morioka, 2002; Cecílio e Morabito, 2004; Morabito e Arenales, 1997; entre outras.

Para compor a amostra, a Tabela 2 apresenta os dados da carga composta de 784 caixas (com volume 26,325 m^3) de m=8 tamanhos diferentes, que deve ser carregada num contêiner padrão internacional ISO série 2, cujas dimensões são: C=5793mm, L=2236mm e H=2261mm (com volume 29,287 m^3), conforme dados do exemplo de George e Robinson (1980) retirados da pesquisa proposta por Morabito e Arenales (1997).

Tabela 2 – Dados do exemplo de George e Robinson (1980).

i	c_i	l_i	h_i	q_i
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
				784 *caixas*

Fonte: MORABITO e ARENALES, 1997

3.3 – Tratamento dos Dados

A GRASP possui uma estratégia heurística mais flexível. Ela combina métodos de otimização baseados em algoritmos determinísticos (busca local) e estocásticos (busca global) configurando um sistema híbrido de otimização. O objetivo dessa combinação é conjugar a atuação entre os dois métodos, de modo que cada um atue na condição de melhor desempenho. Conforme afirmam Feo e Resende (1995), dentro de uma faixa de liberdade, o método global atua definindo uma região de mínimo e o método local atua refinando e definindo o mínimo dessa região.

Propõe-se, portanto, a construção de uma GRASP para projetar um sistema de otimização do problema de preenchimento de contêineres, que será composto de duas fases.

Inicialmente, define-se o conjunto $C = \{b_1, b_2, \cdots, b_m\}$, 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.

Um padrão de preenchimento será obtido tomando-se *n* elementos do conjunto *C*, sendo $n \leq m$, construindo um subconjunto restrito $LRC = \{b_1, b_2, \cdots, b_n\}$ formado pelas caixas de maior volume que compõem a lista de candidatos, e a partir dos itens deste conjunto, procura-se estabelecer uma seqüência de preenchimento do contêiner, camada por camada. Uma camada é uma seção do comprimento do contêiner na completa altura *H* e largura *L*, como mostra a Figura 12, 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.

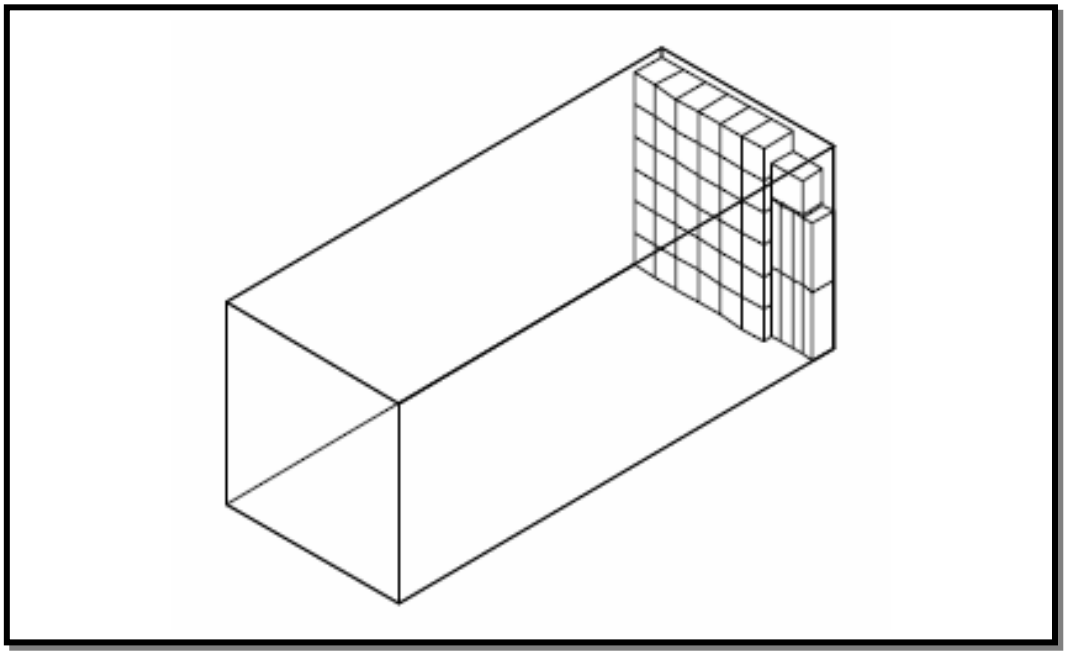

Figura 12 – Primeira camada do contêiner.
Fonte: CECÍLIO e MORABITO, 2004

Na fase de Busca Local da GRASP, a solução construída é submetida a modificações que levam a gerar vizinhos e, com base no valor da função objetivo *f*, escolher, provavelmente, uma nova solução. Quando não se encontra uma solução que seja melhor que a construída, diz-se que a solução é localmente de boa qualidade.

3.4 – Modelagem do Problema

A resolução de um problema de otimização passa por duas fases: uma consiste em transformar o problema em um modelo e, posteriormente, a outra, em que um algoritmo deve ser construído e implementado para resolver o modelo.

Conforme Perin *et al.* (2003), 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*. Todas as caixas possuem densidades próximas, e serão carregadas ortogonalmente dentro do contêiner, e com uma orientação fixada, isto é, com c_i , l_i e h_i paralelos a *C, L* e *H,* respectivamente, como mostra a Figura 13.

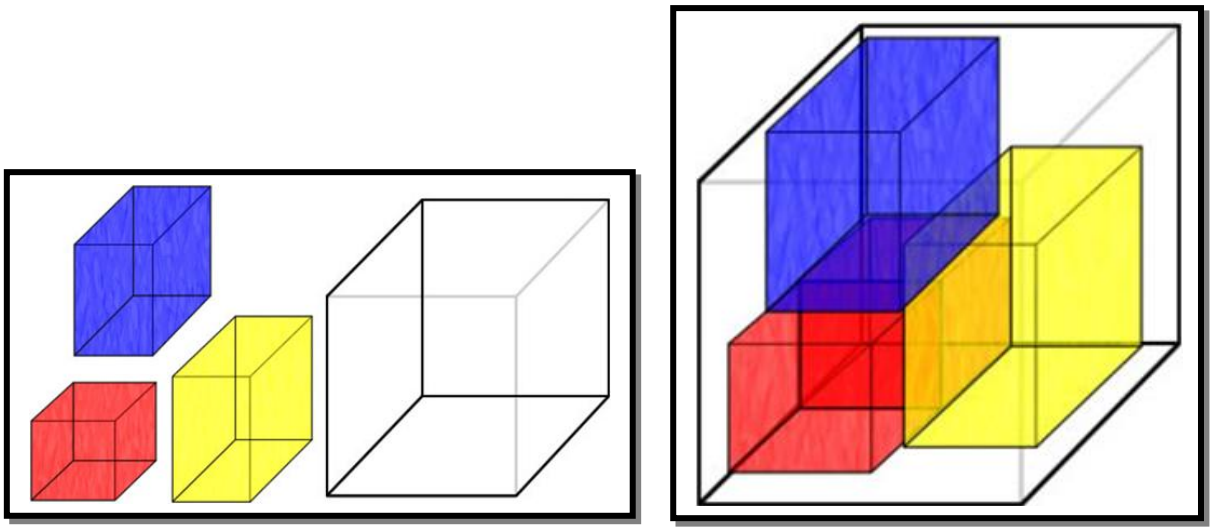

Figura 13 – Carregamento de caixas dentro do contêiner.
Fonte: Figura com base em MORABITO e ARENALES (1997).

Supõe-se que há *n* padrões de empacotamento especificados por uma matriz *mxn*-matriz *A*, isto é, o elemento a_{ij} da matriz especifica o número de vezes que a caixa *i* é considerada no padrão de empacotamento *j*. Isto implica que $f_j = \sum_{i=1}^{m} a_{ij} c_i l_i h_i \leq CLH$. Deseja-se determinar um vetor de dimensão n, de quantidades $w = (w_j)$, para todo $j \in J = \{1,2,...,n\}$. Este problema é dado por:

$$\max \left\{ \sum_{j=1}^{n} f_j w_j \ : Aw < q, \ \ w \in W \right\} \qquad (01)$$

onde:

f_j representa o somatório das caixas consideradas no padrão de empacotamento.

w_j representa o número de vezes que a camada (padrão) j é utilizada.

A função objetivo deste modelo implica *maximizar o volume empacotado no contêiner*, atendendo, se possível, a todos os itens dados no problema.

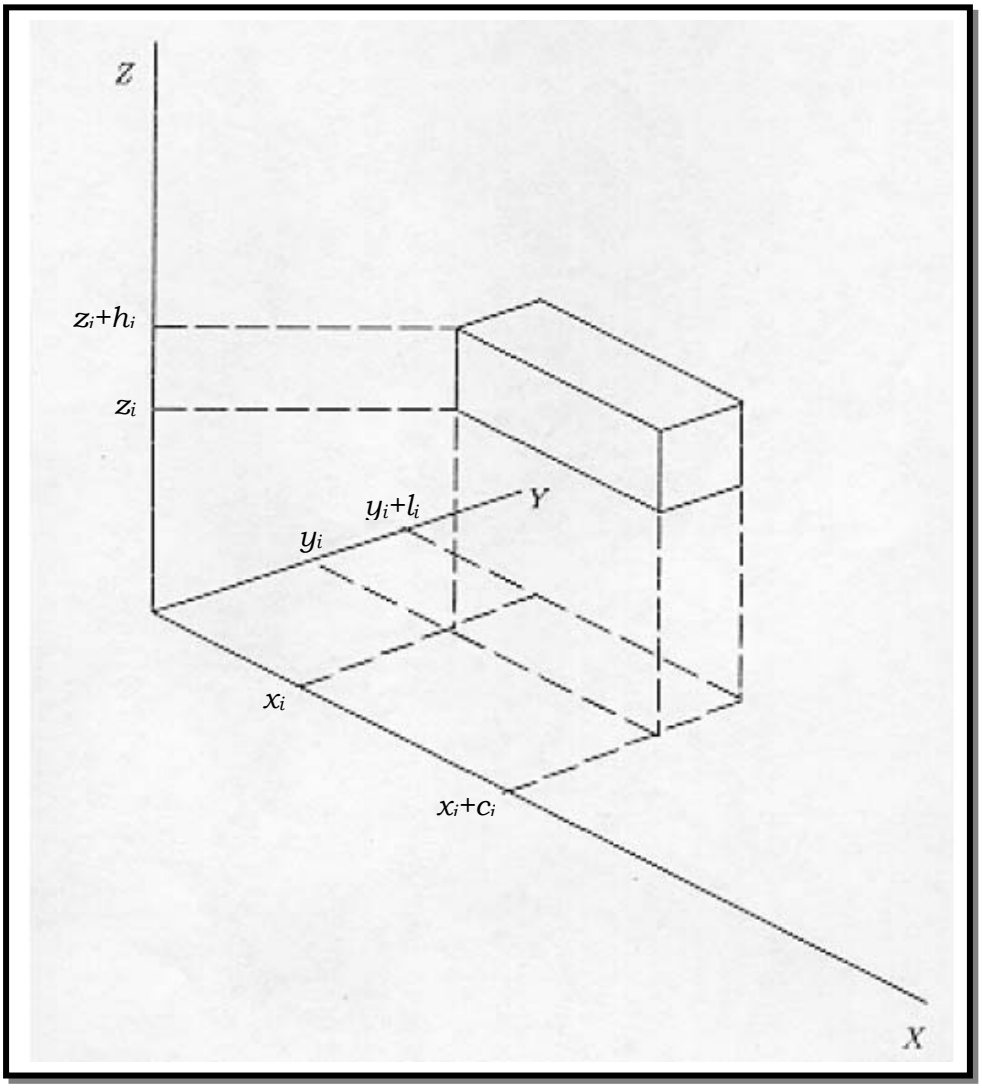

Figura 14 – Colocação da Caixa tipo *i* (c_i, l_i, h_i) na posição (x_i, y_i, z_i) do contêiner.

Fonte: MORABITO e ARENALES, 1997

A resolução de um problema de empacotamento, em geral, envolve a geração de padrões de empacotamento. Utilizando ternos de variáveis de decisão (x_i , y_i ,z_i) para denotar as coordenadas do canto inferior esquerdo frontal da caixa *i* no padrão de empacotamento *j*, isto é, a localização das caixas; a matriz *A* deve ser construída de tal modo que assegure que todos os itens colocados dentro do contêiner encaixam-se dentro de suas dimensões físicas, como mostra a Figura 14, e deve satisfazer as restrições:

$$x_i + c_i \leq C \quad (02)$$
$$y_i + l_i \leq L \quad (03)$$
$$z_i + h_i \leq H \quad (04)$$

$$x_i, y_i, z_i, c_i, l_i, h_i \geq 0, \forall i \in I$$

onde:

x_i, y_i, z_i são as coordenadas do canto inferior esquerdo frontal da caixa *i*.
c_i, l_i, h_i são as medidas de comprimento, largura e altura da caixa *i*.
C, L, H são as medidas de comprimento, largura e altura do contêiner.

Além disso, assegure que não haverá sobreposição para nenhum par *(i,j)* de itens, isto é, que as caixas não ocupem o mesmo espaço dentro de um mesmo contêiner; e isso é construído com as seguintes restrições:

$$x_i + c_i \leq x_j \quad ou \quad x_j + c_j \leq x_i \quad (05)$$
$$ou$$
$$y_i + l_i \leq y_j \quad ou \quad y_j + l_j \leq y_i \quad (06)$$
$$ou$$
$$z_i + h_i \leq z_j \quad ou \quad z_j + h_j \leq z_i \quad (07)$$

$$x_i, y_i, z_i \geq 0, \forall i \in I$$
$$x_j, y_j, z_j \geq 0, \forall j \in J$$

onde:

x_i, y_i, z_i são as coordenadas do canto inferior esquerdo frontal da caixa *i*.
c_i, l_i, h_i são as medidas de comprimento, largura e altura da caixa *i*.
x_j, y_j, z_j são as coordenadas do canto inferior esquerdo frontal da caixa *j*.
c_j, l_j, h_j são as medidas de comprimento, largura e altura da caixa *j*.

3.5 – Aplicação da GRASP ao Problema

O algoritmo GRASP-3D, implementado nesta dissertação, baseia-se no preenchimento de um contêiner construindo camadas ao longo do seu comprimento. Em cada camada, procura-se carregar o máximo de colunas completas de caixas do mesmo tipo possíveis. O espaço restante à frente, ao lado e em cima das caixas já empacotadas forma os novos espaços de preenchimento. Uma nova camada não deve ser iniciada até que a camada anterior esteja totalmente preenchida.

Quando um conjunto de caixas é ajustado em um espaço, podem ser criados três sub-espaços: acima das caixas empacotadas (espaço da altura), ao lado (espaço da largura) e em frente (espaço do comprimento), conforme indica a Figura 15. O espaço à frente de cada caixa empacotada é criado após a definição do comprimento da camada, que é determinada pela caixa de maior comprimento utilizada no preenchimento da camada atual.

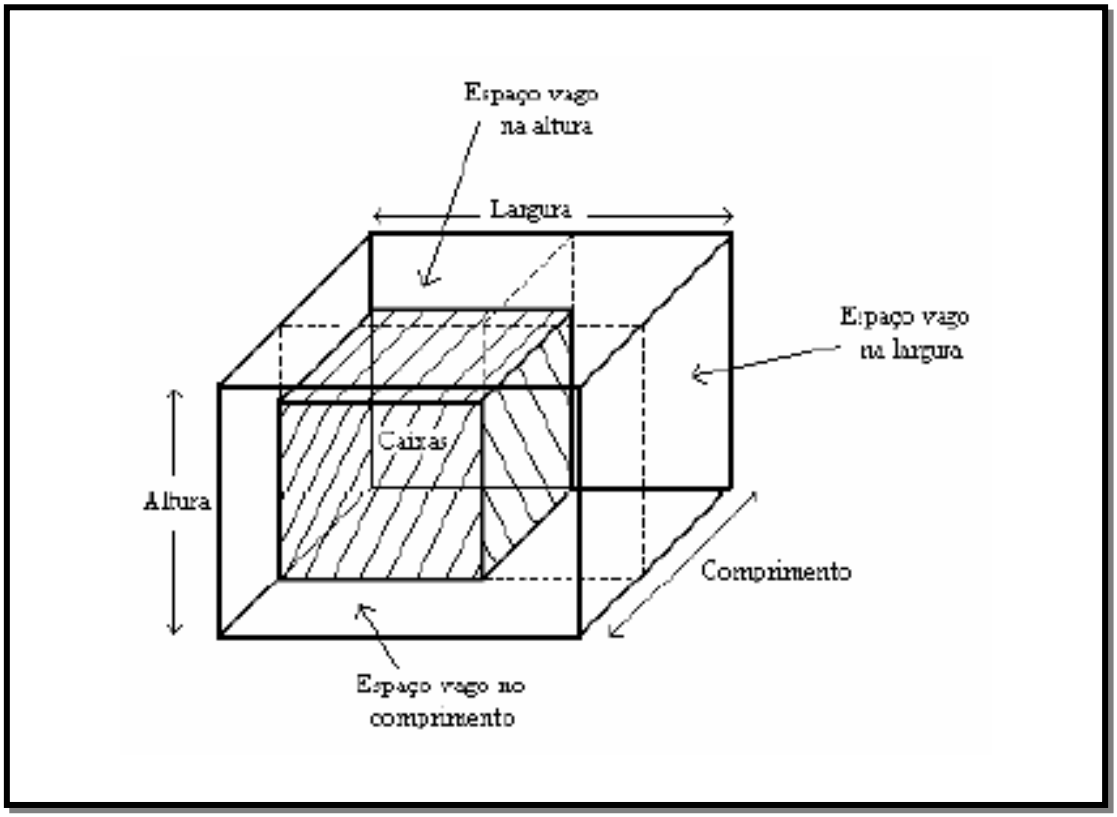

Figura 15 – Criação de novos espaços.

Fonte: CECÍLIO e MORABITO, 2004

O espaço restante à frente da camada atual é colocado em um estoque de espaços para criação de novas camadas.

O espaço à frente das caixas menores que a dimensão de comprimento da camada, o espaço do lado e/ou de cima das caixas serão descartadas quando suas dimensões são menores que a dimensão mínima de qualquer caixa que esteja na lista restrita de candidatos. Estas informações são armazenadas em espaço rejeitado, isto é, espaço que não pôde ser carregado com nenhum tipo de caixa.

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 $\overline{\mathrm{LRC}}$. 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 $\overline{\mathrm{LRC}}$ 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 atualização das quantidades das caixas utilizadas e das listas restritas cuja quantidade de caixa zerar.

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. E faz este processo sucessivamente a cada tipo de caixa diferente existente 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 vazio a ser preenchido.

Na busca pela caixa para preencher os espaços ociosos na altura, na largura ou no comprimento, se necessário for, as caixas poderão ser rotacionadas trocando

suas dimensões de largura com comprimento, altura com comprimento ou largura com a altura.

A cada camada criada atualiza-se o estoque de espaços, que é o espaço ao longo do comprimento do contêiner. Assim que não houver mais caixas que caibam neste espaço, este também será armazenado como espaço rejeitado.

O algoritmo GRASP-3D tem como critérios de parada: a quantidade total de caixas zerar, se estoque de espaços zerar ou o número de iterações chegar ao máximo.

Ao término, o algoritmo apresentará a melhor solução encontrada indicando a quantidade de caixas que não foram empacotadas, a porcentagem de espaço ocioso do contêiner, o número da iteração e o tempo que ocorreu a melhor solução.

3.6 – Algoritmo GRASP-3D

Nesta seção, apresentar-se-á o algoritmo GRASP-3D proposto para solução do problema de carregamento de um contêiner. Inicialmente, pretende-se descrever as variáveis envolvidas no algoritmo e em seguida, descreve-se o pseudocódigo do algoritmo.

Para melhor compreensão do algoritmo GRASP-3D, encontra-se no apêndice A o detalhamento do algoritmo apresentado em fluxograma.

3.6.1 – Variáveis envolvidas no algoritmo:

- b_i – item (caixa) *i* a ser utilizada no padrão de empacotamento, caracterizada pelo seu comprimento, largura e altura;
- q_i – quantidade de cada caixa;

- *contêiner* – contêiner a ser preenchido, caracterizado pelo seu comprimento, largura e altura;
- α - parâmetro que controla o grau de miopia e aleatoriedade da fase de construção;
- *maxiter* - número máximo de iterações definido como critério de parada;
- *S** – melhor solução encontrada dentro de todas as iterações;
- iter – parâmetro que controla o número de iterações;
- *S* – melhor solução encontrada dentro de uma iteração;
- *numcaixa* – número total de caixas a serem empacotadas;
- *C* – lista de candidatos composta pelas caixas posicionadas em ordem decrescente de volume;
- lista – parâmetro que controla a lista de caixas a ser utilizada para o preenchimento (lista=1 – LRC; lista=2 – $\overline{\mathrm{LRC}}$, lista=3 ou lista=4 – caixas rotacionadas);
- |C| – tamanho da lista de candidatos;
- estEspaco – parâmetro que controla o estoque de espaços (espaço ocioso do comprimento do contêiner);
- β – caixa de maior volume encontrada na lista de candidatos *C*;
- LRC – conjunto de lista restrita composta pelos melhores elementos da lista *C*;
- $\overline{\mathrm{LRC}}$ – conjunto de lista restrita composta pelos elementos restantes;
- largDisp – valor da largura disponível da camada atual;
- largConteiner – valor da largura do contêiner;
- restComprimento – valor do restante do comprimento da camada preenchida;
- volVazio – valor do volume vazio do espaço ocioso da altura ou do comprimento a ser preenchido;
- restAltura – valor do restante da altura da coluna preenchida;
- largCol – valor da largura da coluna preenchida;

- compCol – valor do comprimento da coluna preenchida;
- LCRestAlt – conjunto de lista de candidatos que caibam no espaço criado acima de cada coluna preenchida na camada;
- compVazio – valor do comprimento ocioso a frente das caixas;
- altConteiner – valor da altura do contêiner;
- LCRestComp – conjunto de lista de candidatos que caibam no espaço ocioso criado a frente das caixas da camada atual;

3.6.2 – Pseudocódigo do algoritmo GRASP-3D:

1. Entrar com os dados (b_i, q_i, *contêiner*, α, *maxiter*).
2. S* ← 0.
3. Calcular volume das caixas.
4. Para (iter = 1,2,3,...,maxiter) faça
5. Calcular o número total de caixas.

$$numcaixa = \sum q_i$$

6. S ← numcaixa.
7. Criar C ← (b_1, ... , b_m), posicionadas em ordem decrescente de volume.
8. Criar lista ← 1.

Fase de Construção

9. Enquanto ((|C| > 0) e (estEspaco > 0) e (lista > 0)) faça
10. Identificar a caixa de maior volume (β).

$$\beta = \max\{v(b_i); b_i \in C\}$$

11. Criar o conjunto LRC e $\overline{\text{LRC}}$.

$$\text{LRC} = \{b_i \in C / \alpha\beta \leq v(b_i) \leq \beta\}$$

$\overline{\text{LRC}}$ = caixas restantes.

12. Enquanto ((*numcaixa* > 0) e (estEspaco > 0) e (largDisp <= largConteiner) e (lista > 0)) faça
13. Se (lista=3) ou (lista=4), vá para o *Passo 14*. Senão, vá para o *Passo 20.*
14. Se (lista=3), restComprimento ← comprimento da camada atual. Senão, restComprimento ← estoque de espaços.
15. Verificar, nas listas restritas, se existe alguma caixa cuja dimensão de largura e de altura ocupe o restante do comprimento. Se existir, vá para o *Passo 16*, Senão, vá para o *Passo 17.*
16. Verificar se a dimensão de largura da caixa cabe na altura e o comprimento da caixa cabe no restante da largura da camada. Se for, rotacionar a caixa trocando suas dimensões de largura com comprimento. Vá para o *Passo 21*.
17. Verifique se existe alguma caixa cuja dimensão de largura ocupe o restante do comprimento e dimensão de comprimento ocupe o restante da largura da camada. Se existir, rotacionar a caixa trocando suas dimensões de largura com comprimento e vá para o *Passo 21*. Senão, vá para o *Passo 18*.
18. Verificar se existe alguma caixa cuja dimensão de altura ocupe o restante do comprimento. Se existir, rotacionar a caixa trocando suas dimensões de altura com comprimento e vá para o *Passo 21*. Senão, vá para o *Passo 19*.
19. lista ← 0. Considerar este espaço como espaço rejeitado (isto é, espaço que não pôde ser carregado com nenhum tipo de caixa). Vá para o *Passo 12.*
20. Se (lista=1), escolher uma caixa de LRC aleatoriamente. Senão, se (lista=2) escolher uma caixa de $\overline{\text{LRC}}$ aleatoriamente. Fazer a escolha de outra caixa da lista selecionada enquanto a largura da caixa for maior que o restante da largura ou se o comprimento da caixa for maior que o estoque de espaços.
21. Verificar se a quantidade disponível é suficiente para completar uma coluna. Se for suficiente, vá para *Passo 23*. Senão, para *Passo 22*.
22. Empacotar a quantidade completa. Vá para *Passo 26*.
23. Verificar se o número de colunas excede a largura disponível na camada. Se exceder, vá para *Passo 24*. Senão, vá para *Passo 25*.
24. Empacotar as colunas permitidas pela largura. Vá para *Passo 26*.

25. Empacotar tantas colunas completas quanto possíveis. Vá para *Passo 26*.
26. Atualizar as quantidades das caixas.
27. Atualizar as Listas Restritas cuja quantidade da caixa zerar.
28. Criar o espaço restante da altura e o espaço restante da largura.
29. Verificar, na lista LRC, se existe alguma caixa que preencha a largura restante. Se existir, lista ← 1 e vá para *Passo 20*. Senão, vá para *Passo 30*.
30. Verificar, na lista $\overline{\text{LRC}}$, se existe alguma caixa que preencha a largura restante. Se existir, lista ← 2 e vá para *Passo 20*. Senão, vá para *Passo 31*.
31. Verificar se a dimensão da largura restante é maior ou igual à mínima dimensão de comprimento de uma caixa das listas restritas. Se for, lista ← 3 e vá para *Passo 13*. Senão, vá para *Passo 32*.
32. Criar o espaço restante da largura.
33. Criar o espaço restante na direção da profundidade e acrescenta-lo ao estoque de espaços.
34. Criar o espaço não ocupado da camada atual e considera-lo como espaço rejeitado.
35. Criar o volume ocioso acima das caixas das colunas preenchidas.

volVazio = restAltura * largCol * compCol

36. 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, vá para o *Passo 37*. Senão, atualizar as listas restritas cuja quantidade de caixa zerar e vá para o *Passo 44.*
37. Criar o conjunto LCRestAlt.

LCRestAlt = $\left\{ b_i / v(b_i) < volVazio; b_i \in LRC \text{ ou } b_i \in \overline{LRC} \right\}$

38. Escolher uma caixa de LCRestAlt que ocupe ao máximo o comprimento da coluna atual.
39. Verificar se a quantidade de caixas disponíveis é suficiente para completar o volume ocioso. Se for suficiente, vá para *Passo 42*. Senão, para *Passo 32*.
40. Empacote tantas colunas completas quanto possíveis.

41. Atualizar as quantidades das caixas, o espaço rejeitado e o espaço no restante da altura. Vá para o *Passo 36*.

42. Verificar se há alguma coluna cujo comprimento seja menor que o comprimento da camada. Se há, vá para *Passo 43*. Senão, vá para *Passo 51*.

43. Criar o volume ocioso a frente das caixas da camada.

volVazio = compVazio * largCol * altConteiner

44. Verificar, nas listas restritas, se existe alguma caixa cujo volume ocupe o volume ocioso criado. Se existir, vá para o *Passo 45*. Senão, vá para o *Passo 51.*

45. Criar o conjunto LCRestComp.

LCRestComp = $\left\{ b_i / v(b_i) < volVazio; b_i \in LRC \text{ ou } b_i \in \overline{LRC} \right\}$

46. Escolher uma caixa de LCRestComp que ocupe ao máximo o comprimento da coluna atual.

47. Verificar se a quantidade de caixas disponíveis é suficiente para completar o volume ocioso. Se for suficiente, vá para o *Passo 48*. Senão, para o *Passo 51.*

48. Empacote tantas colunas completas quanto possíveis.

49. Atualizar as quantidades das caixas e o espaço rejeitado.

50. Atualizar as listas restritas cuja quantidade da caixa zerar.

51. Verificar se há alguma caixa a ser empacotada. Se há, vá para o *Passo 52*. Senão, vá para o *Passo 12*, pois “Toda a carga foi empacotada: FIM”.

52. Verificar se há algum espaço no estoque de espaços. Se há, vá para o *Passo 53.* Senão, vá para o *Passo 12*, pois “O problema não tem solução: FIM”.

53. Criar uma nova camada.

54. Verificar se existe alguma caixa da lista LRC cuja dimensão de comprimento ocupe o estoque de espaços. Se existir, lista $\leftarrow$ 1 e vá para o *Passo 20*. Senão, vá para o *Passo 55*.

55. Verificar se existe alguma caixa da lista $\overline{\text{LRC}}$ cuja dimensão de comprimento ocupe o estoque de espaços. Se existir, lista $\leftarrow$ 2 e vá para o *Passo 20*. Senão, vá para o *Passo 56*.

56. Verificar se existe alguma caixa das listas restritas cuja dimensão de largura ou de altura ocupe o estoque de espaços. Se existir, lista ← 4 e vá para o *Passo 13*. Senão, considerar este espaço como espaço rejeitado e "O problema não tem solução: FIM" e vá para o *Passo 12.*
57. Fim do Loop do *Passo 12*.
58. Verificar se a quantidade de cada caixa da lista de candidatos é maior que zero. Atualizar o tamanho da lista $|C|$.
59. Verificar se o número total de caixas é menor que S. Se for, S ← numcaixa.
60. Verificar se iter=1. Se for, S*←S. Senão, vá para o *Passo 9*.
61. Fim do Loop do *Passo 9*.
62. S ← número total de caixas que não foram utilizadas no preenchimento.
63. Verificar se a solução S é menor que S*. Se for, S*←S. Vá para o *Passo 4*.
64. Fim do Loop do *Passo 4*.
65. Escrever S*.

4 – RESULTADOS COMPUTACIONAIS

O algoritmo GRASP-3D apresentado na seção 3.6 foi implementado em linguagem C++, utilizando o compilador Borland C++ Builder versão 5.0 (ou DevC++ (Windows) – versão gratuita). Os testes foram realizados em um microcomputador pessoal equipado com processador Athlon XP 2.8+GHz e com memória RAM DDR 333 de 512MB, e sistema operacional Microsoft Windows XP.

A título de ilustração do algoritmo GRASP-3D, foi utilizado o exemplo real apresentado por George e Robinson (1980), que tem 8 tipos de caixa e total de 784 caixas. Suas características são apresentadas na Tabela 2 na seção 3.2.

Em seu trabalho, George e Robinson (1980) consideram o problema do carregamento de contêiner e desenvolveram um procedimento heurístico para colocar caixas de tamanhos diferentes em um contêiner. O procedimento de preenchimento do contêiner é feito construindo camadas ao longo do seu comprimento, e combinando espaços vazios entre camadas para aumentar a utilização de espaços. Não são impostas 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, embora a heurística possa ser adaptada para tratá-las.

Os resultados obtidos com o algoritmo GRASP-3D, foram testados com o parâmetro α assumindo valores 0.3, 0.5, 0.7 e 1.0. Esses valores influenciam no tamanho das listas restritas que serão utilizadas para o preenchimento do contêiner. Como critério de parada, caso não ocorra o preenchimento total das caixas dentro do contêiner, fica estabelecido o número máximo de iterações determinado pelo usuário, que para fins de testes foi estabelecido até 20000 iterações. Assim, a cada iteração executada pelo algoritmo é registrada a solução que apresente o menor número de caixas não utilizadas e a porcentagem de espaço ocioso no contêiner.

Quando os valores do exemplo real de George e Robinson são aplicados ao algoritmo GRASP-3D, os resultados obtidos são:

Tabela 3 – Resultados obtidos pelo Algoritmo GRASP-3D.

Nº máx. Iterações	*Alpha*	*Nº de caixas não alocadas*	*Espaço Ocioso*	*Tamanho de C*								*Iteração*	*Tempo (seg.)*
				1	*2*	*3*	*4*	*5*	*6*	*7*	*8*		
500	0.3	5	10,67%	0	5	0	0	0	0	0	0	5	0,06
	0.5	4	10,52%	0	3	0	0	0	1	0	0	481	0,16
	0.7	4	10,56%	0	4	0	0	0	0	0	0	42	0,05
	1.0	4	10,56%	0	4	0	0	0	0	0	0	137	0,08
2000	0.3	5	10,67%	0	2	0	0	0	0	0	0	459	0,16
	0.5	4	10,52%	0	3	0	0	0	1	0	0	53	0,06
	0.7	4	10,56%	0	4	0	0	0	0	0	0	39	0,05
	1.0	5	10,67%	0	5	0	0	0	0	0	0	245	0,12
10000	0.3	1	10,23%	0	1	0	0	0	0	0	0	1501	0,36
	0.5	5	10,52%	0	1	0	0	0	4	0	0	548	0,19
	0.7	4	10,56%	0	4	0	0	0	0	0	0	326	0,11
	1.0	0	10,01%	0	0	0	0	0	0	0	0	5293	0,06
20000	0.3	0	10,01%	0	0	0	0	0	0	0	0	1792	0,39
	0.5	4	10,52%	0	3	0	0	0	1	0	0	1812	0,42
	0.7	4	10,56%	0	4	0	0	0	0	0	0	443	0,16
	1.0	0	10,01%	0	0	0	0	0	0	0	0	1971	0,42

Fonte: Tabela do autor

Na Tabela 3, são apresentadas as seguintes informações:

- **Nº máx. iterações** – número máximo de iterações que o algoritmo irá executar.
- **Alpha** – parâmetro que define o tamanho das listas restritas que serão utilizadas no preenchimento do contêiner.

• **Nº de caixas não alocadas** – número de caixas que sobraram fora do contêiner.

• **Espaço Ocioso** – valor percentual do espaço que não foi preenchido no contêiner.

• **Tamanho de C** – tamanho das listas de candidatos apresentando a quantidade de cada tipo de caixa que não foi alocada no contêiner.

• **Iteração** – número da iteração que obteve a melhor solução.

• **Tempo** – tempo de execução em segundos para obter a referente solução.

Os resultados produzidos com a execução do algoritmo GRASP-3D demonstram ser satisfatórios, pois em sua maioria apresentam como solução 4 ou 5 caixas não alocadas, e com bons valores percentuais do espaço ocioso.

É importante ressaltar que, em alguns casos, o valor atribuído ao parâmetro α foi decisivo, influenciando o resultado final, conseguindo alcançar o objetivo de empacotar todas as caixas.

Pode-se observar pelos resultados obtidos na Tabela 3, que ocorreu o empacotamento de todas as caixas quando o alpha assume o valor 0.5 ou o valor 1.0, isto é, quando todas as caixas se encontram numa mesma lista.

Os tempos computacionais gerados nos testes realizados com o algoritmo GRASP-3D foram bons, considerando o número de iterações, conforme mostra a Tabela 3.

Para melhor avaliação do comportamento do algoritmo GRASP-3D foram feitos outros testes com dados gerados aleatoriamente. Esses dados formam tabelas com 8 (oito) tipos de caixas cujas dimensões de comprimento, largura, altura e quantidade admitem valores aleatórios dentro de uma escala definida. Das tabelas geradas, existem aquelas em que o volume total de caixas cabe dentro do contêiner de medidas já mencionadas na seção 3.2, e existem outras em que o volume ultrapassa as medidas do contêiner. As tabelas geradas encontram-se no Anexo A.

Quando os valores das tabelas geradas são aplicados ao algoritmo GRASP-3D assumindo o máximo de 20000 iterações, os resultados obtidos são:

Tabela 4 – Resultados do Algoritmo GRASP-3D com dados aleatórios para 8 tipos de caixas.

Dados	*Alpha*	*Nº de caixas não alocadas*	*Espaço Ocioso (%)*	*Tamanho de C*								*Iteração*	*Tempo (seg.)*
				1	*2*	*3*	*4*	*5*	*6*	*7*	*8*		
DA1	0.3	0	34,03	0	0	0	0	0	0	0	0	1	0,03
	0.5	0	32,02	0	0	0	0	0	0	0	0	1	0,05
	0.7	0	32,91	0	0	0	0	0	0	0	0	1	0,03
	1.0	0	33,15	0	0	0	0	0	0	0	0	2	0,03
DA2	0.3	15	13,58	12	0	0	2	1	0	0	0	344	0,36
	0.5	25	15,53	2	0	5	12	1	0	5	0	14441	0,53
	0.7	47	17,82	0	5	0	0	0	42	0	0	111	0,06
	1.0	28	16,39	10	0	1	0	1	2	4	0	7909	0,39
DA3	0.3	0	15,56	0	0	0	0	0	0	0	0	1	0,03
	0.5	0	16,06	0	0	0	0	0	0	0	0	1	0,03
	0.7	0	14,94	0	0	0	0	0	0	0	0	2	0,03
	1.0	0	15,87	0	0	0	0	0	0	0	0	1	0,03
DA4	0.3	69	21,30	67	0	0	0	0	2	0	0	429	0,11
	0.5	63	18,34	61	0	0	2	0	0	0	0	89	0,06
	0.7	105	17,05	0	0	2	8	2	16	0	77	1009	0,20
	1.0	105	17,05	0	0	2	8	2	16	0	77	192	0,06

Fonte: Tabela do autor

Na Tabela 4, tem-se resultados obtidos com o algoritmo GRASP-3D para dados gerados aleatoriamente para 8 tipos de caixas. As tabelas DA1, DA2 e DA3 contêm caixas cujos volumes totais cabem no contêiner, e a tabela DA4 contém caixas cujos volumes totais não cabem no contêiner.

É possível observar, que se obtiveram bons resultados para outros tipos de tabelas, podendo assim trabalhar com outros exemplos reais. Mesmo quando aumenta o número de tipos de caixas, é possível obter resultados satisfatórios, como mostra as Tabelas 5 e 6.

Tabela 5 – Resultados do Algoritmo GRASP-3D com dados aleatórios para 12 tipos de caixas.

Dados	*Alpha*	*Nº de caixas não alocadas*	*Espaço Ocioso (%)*	*Tamanho de C*												*Iteração*	*Tempo (seg.)*
				1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*		
DA5	0.3	0	16,80	0	0	0	0	0	0	0	0	0	0	0	0	223	0,11
	0.5	0	16,80	0	0	0	0	0	0	0	0	0	0	0	0	107	0,09
	0.7	0	16,42	0	0	0	0	0	0	0	0	0	0	0	0	262	0,12
	1.0	0	16,70	0	0	0	0	0	0	0	0	0	0	0	0	1456	0,34
DA6	0.3	35	12,42	10	17	0	0	0	0	7	7	0	3	0	0	11174	0,44
	0.5	46	14,50	0	39	2	0	0	0	0	0	0	0	5	0	9793	0,17
	0.7	44	11,51	0	0	0	0	0	5	0	0	0	0	3	36	14082	0,05
	1.0	30	11,13	14	0	0	0	6	0	7	0	0	3	0	0	1282	0,33
DA7	0.3	0	15,50	0	0	0	0	0	0	0	0	0	0	0	0	301	0,12
	0.5	0	15,03	0	0	0	0	0	0	0	0	0	0	0	0	133	0,09
	0.7	0	15,27	0	0	0	0	0	0	0	0	0	0	0	0	55	0,08
	1.0	0	15,74	0	0	0	0	0	0	0	0	0	0	0	0	80	0,06
DA8	0.3	65	11,87	0	0	0	37	2	20	0	0	0	2	4	0	5771	0,14
	0.5	65	11,70	0	0	0	37	12	10	0	0	0	2	4	0	616	0,19
	0.7	65	11,87	0	0	0	37	2	20	0	0	0	2	4	0	6959	0,36
	1.0	54	13,55	5	0	49	0	0	0	0	0	0	0	0	0	8	0,03

Fonte: Tabela do autor

A Tabela 5 apresenta os resultados obtidos com o algoritmo GRASP-3D para dados gerados aleatoriamente para 12 tipos de caixas. As tabelas DA5, DA6 e DA7 contêm caixas cujos volumes totais cabem no contêiner, e a tabela DA8 contém caixas cujos volumes totais não cabem no contêiner.

Tabela 6 – Resultados do Algoritmo GRASP-3D com dados aleatórios para 20 tipos de caixas.

Dados	*Alpha*	*Nº de caixas não alocadas*	*Espaço Ocioso (%)*	*Tamanho de C*	*Iteração*	*Tempo (seg.)*
DA9	0.3	0	19,63	0	2567	0,83
	0.5	0	18,87	0	3646	0,11
	0.7	0	18,10	0	1186	0,45
	1.0	0	31,14	0	86	1,14

Fonte: Tabela do autor

Os resultados apresentados na Tabela 6 foram obtidos com o algoritmo GRASP-3D para dados gerados aleatoriamente para 20 tipos de caixas, sendo que a tabela DA9 contém caixas cujos volumes totais cabem no contêiner.

5 – ANÁLISE DOS RESULTADOS E CONCLUSÃO

5.1 – Análise dos Resultados

A fim de validar o algoritmo proposto, são comparados os valores fornecidos nos testes computacionais com os métodos de refinamentos da heurística de George e Robinson (1980) propostos por Cecílio e Morabito (2004).

A partir da heurística de George e Robinson, Cecílio e Morabito definiram seis métodos de solução (original mais cinco novas heurísticas) para o problema, conforme Tabela 3. Os métodos de solução são compostos por refinamento proposto para a heurística e das duas novas versões da heurística (versão Arranjo e versão Camada). São eles:

Tabela 7 – Métodos de solução propostos por Cecílio e Morabito.

Método	***Heurística***
1	George e Robinson original
2	Método 1 + refinamento proposto
3	Método 1 + versão Arranjo
4	Método 1 + versão Arranjo + refinamento
5	Método 1 + versão Camada
6	Método 1 + versão Camada + refinamento

Fonte: CECÍLIO E MORABITO, 2004

A solução publicada em George e Robinson (1980) carregou apenas 783 caixas (89,7% do volume do contêiner), deixando de fora uma caixa do tipo 7. Os resultados obtidos pelos 6 métodos propostos estão descritos na Tabela 5.

Tabela 8 – Resultados obtidos pelos métodos propostos por Cecílio e Morabito.

Método	*Solução (espaço ocupado)*	*Nº de Caixas*
1	0,8974	783
2	0,8988	784
3	0,8988	784
4	0,8988	784
5	0,8988	784
6	0,8988	784

Fonte: CECÍLIO E MORABITO, 2004

Analisando os resultados obtidos da Tabela 3 e comparando com a Tabela 5, verifica-se, conforme mostra a Tabela 6, que foi possível alocar todas as caixas dentro de um contêiner obtendo 89,99% do volume do contêiner, quando o alpha assume o valor 0.3 ou 1.0.

Tabela 9 – Comparativo do resultado obtido.

Método	*Solução*	*Nº de Caixas*
George e Robinson	0,8974	783
Cecílio e Morabito	0,8988	784
GRASP-3D	0,8999	784

Fonte: Tabela do autor

Baseado nas Tabelas 4, 5 e 6 da seção anterior, é possível constatar a performance do algoritmo GRASP-3D para o problema de carregamento de contêiner, pois obteve bons resultados com os dados gerados aleatoriamente, bem como, com o exemplo comparativo proposto acima.

5.2 – Conclusão

Com base na literatura consultada é possível concluir que as metodologias utilizadas para problemas de empacotamento tridimensionais, como por exemplo, preenchimento de contêineres com itens, previamente, com dimensões e características definidas, começaram a ter grande sucesso a partir do momento que passaram a resolver tais problemas utilizando as meta-heurísticas.

A pesquisa apresentada propõe solucionar o problema preenchimento de contêiner, utilizando GRASP, capaz de apresentar soluções de boa qualidade. É relevante destacar que, até o momento, não há nenhum estudo que envolva aplicação desta metodologia para resolução de problemas que demandam otimização da ocupação dos espaços tridimensionais, o que vem ratificar a importância desta pesquisa, embora se reconheça à necessidade de efetiva realização de estudos mais consistentes para melhor refinamento do algoritmo proposto, com propósito de obter melhores resultados.

Para o exemplo, retirado da literatura, o resultado produzido com a execução do GRASP-3D foi quase equiparado com o resultado apresentado pelos outros métodos. E demonstra que a GRASP pode ser aplicada com sucesso em problemas de carregamento de contêiner.

Quanto aos exemplos gerados aleatoriamente, o GRASP-3D mostrou-se bastante eficaz. Os resultados apresentados mostram que é possível empacotar todas as caixas com boa porcentagem de espaço ocioso dentro do contêiner, 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, conclui-se que é interessante a utilização da GRASP para o problema de carregamento de contêiner, dando retorno suficientemente aceitável; e demonstram que a idéia do uso da GRASP nesse tipo de problema é promissor mesmo que ainda se tem espaço para amadurecimento e novos refinamentos.

Apesar de julgar-se que este trabalho tenha produzido avanço significativo na modelagem do problema de carregamento de contêiner, esse campo continua promissor para aperfeiçoamentos e novas pesquisas.

Como trabalhos subseqüentes sugerem:

• Desenvolver diferentes aplicações considerando características não levantadas do Problema de Carregamento de Contêiner abordado nesta dissertação – como estabilidade da carga e carregamento paralelo de diversos contêineres –, ou ainda, tratando de outros Problemas de Corte e Empacotamento.

• Aprimorar os resultados aqui obtidos – em termos de qualidade da solução e/ou tempo de processamento – estudando novas heurísticas.

• Criar uma ferramenta computacional com interface gráfica simples e funcional voltada para ambiente de produção.

• Estender a aplicação da meta-heurística GRASP a outros tipos de problemas.

REFERÊNCIAS BIBLIOGRÁFICAS

ARENALES, M.N., MORABITO, R., YANASSE, H.H. (2004). O Problema de Corte e Empacotamento. *Livro-texto de Mini Curso, XXXVI SBPO, São José Del Rei - MG.*

BISCHOFF, E.E., RATCLIFF, (1995). Issues in the Development of Approaches to Container Loading. *Omega International Journal of Management Science,* v. 23, p.377-390.

BISCHOFF, E.E. (2003). Dealing With Load Bearing Strength Considerations in Container Loading Problems. EBMS – European Business Management School – *Working Paper Series 2003. Disponível em: School of Business and Economics/2003/3. Acesso em: 04/12/2006.*

CEAC (1999). Cortes e Empacotamento Assistidos por Computador. *Disponível em: http://www.lac.inpe.br/po/projects/ceac/fapesp3.html. Acesso em: 27/07/2006.*

CECÍLIO, F.O., MORABITO, R. (2001). Otimização do Problema do Carregamento de Carga dentro de Contêineres. *Anais em CD-ROM: IV Simpósio de Pesquisa Operacional e V Simpósio de Logística da Marinha (SPOLM),* v.1, p.1-10.

CECÍLIO, F.O., MORABITO, R. (2004). Refinamentos na Heurística de George e Robinson para o Problema de Carregamento de Caixas dentro de Contêineres. *Transportes,* v.12, n.1, p. 32-45.

CHEN, C.S., LEE, S.M., SHEN, Q.S. (1995). An Analytical Model for the Container Loading Problem. *European Journal of Operational Research*, v.80, p.68-76.

CONSTANTINO, A.A., GOMES JÚNIOR, A.M. (2002). Um algoritmo Genético Híbrido para o Problema de Corte Industrial Bidimensional. *Acta Scientinum. Maringá*, v.24, n.6. p.1727-1731.

CORCORAN, L., WAINWRIGHT, R.L. (1992). A Genetic Algorithm for Packing in Three Dimensions. *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing,* p.1021-1030.

CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., STEIN, C. (2002). *Algoritmos: teoria e prática.* 1ª edição. Rio de Janeiro: Campus. 919p.

DYCKHOFF, H. (1990). A Typology of Cutting and Packing Problems. *European Journal of Operational Research,* v.44, p.145-159.

DYCKHOFF, H., SHEITHAUER, G., TERNO (1997). *Cutting and Packing*. Annoted Bibliographies in Combinatorial Optimization [edited by M. Amico, F. Maffioli and S. Martello], John Wiley & Sons, New York, p.393-414.

FEO, T. A., RESENDE, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Jornal of Global Optimization,* v.6, p.109-133.

GEORGE, J.A., ROBINSON, D.F. (1980). A heuristic for Packing Boxes into a Container. *Computers and Operational Research,* v.7, p.147-156.

GILMORE, P., GOMORY, R. (1961). A Linear Programming Approach to the Cutting Stock Problem. *Operattions Research,* v.9, p.849-859.

KLEIN, R. (2006). *Biblioteca Java para Solucionar o Problema de Carregamento de Contêineres.* Trabalho de Conclusão de Curso (Graduação em ciência da Computação) – Centro Universitário Feevale.

MARTELLO, S., PISINGER, D., TOTH, P. (1997). New Trends in exact Algorithms for the 0-1 Knapsack Problem. J. Barcelo Editor, Proceedings of EURO/INFORMS-97, Barcelona, p.151-160. *Disponível em: http://citeseer.ist.psu.edu/correct/143778. Acessado em: 27/07/2006.*

MARTELLO, S., PISINGER, D., VIGO, D. (2000). The Three-dimensional Bin Packing Problem. *Operations Research,* v.48, p.256-267.

MORABITO, R., ARENALES, M. (1994). An And/Or-graph Approach to the Container Loading Problem. *International Transactions in Operations Research,* v.1, n.1, p.59-73.

MORABITO, R., ARENALES, M. (1997). Abordagens para o Problema do Carregamento de Contêineres. *Pesquisa Operacional,* v.17, n.1, p.29-56.

MORIOKA, S.A., RONCONI, D.P. (2002). Estudo de Heurísticas para a Resolução do Problema de Carregamento de Paletes do Distribuidor. *Trabalhos apresentados, São Paulo: EPUSP.*

PERIN, C., GOMES, V.P., MORETTI, A.C. (2003). Heurísticas para Empacotamento em Containers. *Relatório de Pesquisa: RP57/03 – IMECC;Disponível: http://www.ime.unicamp.br/rel_pesq/2003/relatorio03.html. Acesso em: 22/04/2005.*

PISINGER, D. (2002). Heuristics for the Container Loading Problem. *European Journal of Operation Research,* v.141, p. 382-392.

PISINGER, D., FARØE, O., ZACHARIASEN, M. (1999). Guided Local Search for the Three-dimensional Bin Packing Problem. *Techinical Report, DIKU 99-13, Københavns Universitet, Datalogisk institut.*

POLDI, K.C., ARENALES, M.N. (2003). *Algumas Extensões do Problema de Corte de Estoque.* Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação – ICMC-USP – São Carlos-SP.

RESENDE, M.G.C. (1998). Greedy Randomizes Adaptive Search Procedures (GRASP). *AT&T Labs Research Technical Report:* 98.41.1.

SHEITHAUER, G. (1992). Algorithms for the Container Loading Problem. *Operations Research Proceedings, 1991, Springer, Berlin, 1992,* p. 445-452.

SILVA, J.L.C., SOMA, N.Y. (2003). Um Algoritmo Polinomial para o Problema de Empacotamento de Contêineres com Estabilidade Estática de Carga. *Pesquisa Operacional,* v.23, n.1, p. 79-98.

SILVEIRA, C.M.D., TOSCANI, L.V. (1999). Análise de Métodos Heurísticos de Característica Gulosa. *IV Semana Acadêmica do PPGC (Programa de Pós-Graduação em Computação) – UFRGS.*

SOUZA, M. J. F. (2006). Inteligência Computacional para Otimização. Notas de aula da disciplina Inteligência Computacional para Otimização (2006/2). *DECOM/ICEB/UFOP, Ouro Preto. Disponível:*

http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm. Acesso em: 07/10/2006.

VIEIRA NETO, E. (2004). *GRASP: efeito da independência das soluções iniciais na otimização do corte unidimensional.* Tese (Doutorado em Ciências de Engenharia) – Centro de Ciências e Tecnologia, Laboratório de Engenharia de Produção – LEPROD, UENF, Campos dos Goytacazes-RJ.

VELASCO, A.S. (2005). *Uma GRASP para o Problema de Corte Bidimensional Guilhotinado e Restrito.* Dissertação (Mestrado em Ciências de Engenharia) – Centro de Ciências e Tecnologia, Laboratório de Engenharia de Produção – LEPROD, UENF, Campos dos Goytacazes-RJ.

VERGARA, S.C. (2004). *Projetos e Relatórios de Pesquisa em Administração*. 5ª edição. São Paulo: Editora Atlas.

APÊNDICE A – Fluxograma Detalhado do Algoritmo GRASP-3D

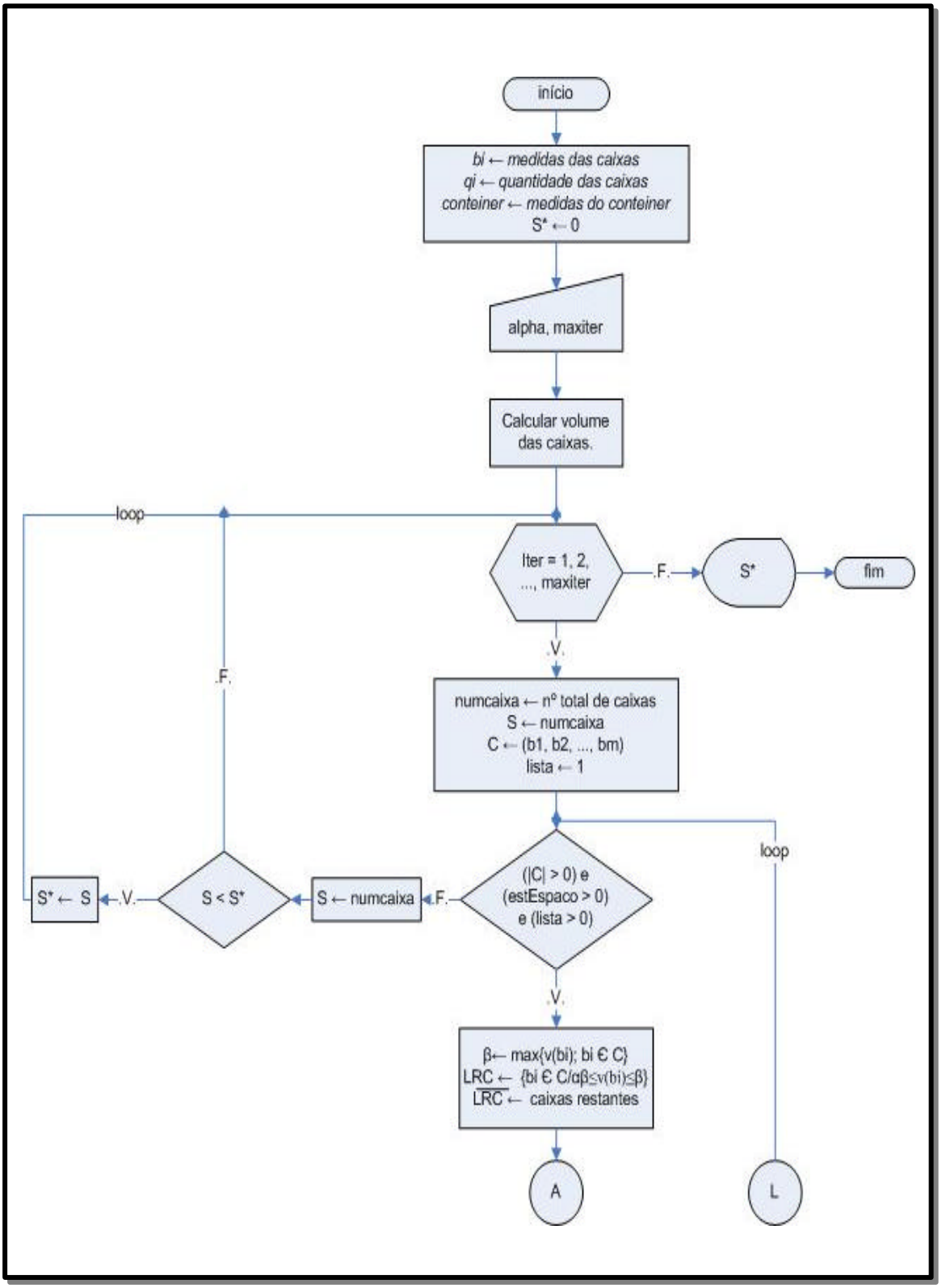

Figura 16 – Fluxograma GRASP-3D Parte 1.

Fonte: Figura do autor

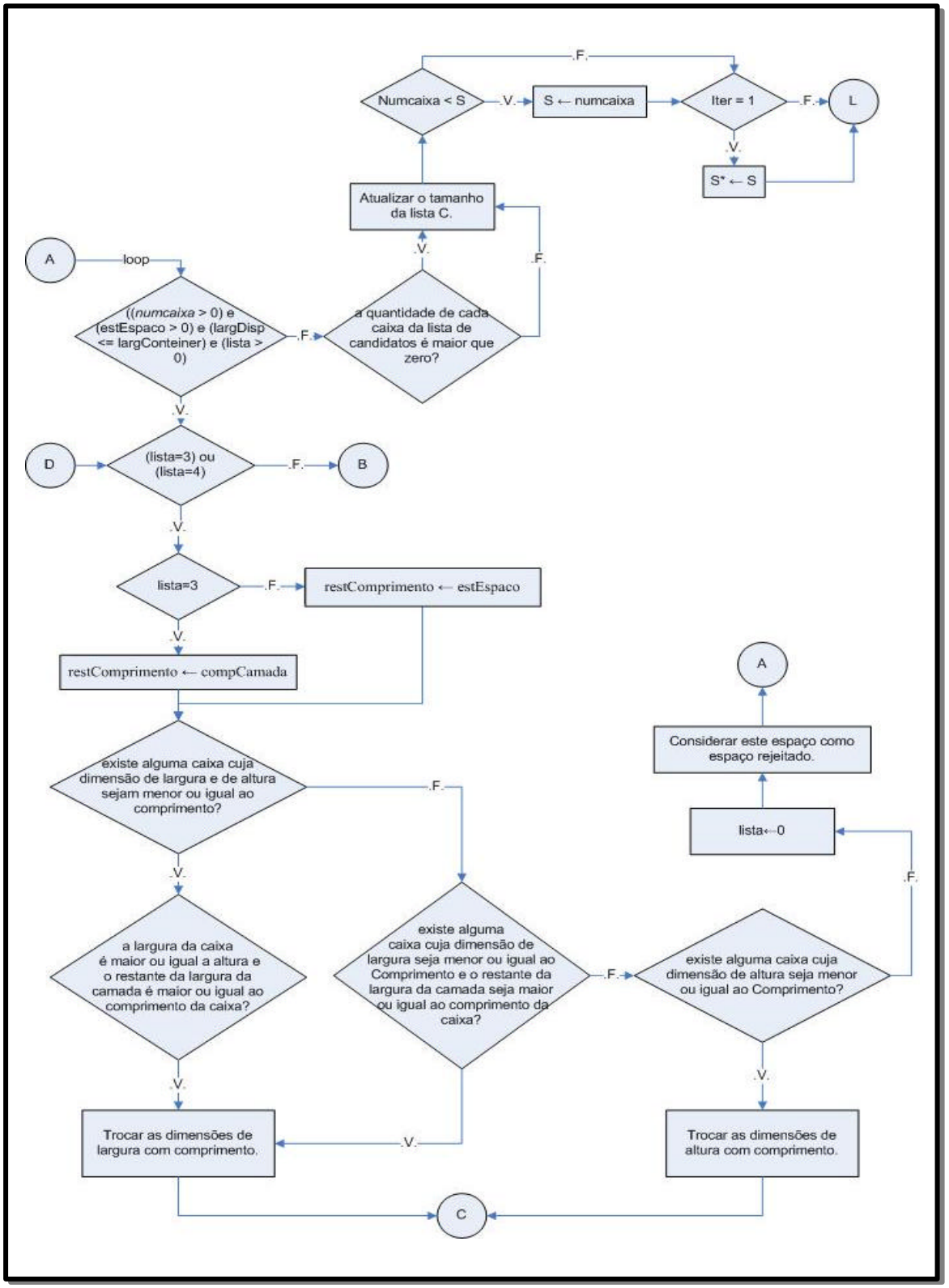

Figura 17 – Fluxograma GRASP-3D Parte 2.

Fonte: Figura do autor

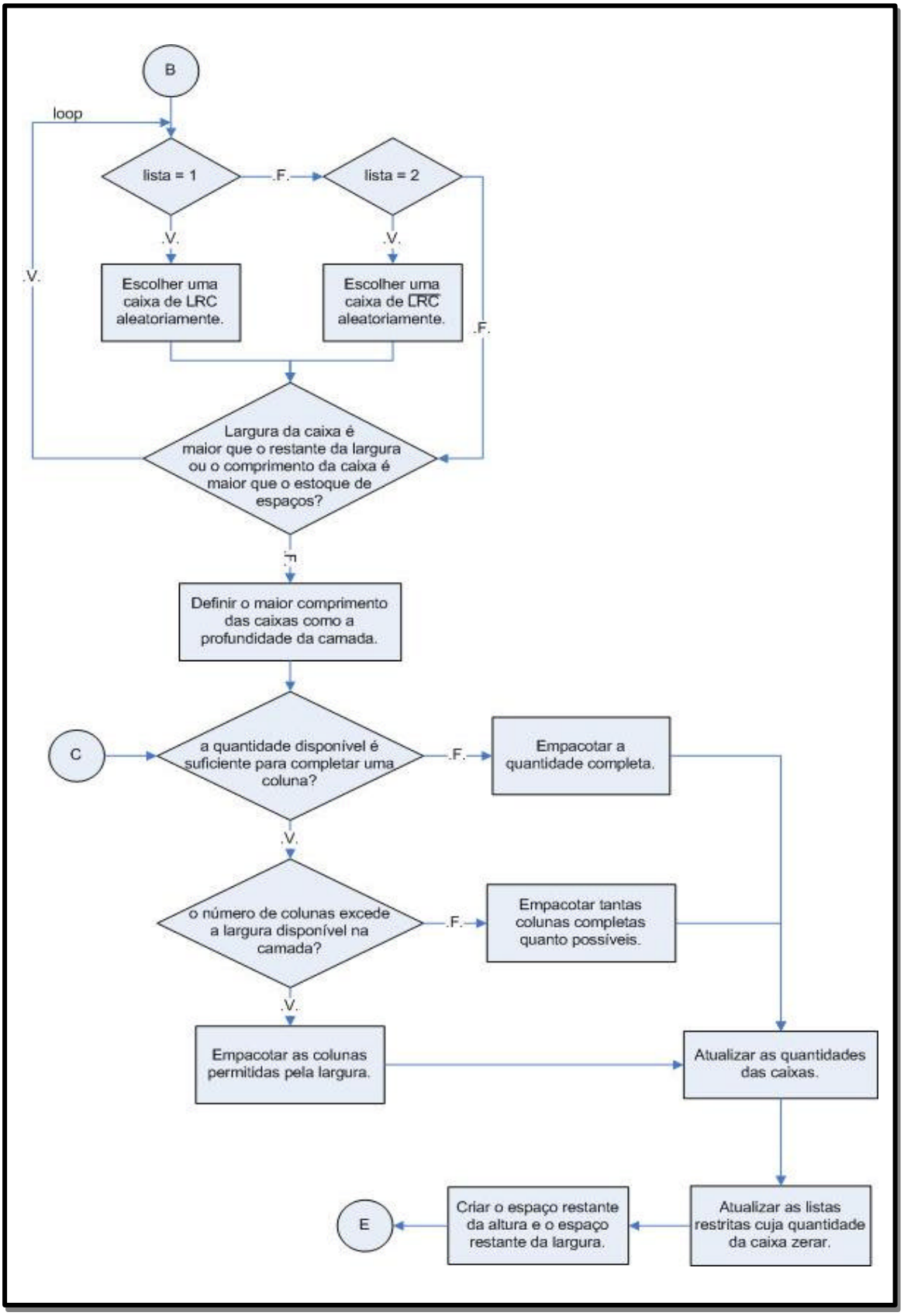

Figura 18 – Fluxograma GRASP-3D Parte 3.

Fonte: Figura do autor

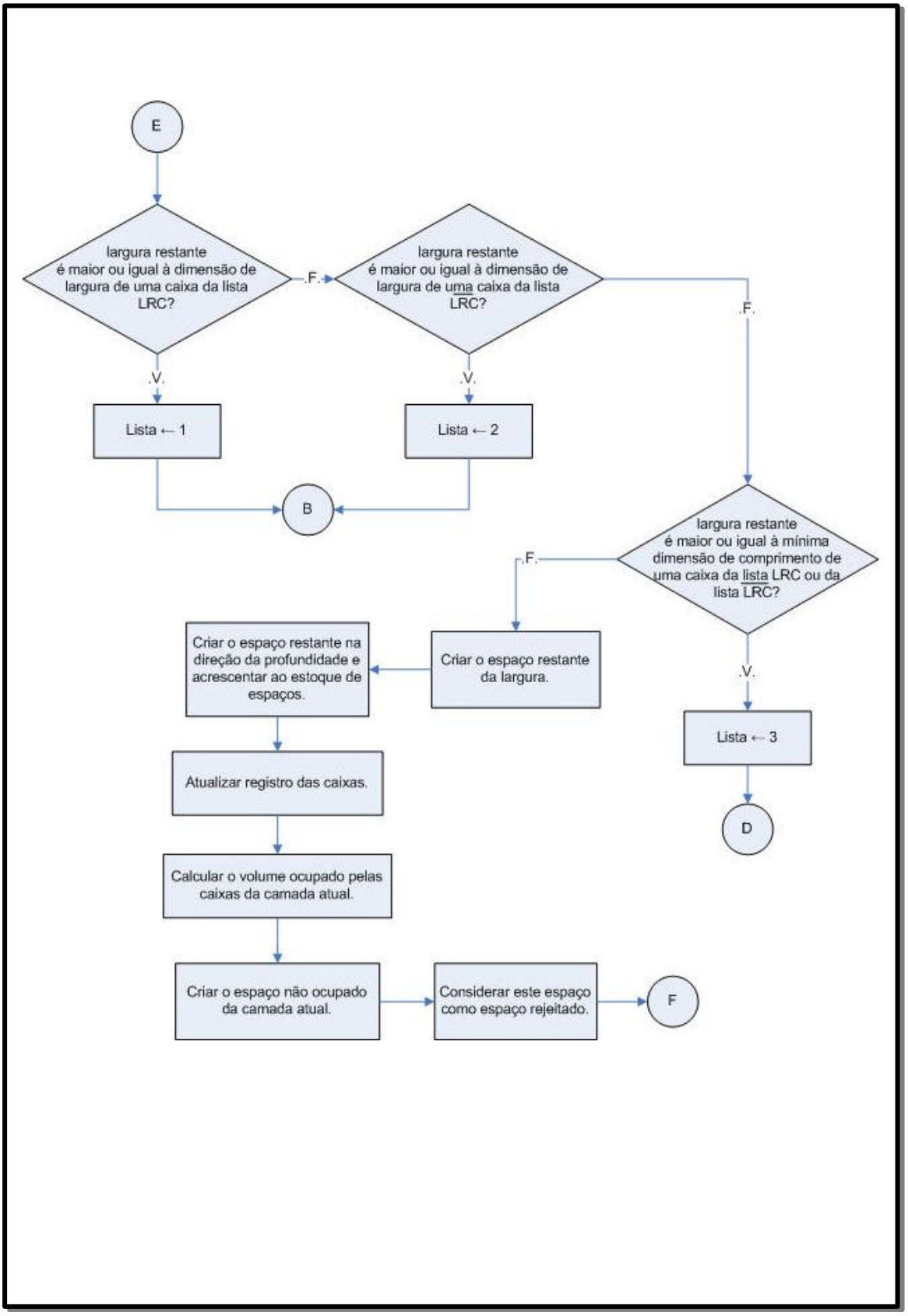

Figura 19 – Fluxograma GRASP-3D Parte 4.

Fonte: Figura do autor

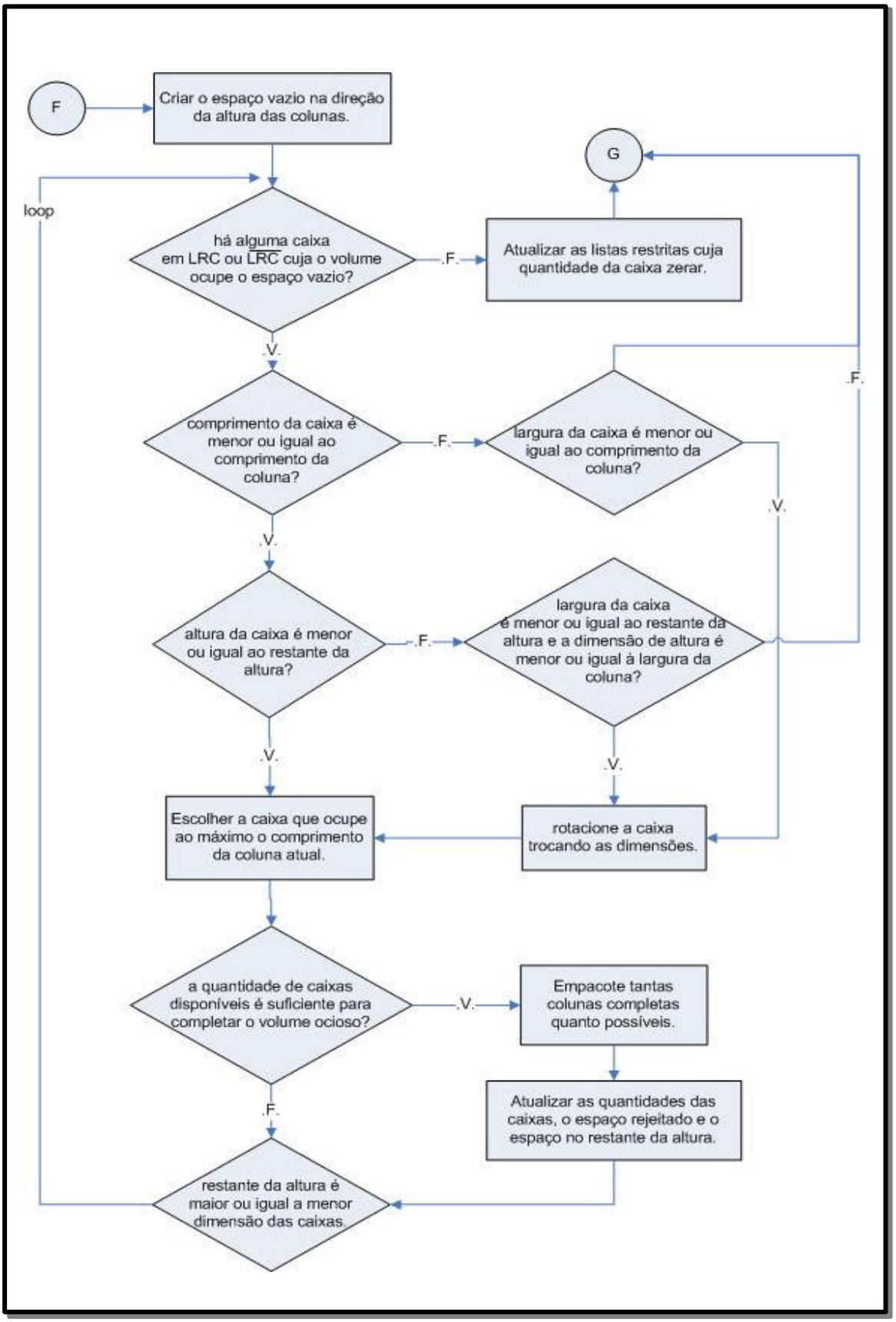

Figura 20 – Fluxograma GRASP-3D Parte 5.

Fonte: Figura do autor

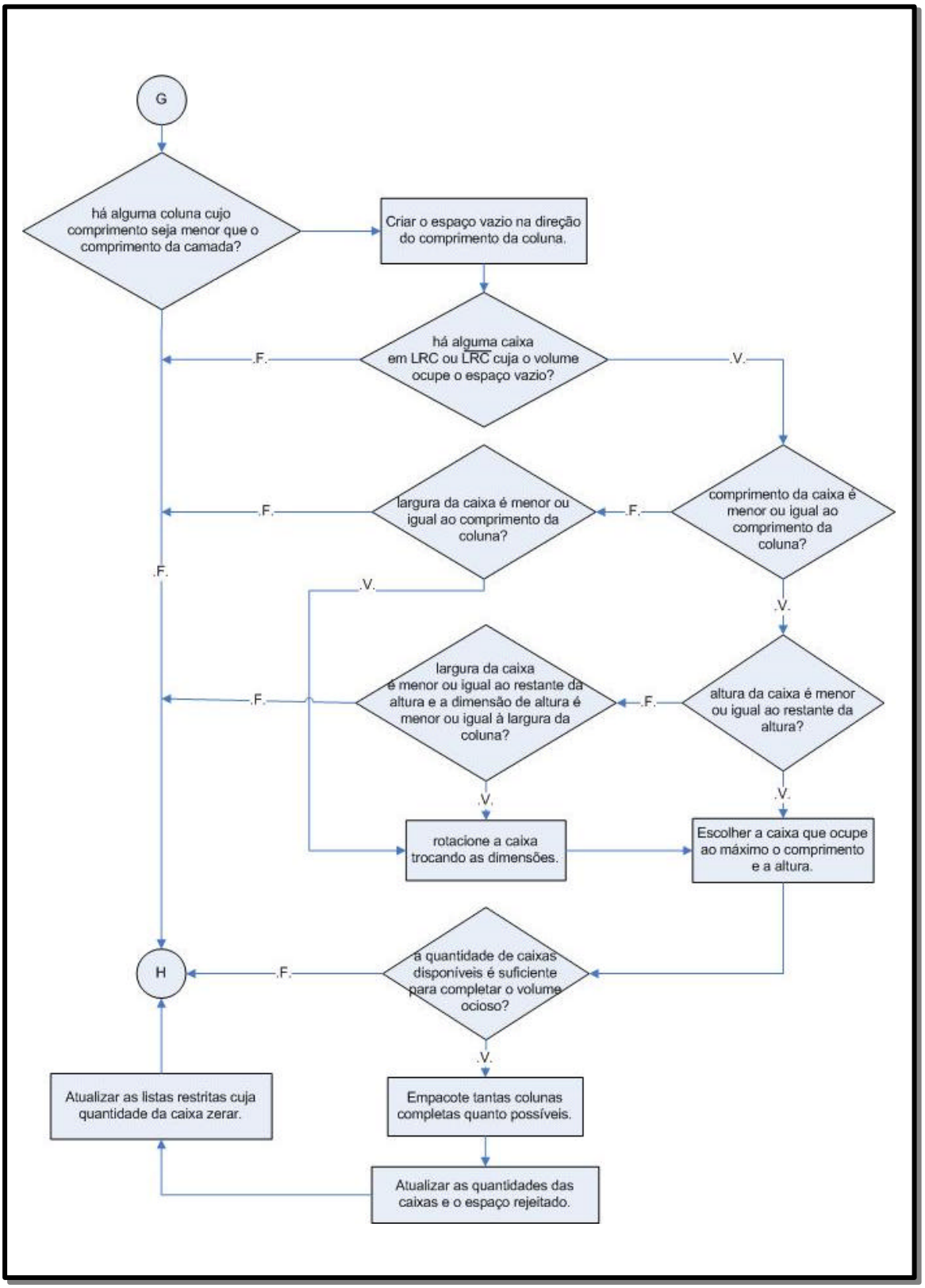

Figura 21 – Fluxograma GRASP-3D Parte 6.

Fonte: Figura do autor

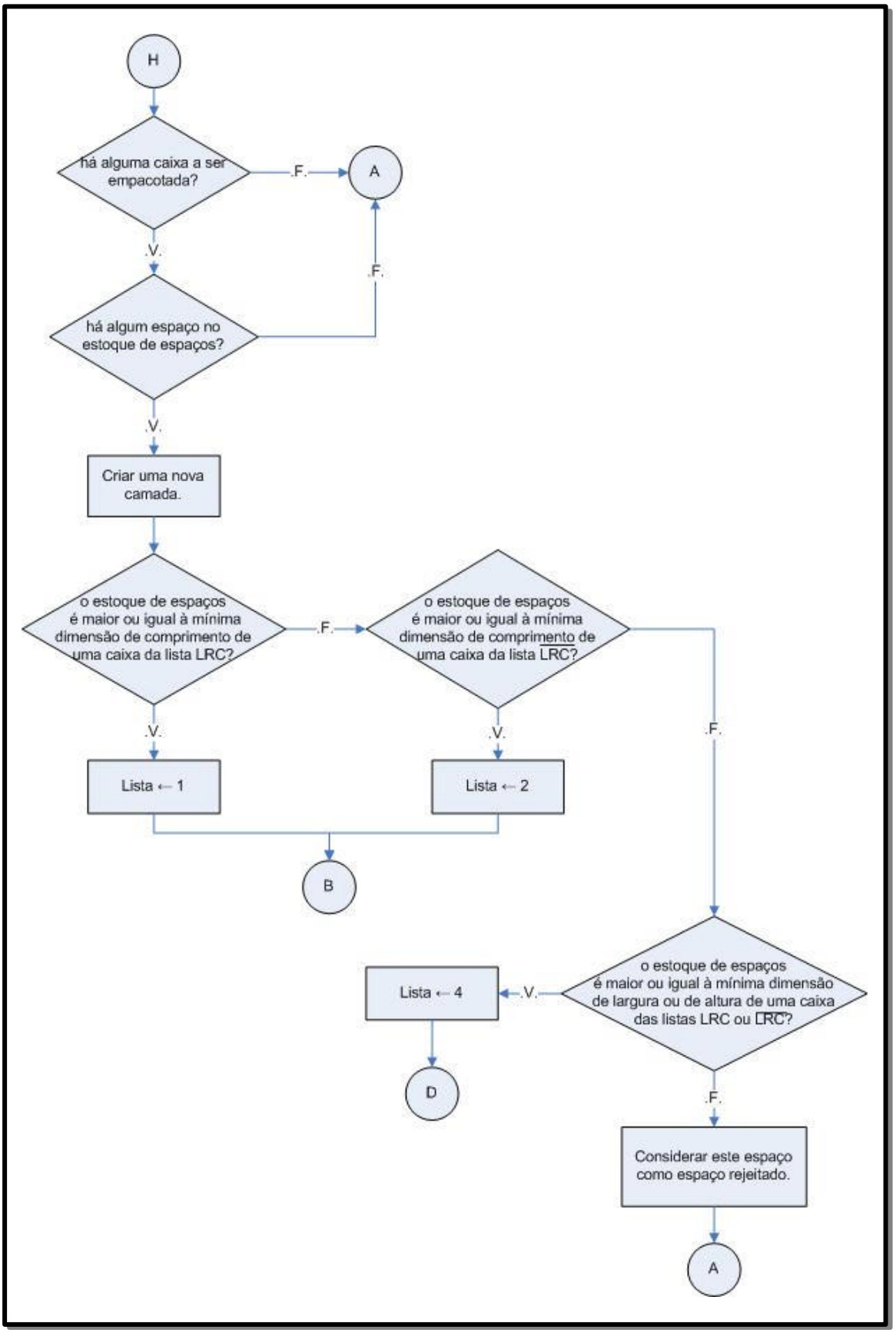

Figura 22 – Fluxograma GRASP-3D Parte 7.

Fonte: Figura do autor

APÊNDICE B – Tabelas de Dados Gerados Aleatoriamente

A seguir, as tabelas apresentarão os dados gerados aleatoriamente. Estas tabelas possuem as seguintes características gerais:

- Comprimento: $300 \leq c_i \geq 1500$
- Largura: $100 \leq l_i \geq 300$
- Altura: $100 \leq h_i \geq 300$
- Quantidade: $1 \leq q_i \geq 200$

DA1

Tabela 10 – Dados Aleatórios com 306 caixas.

i	c_i	l_i	h_i	q_i
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
				306 *caixas*
			Volume Total	15,919 m^3

Fonte: Tabela do Autor

DA2

Tabela 11 – Dados Aleatórios com 453 caixas.

i	c_i	l_i	h_i	q_i
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
				453 *caixas*
			Volume Total	26,593 m^3

Fonte: Tabela do Autor

DA3

Tabela 12 – Dados Aleatórios com 679 caixas.

i	c_i	l_i	h_i	q_i
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
				679 *caixas*
			Volume Total	26,205 m^3

Fonte: Tabela do Autor

DA4

Tabela 13 – Dados Aleatórios com 471 caixas.

i	c_i	l_i	h_i	q_i
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
				471 *caixas*
			Volume Total	31,940 m^3

Fonte: Tabela do Autor

As próximas tabelas possuem as seguintes características gerais:

• Comprimento: $300 \leq c_i \geq 1000$

• Largura: $100 \leq l_i \geq 200$

• Altura: $100 \leq h_i \geq 200$

• Quantidade: $1 \leq q_i \geq 100$

DA5

Tabela 14 – Dados Aleatórios com 614 caixas.

i	c_i	l_i	h_i	q_i
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
				614 *caixas*
			Volume Total	24,636 m^3

Fonte: Tabela do Autor

DA6

Tabela 15 – Dados Aleatórios com 785 caixas.

i	c_i	l_i	h_i	q_i
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
				785 *caixas*
			Volume Total	27,372 m^3

Fonte: Tabela do Autor

DA7

Tabela 16 – Dados Aleatórios com 661 caixas.

i	c_i	l_i	h_i	q_i
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
				661 *caixas*
			Volume Total	24,637 m^3

Fonte: Tabela do Autor

DA8

Tabela 17 – Dados Aleatórios com 458 caixas.

i	c_i	l_i	h_i	q_i
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
				458 *caixas*
			Volume Total	33,275 m^3

Fonte: Tabela do Autor

DA9

Tabela 18 – Dados Aleatórios com 930 caixas.

i	c_i	l_i	h_i	q_i
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
				930 *caixas*
			Volume Total	23,678 m^3

Fonte: Tabela do Autor

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo