ESCOLA
POLITÉCNICA DE
PERNAMBUCO

ROTEAMENTO DE VIAS E EMPACOTAMENTO 3D: SOLUÇÕES COMPUTACIONAIS PARA A LOGÍSTICA MODERNA

Trabalho de Conclusão de Curso

Engenharia da Computação

José Sandro Rogério Mendes de Araújo

Orientador: Prof. Sérgio Murilo Maciel Fernandes

JOSÉ SANDRO ROGÉRIO MENDES DE ARAÚJO

ESCOLA
POLITÉCNICA DE
PERNAMBUCO

ROTEAMENTO DE VIAS E EMPACOTAMENTO 3D: SOLUÇÕES COMPUTACIONAIS PARA A LOGÍSTICA MODERNA

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro de 2010.

Dedico este trabalho à minha família, onde todos se empenharam para o meu progresso, em especial à minha mãe Fátima Mendes, minha avó Júlia Mendes(in memorian), meu avô João José(in memorian), dedico ao meu pai Romildo José e ao meu irmão Saulo Mendes. A todos dedico este e todos os futuros trabalhos da minha vida.

Agradecimentos

Agradeço em primeiro lugar a Deus, que todas as manhãs oferece a todos os raios de Sol motivando ao trabalho. Agradeço pela oportunidade de viver e ser útil de alguma forma para a construção de um mundo justo e pacífico.

Com muito amor e orgulho agradeço à minha família por todo o apoio, incentivo, proteção e carinho que sempre me dão. Saudades para os meus queridos avós que já partiram. E promessas que irei honrar o conhecimento que me foi ensinado sendo um homem de bem. Agradeço com louvor todos os dias que me orientaram

Agradeço ao meu orientador, Sérgio Murilo. Pelas manhãs dedicadas a me auxiliar. Pela paciência e compreensão quando eu chegava sem qualquer norte. E pelo incentivo e orientação que sempre me concedeu.

E sem esquecer da minha turma que em todos os momentos, de alegrias pelas vitórias conquistadas ou de desespero perante as maiores provações na graduação, sempre esteve unida.

Resumo

A logística de armazenamento e distribuição abrange questões de grande interesse computacional. O objetivo de estudo e aplicação deste trabalho está em duas classes de problemas de otimização, o empacotamento 3D que consistem em arranjar itens dentro de contêineres e a roteirização para atender múltiplos destinos. Foram desenvolvidos um algoritmo para cada problema e uma interface gráfica para cada algoritmo utilizando realidade virtual e um framework para visualização e testes de grafos. No decorrer do projeto do software, foi descoberta uma estrutura que poderá vir a comportar heurísticas de empacotamento tridimensional. A idéia do aperfeiçoamento de um algoritmo de roteamento, de menor caminho de todos os pares, facilitou decisão de empacotamento com múltiplos destinos. Os testes vieram a comprovar a complexidade dos algoritmos e a base de dados obtida foi suficiente para considerar um desempenho satisfatório.

Abstract

The logistics of storage and distribution covers issues of great computational interest. The aim of the study and application of this work is in two classes of optimization problems, the 3D packaging consisting of arranging items in containers and routing to meet multiple destinations. We developed an algorithm for each problem and a graphical interface for each algorithm using virtual reality and a framework for viewing and testing of graphs. During the software project, was discovered a structure which may include three-dimensional packing heuristics. The idea of improving a routing algorithm, shortest path of all pairs, facilitated decision packaging with multiple destinations. The tests came to show the complexity of the algorithms and database obtained was insufficient to show a satisfactory performance.

ESCOLA POLITÉCNICA DE PERNAMBUCO

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

3D – Três dimensões

NP-Difícil – Não polinomial Difícil

2.5D – Variação dos problemas de duas dimensões

PCC – Problema de carregamento de contêineres

PCB – Problema de carregamento de bins

PCMC – Problema de carregamento de múltiplos contêineres

BPP – Bin Packing Problem

1DBPP – Bin Packing Problem para uma dimensão

2DBPP - Bin Packing Problem para duas dimensões

$D = (d_{ij})$ – Matriz de custos

$\Pi = (\pi_{ij})$ – Matriz de predecessores

VRML – Virtual Reality Modeling Language

JUNG – Java Universal Network/Graph Framework

pci – Ponto de canto disponível para conter um item(caixa)

ACE – Array de Cantos Escolhidos

Capítulo 1

Introdução

Dentre as principais atividades logísticas, destacam-se as atividades de armazenagem, de manuseio de materiais e de transporte de mercadorias, como chaves para a integração de cadeias de suprimentos, desde os fornecedores até os clientes. Como decorrência do aperfeiçoamento contínuo destas atividades dentro das empresas, nos últimos tempos, têm ganhado espaço conceitos de empacotamento tridimensional bem como a roteirização auxiliado por computadores. Porém, em conceituadas empresas de logísticas, encontra-se ainda um trabalho manual e impreciso de roteirização e empacotamento.

Neste trabalho, o principal objetivo é propor e testar uma heurística para resolução do problema de empacotamento 3D com múltiplos destinos e, dado que o empacotamento requer os dados prévios da roteirização, será introduzida uma solução para encontrar a melhor rota com alcances múltiplos.

O princípio deste trabalho é desenvolver uma ferramenta de simulação e visualização para as duas áreas da logística abordadas.

No Capítulo 2 será abordado o cenário da logística, limitando o escopo para o empacotamento e roteirização, mostrando aspectos e necessidades. O Capítulo 3 descreve a primeira formalização dos problemas estudados neste trabalho, e dá suporte aos fundamentos de desenvolvimento dos Capítulos 4, 5 e 6, estratégia, desenvolvimento e implementação respectivamente, onde é descrita a lógica para a resolução dos problemas de otimização. O Capítulo 7 expõe os resultados obtidos na execução dos dois algoritmos e apresenta uma breve análise. Em Conclusão e Trabalhos Futuros, Capítulo 8, é feita uma análise do desenvolvimento do trabalho e as possíveis contribuições e melhorias de projeto.

Capítulo 2

Cenário Logístico

Tendo em vista o atual cenário mundial de transporte e armazenamento, altamente competitivo e globalizado, a pressão sobre as empresas para desenvolverem cadeias de suprimento que respondam e acompanhem rapidamente às necessidades dos clientes é muito forte. Para permanecerem com poder de competição, essas empresas devem reduzir custos estratégicos, táticos e operacionais.

Enquanto isso, elas devem continuamente melhorar a qualidade do serviço ao cliente. O sucesso na administração da cadeia de suprimentos está na integração das atividades, cooperação, coordenação e compartilhamento de informações através de toda a extensão da cadeia, dos fornecedores até os clientes. Com a integração das atividades consegue-se uma redução sistemática dos custos dos estoques de produtos e a distribuição dos mesmos com maior eficácia, com menores perdas, extravios e menor tempo de entrega e com maior confiabilidade.

2.1 Empacotamento

A organização de cargas em contêineres, caminhões tipo baú, vagões ferroviários, seja por pallets ou organizada em caixas, é uma das tarefas mais complexas em empresas que tem no transporte de carga um fator logístico importante e de alto custo. Em geral, métodos manuais utilizados para estimar quantas caixas de diferentes tamanhos podem ser carregadas em um contêiner são simplistas e não apresentam um bom desempenho para carregamentos complexos.

Mesmo com a experiência de profissionais que desenvolvem essa função a um bom tempo, o resultado é uma utilização ineficiente da capacidade dos contêineres. Em certas situações é necessário até que sejam retiradas as caixas e organizadas em diferentes contêineres, devido a estimações equivocadas de capacidade.

Com uma melhor utilização do espaço nos contêineres é possível obter redução nos custos e tempo de carregamento e descarregamento.

2.2 Roteirização

O transporte é uma das atividades mais importantes da logística, que pode absorver até dois terços do seu custo, além de criar possibilidades para agregar valor ao produto. Em operações de comércio internacional há uma grande utilização do contêiner, pelas facilidades de manuseio e formas de acondicionamento das cargas, que acarretam maior segurança, confiabilidade da manutenção do estado da carga e rapidez no transporte de mercadorias.

Capítulo 3

Problemas Abordados

Saindo do cenário logístico e fixando as atenções no cenário computacional, temos que esses dois problemas são clássicos e muito estudados pelos pesquisadores. Tanto o aproveitamento de espaço, quanto encontrar a melhor rota, são descritos em inúmeros trabalhos de autores de várias nacionalidades. Cada trabalho visa testar uma técnica moderna ou comparar técnicas já vigentes.

Tendo em vista que a maioria dos estudos foi feito na década de 90, e considerando um forte crescimento do poder de processamento dos computadores, muitas das técnicas, antes muito complexas para serem implementadas, vêm sendo cada vez mais estudadas.

Um outro caso é a utilização de heurísticas baseadas em algoritmos genéticos e redes neurais para a simulação e resolução destes.

3.1 Aproveitamento de espaço

A definição dos problemas de empacotamento é aplicada a problemas em que uma ou mais unidades de maior volume, que podem representar um determinado material ou espaço, devem ser divididas em unidades pequenas. Em geral, o objetivo considerado refere-se à minimização do desperdício, ou seja, a quantidade do material ou espaço não utilizado nas unidades maiores. Os problemas de empacotamento tridimensional são uma generalização natural dos problemas clássicos de empacotamento unidimensional e bidimensional, e desta forma pertencem à classe NP-difícil. Ou seja, muito provavelmente não existem métodos ótimos que resolvam estes problemas em um tempo computacional considerável. Por este motivo, a maioria dos estudos sobre problemas de empacotamento em três dimensões concentram-se em desenvolver métodos heurísticos de resolução.

Existe um grande número de trabalhos que tratam desses problemas, mas estes estudos em sua maioria restringem-se a problemas unidimensionais e bidimensionais.

A divisão do problema de carregar caixas dentro de contêineres foi descrito por Pisinger (2002), assumindo quatro variantes: o Problema de Carregamento de Contêineres 2.5D - PCC2.5D (Strip Packing Problem), o Problema de Carregamento de Contêineres 3D - PCC3D (Knapsack Loading Problem), o Problema de Carregamento de Bins - PCB (Bin Packing Problem) e o Problema de Carregamento de Múltiplos Contêineres - PCMC (Multi Container Loading Problem).

O PCC3D foi estudado por diversos autores, por exemplo, George & Robinson (1980), Morabito & Arenales (1994, 1997), Bischoff & Ratcliff (1995) que desenvolveram uma excelente heurística para a geração de instâncias de problemas para testes, Martello et al. (2000), Bortfeldt & Gehring (2001), Eley (2002), Gehring & Bortfeldt (2002), Lins et al. (2002), Pisinger (2002), Bortfeldt et al. (2003), Li et al. (2003), Cecilio & Morabito (2004), Chien & Deng (2004), Mack et al. (2004), Lim et al. (2005), Moura & Oliveira (2005), Bischoff (2006), Araujo & Armentano (2007), Wang et al. (2008), Christensen & Rousøe (2009), e Huang & He (2009). Mesmo assim, o volume de trabalhos publicados sobre problemas tridimensionais é ainda bastante limitado comparado aos variantes 1DBPP (Bin Packing Problem para uma dimensão) e 2DBPP (Bin Packing Problem para duas dimensões), provavelmente devido à complexidade associada a estes problemas ao invés da falta de aplicações práticas.

3.2 Melhor rota

O problema consiste em encontrar caminhos mais curtos entre todos os pares de vértices em um grafo G = (V,E) ponderado, onde V sãos os vértices e E são as arestas de G, com função peso ω:E $\rightarrow$R que mapeia arestas como pesos de valores reais. Deseja-se encontrar, para todo par de vértices u,v $\in$ V, um caminho mais curto(de peso mínimo) de u até v, onde o peso de um caminho é a soma dos pesos de suas arestas constituintes.

Em geral este problema pode ser resolvido executando um algoritmo de caminhos mais curtos um número V de vezes, uma vez para cada vértice

usado como origem e, sendo todos os pesos de arestas não negativos, pode-se usar o algoritmo de Dijkstra.

Diferentes dos algoritmos de única origem, que pressupõem uma representação do grafo como uma lista de adjacências, a maioria dos algoritmos utiliza uma representação de matrizes de adjacências n X n $D=(d_{ij})$, onde a entrada d_{ij} contém o peso de um caminho mais curto desde o vértice i até o vértice j. Uma outra saída para esses tipos de algortimos é a produção de uma matriz de predecessores $\Pi = (\pi_{ij})$, utilizada para encontrar os percursos intermediários entre os nós de origem e destino.

Dentre os algoritmos de caminhos mais curtos de todos os pares, foi escolhido o algoritmo de Floyd-Warshall, pois foi o que melhor atendeu às especificações do problema de roteirização com múltiplos destinos. Este algoritmo foi modificado, tendo sua nova construção descrita na seção 5.1.2.

Na Tabela 1 é mostrada uma analogia do uso de grafos para casos reais. O desenvolvimento de algoritmos de roteamento e a aplicação nessas áreas é de grande utilidade seja para encontrar o melhor percurso entre vias, ou a melhor rota entre roteadores, ou ainda os melhores caminhos para fluxo de dados na arquitetura de um processador.

Tabela 1. Analogia para o uso de grafos

Grafo	Vértices	Arestas
Comunicação	Roteadores, Switches	Cabo de fibra ótica
Circuitos	Registradores, Processadores	Barramento
Transporte	Intersecções viárias, Aeroportos	Avenidas, rotas aéreas
Internet	páginas web	Hiperlinks
Redes Neurais	Neurônios	Sinapse
Malha elétrica	Subestações	Cabos

3.2.1 Extensão (Rotas com destinos múltiplos)

O problema abordado neste trabalho refere-se a encontrar a melhor rota de qualquer ponto que tenha que alcançar múltiplos destinos. As questões de qual o melhor arranjo de nós e qual o sub-percurso escolhido são resolvidas com uma extensão de um algoritmo de all-pair-shortest-path(caminhos mais curtos de todos os pares).

Capítulo 4

Estratégia Proposta

O problema de empacotamento para entregas com múltiplos destinos acarreta em ter um resultado da roteirização antes mesmo do processamento do empacotamento. Pois, sabendo dos destinos e as definições de suas rotas, o algoritmo de empacotamento irá atuar levando em consideração a sequência de caixas que devem ser organizadas de modo a atender a ordem de desembarque.

Em vista disso, foram desenvolvidos dois algoritmos: ROTAK – o roteirizador e o ORGANIZAK – o empacotador. Para unir os resultados, foi criada uma interface que recebe os dados do roteirizador, repassa a ordem de empacotamento e executa o ORGANIZAK.

Quanto à questão da vizualização da informação, foi escolhido trabalhar com applets em Java, VRML(para o empacotador) e a biblioteca para grafos JUNG(para o roteirizador).

Capítulo 5

Elaboração

Neste capítulo são descritos as fundamentações para a construção dos algoritmos deste trabalho.

5.1 Algoritmo de roteamento

Nesta seção é descrita uma abordagem de um algoritmo de roteamento que considera todas as possibilidades de alcançabilidade e a sequência de nós intermediários para traçar uma rota entre dois nós. A base para a criação deste algoritmo foi o trabalho proposto por Floyd-Warshall que sofreu aprimoramentos e que leva em consideração, agora, múltiplos destinos.

5.1.1 Algoritmo de Floyd-Warshall

Este algoritmo considera os vértices intermediários de um caminho mais curto, onde um vértice intermediário de um caminho simples p = {v1,v2,...,vn} é qualquer vértice de p diferente de v1 ou vn, isto é, qualquer vértice no conjunto {v2,v3,..., v(n-1)}.

O algoritmo de Floyd-Warshall se baseia na observação de que, sejam V = {1,2,...,n} os vértices de um grafo G, e o subconjunto {1,2,...,k} de vértices para algum k, para qualquer par de vértices i,j $\in$ V, todos os caminhos desde i até j cujos vértices intermediários são todos traçados a partir de {1,2,...,k} existe um p que é caracterizado como o caminho de peso mínimo dentre eles.

Se k não é um vértice intermediário do caminho p, então todos os vértices intermediários do caminho p estão no conjunto {1,2,...,k-1}. Desse modo, um caminho mais curto desde o vértice i até o vértice j com todos os vértices inermediários no conjunto {1,2,...k-1} também é um caminho mais curto desde i até j com todos os vértices intermediários no conjunto {1,2,...,k}.

Considera-se também que, se k é um vértice intermediário do caminho p, então pode-se desmembrar p em p1 e p2 tendo sub-rotas entre os nós i e j, i $\rightarrow$k$\rightarrow$j.

A definição formulada por Floyd-Warshall expõe que, para um dado $d_{ij}(k)$ o peso de um caminho mais curto desde o vértice i até o vértice j para o qual todos os vértices intermediários estão no conjunto {1,2,...,k}, se k=0 então um caminho desde o vértice i até o vértice j sem vértices intermediários com numeração mais alta que 0 não tem absolutamente nenhum vértice intermediário. Tal caminho tem no máximo uma aresta, e então $d_{ij}(0) = w_{ij}$, onde w é o peso associado ao percurso de i até j.

$$d_{ij}(k) = \begin{cases} w_{ij} & \text{se } k=0, \ \min(d_{ij}(k-1), \\ d_{ik}(k-1) - d_{kj}(k-1) & \text{se } k \geq 1 \end{cases}$$

$$\Pi ij(k) = \begin{cases} \Pi ij(k-1) & \text{se } dij(k-1) \leq dik(k-1)+dkj(k-1), \\ \Pi kj(k-1) & \text{se } dij > dik(k-1)+dkj(k-1) \end{cases}$$

O resultado da execução do algoritmo de Floyd-Warshall são pares de matrizes por iteração. São matrizes de predecessores e matrizes de custos. O número de iterações é limitado pelo tamanho da matriz de custos inicial(onde o alcance direto é dado por um número, o alcance indireto é infinito e o próprio alcance é 0).

A Figura 1 mostra a topologia utilizada para os testes do algoritmo de roteirização baseado em Floyd-Warshall e no [9] estão os resultados para as matrizes encontradas.

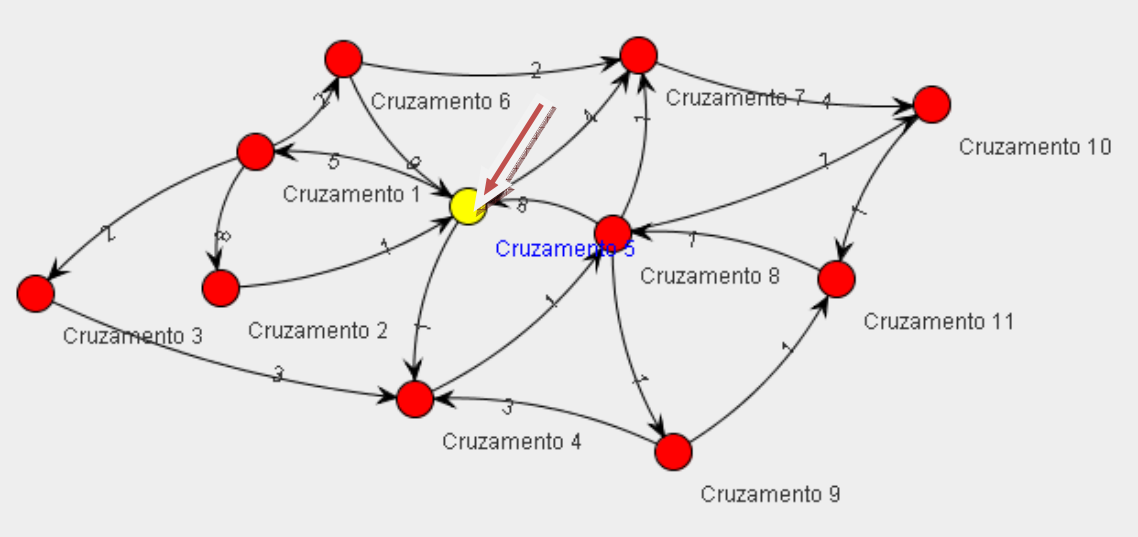


Figura 1. Topologia utilizada para os testes do roteirizador Rotak. Em destaque está o nó inicial indicando o cruzamento 5.

5.1.2 Aprimoramento de Floyd-Warshall

Com base no algoritmo descrito acima, foi desenvolvido um aperfeiçoamento para atender o objetivo da roteirização e empacotamento com múltiplos destinos. Como é considerado um problema do distribuidor, onde existem lotes de caixas com tamanhos diferentes para serem entregues em mais de um destino, a adaptação do algoritmo de Floyd-Warshall consistiu em tratar as últimas matrizes de predecessores e custos para capturar os nós intermediários, juntamente com cada custo associado.

O princípio fundamental está em, tendo destinos alcançáveis D = {d1,d2,d3,...dn} e o conjunto que se quer atingir, por exemplo A = {d1,d2,d5,d80}, a primeira medida é gerar uma permutação para os elementos do conjunto A de modo a se obter todas as possíveis sequências de rota. Depois as sequências são testadas uma por vez, para se conseguir o custo associado para cada uma delas. Sabendo-se qual ou quais as melhores sequências, pode-se traçar a rota par-a-par entre para a sequência escolhida.

Por exemplo, se a sequência A' = {d1,d2,d80,d5} possuir o menor custo associado, então teremos os pares (d1-d2), (d2-d80), (d80-d5) e, com base nas matrizes de custos e predecessores obtidas na última iteração, deve-se traçar cada sub-sequência de alcançabilidade. Ou seja, de d1 para d2 temos, por exemplo, d3-d7-d4, e então obtêm-se d1-d3-d7-d4-d2 como sequência. Assim é feito para o restante de pares até ter o caminho completo.

A Tabela 4, mais adiante na seção de resultados do ROTAK, mostra os arranjos gerados para o conjunto de testes do algoritmo de roteamento. Na Figura 2 são apresenta um fluxograma com as principais etapas do ROTAK. A dificuldade no desenvolvimento deste algoritmo foi a separação dos sub-percursos para cada arranjo escolhido como aquele de menor custo.

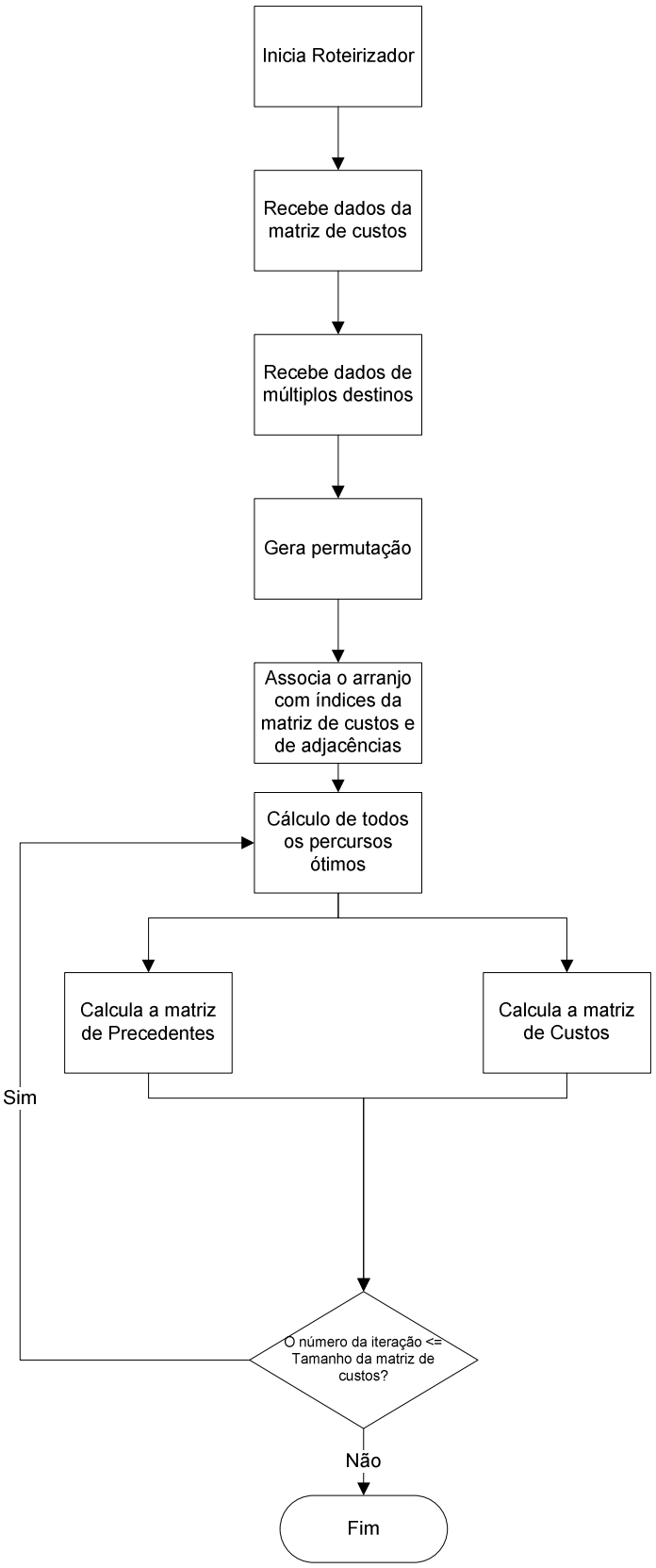


Figura 2. Fluxograma do algoritmo ROTAK

5.2 Algoritmo de empacotamento

A seguir, é mostrada uma definição para o problema de empacotamento em três dimensões. A heurística utilizada neste trabalho foi a escolha aleatória das caixas e pontos de canto.

Seja um objeto (contêiner, caminhão, vagão ferroviário) de largura L, comprimento C e altura H conhecidos, que deve ser carregado com quaisquer tipos de caixas, e cada tipo de caixa deve possuir comprimento c, largura l e altura h. Admitindo-se as dimensões das caixas sendo do tipo inteiro, que elas só podem ser empacotadas ortogonalmente e sendo especificado o ponto de canto para cada caixa, além de ser adotado um sistema de coordenadas cartesianas com origem no canto inferior frontal esquerdo do contêiner, foi desenvolvido um algoritmo de empacotamento que utiliza os pontos de canto associados às caixas que os criaram, sendo disponibilizados para comportar mais caixas, contanto que obedeçam o critério de restrição que envolve não ultrapassar os limites do contêiner.

Para a encontrar a solução de um problema NP-Difícil pode-se usar um método ótimo ou um método heurístico, onde a heurística é a ciência da criação ou do descobrimento. No decorrer do desenvolvimento do algoritmo, foi notado que houve uma construção de uma possível “caixa de heurísticas”, pois, o modo com que foi organizado o código possibilita que, caso uma heurística desenvolvida possa ser adaptada com a abordagem de pontos de canto, ela estará apta a ser testada no Empacotador e exibida no conjunto Gerador e Visualizador em VRML.

5.2.1 Pontos de canto

Um ponto de canto é todo aquele ponto criado após a inserção da caixa em outro ponto(disponível para conter uma caixa) onde o critério de ortogonalidade é satisfeito, ou seja, que este ponto forme um ângulo de 90° com a intersecção de 3 linhas. A Figura 3 mostra, em destaque, os pontos de canto criados após a inserção de três caixas em outros três pontos de canto.

Sejam três coordenadas inteiras e não negativas (x , y, z). Supõe-se que a origem do sistema de coordenadas seja o canto inferior esquerdo do fundo do bin e considere-se que $pc_i(x_p, y_p, z_p)$ sejam as coordenadas de um ponto de

canto pc_i. Tem-se a regra: um item(caixa) j não pode ser colocado em pc_i caso $x_p+w_j > W$ ou $y_p+h_j > H$, ou ainda, que $z_p+d_j > D$. Se p é um ponto de canto então nenhum item de ACE, conjunto dos itens já empacotados, poderá ter alguma interseção à direita de pci, acima de pci, ou defronte a pci.

À medida que uma caixa de dimensões(l,h,c) é empacotada a regra para a criação de pontos de canto considera que, a partir do ponto escolhido, se o lado da caixa a ser inserida for maior que aquela que produziu o ponto de canto, então os valores dos novos pontos serão:

$$(x_p+l,y_p,z_p),(x_p,y_p+h,z_p)(x_p,y_p,z_p+c)$$

Caso contrário, as coordenadas serão:

$$(l,y_p,z_p),(x_p,h,z_p),(x_p,y_p,c)$$

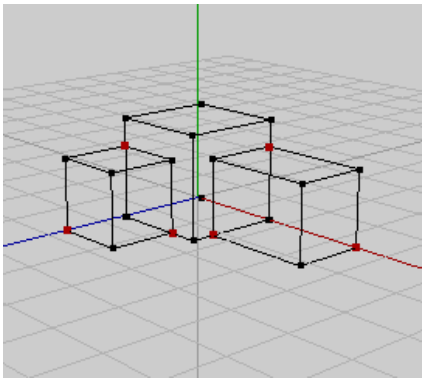


Figura 3. Pontos de canto criados pela inserção de caixas

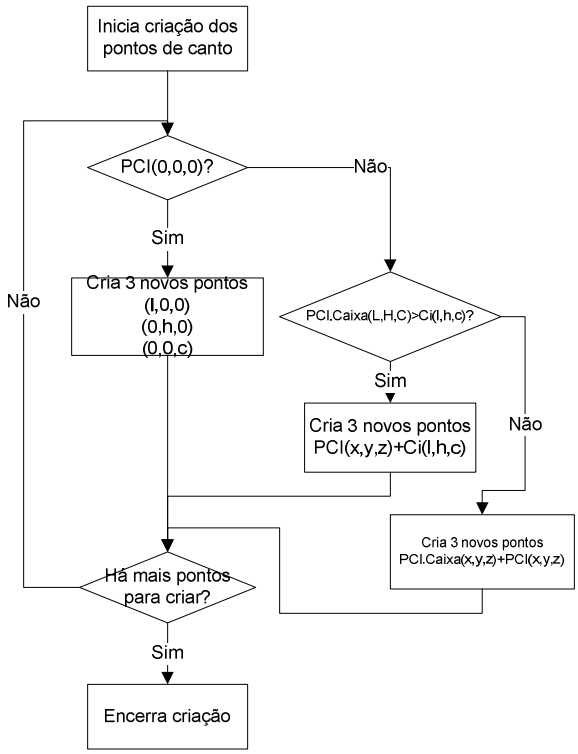


Figura 4. Gerador de Pontos de Canto

O algoritmo completo de empacotamento está descrito no Apêndice C.

5.2.2 Restrições

As restrições adotadas para esse algoritmo são simples. Apenas visam que, a partir de um ponto de canto escolhido, a caixa que nele for empacotada não ultrapasse nenhum limite do contêiner e que uma caixa não pode ser rotacionada em nenhum eixo coordenado, tendo sempre suas orientações fixas.

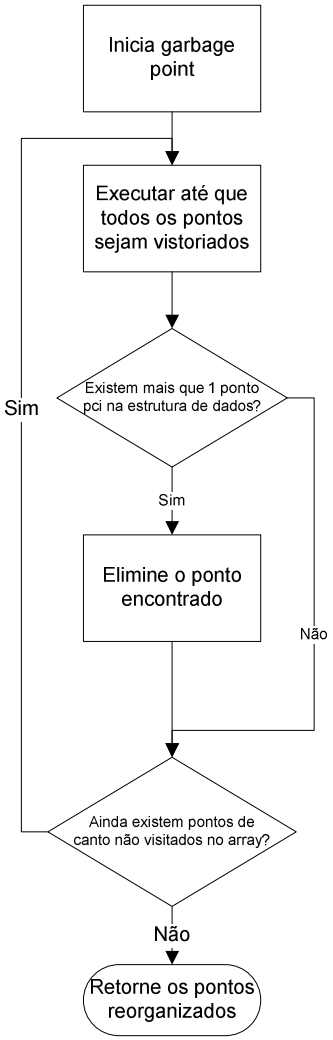
Posteriormente, uma série de medidas de restrição poderão ser aplicadas, visto que uma caixa para heurísticas(contanto que sejam adaptadas à noção de pontos de canto) foi criada. Essas restrições podem ser do tipo, fragilidade da carga, estabilidade do contêiner, estabilidade da carga, limite de empilhamento.

5.2.3 Garbage Point

Durante o desenvolvimento do algoritmo, uma série de objetos do tipo Ponto de Canto eram criados e, muitas vezes repetidos, ficando à disposição

para o empacotamento e causando incoerência, pois poderia já ter sido utilizado.

Em vista disso foi criado um coletor de pontos de canto, seguindo o critério mostrado nos fluxogramas da Figura 5. Na estrutura de dados que contém os pontos, caso haja duplicações ou caso o limite das caixas empacotadas ocupem mais de um ponto de canto, estes serão removidos do conjunto de pontos disponíveis. O coletor de pontos é chamado toda vez que há a criação de novos pontos de canto, ou seja, a cada inserção(empacotamento) de caixas. No Apêndice B está detalhado na íntegra o código do garbage point.



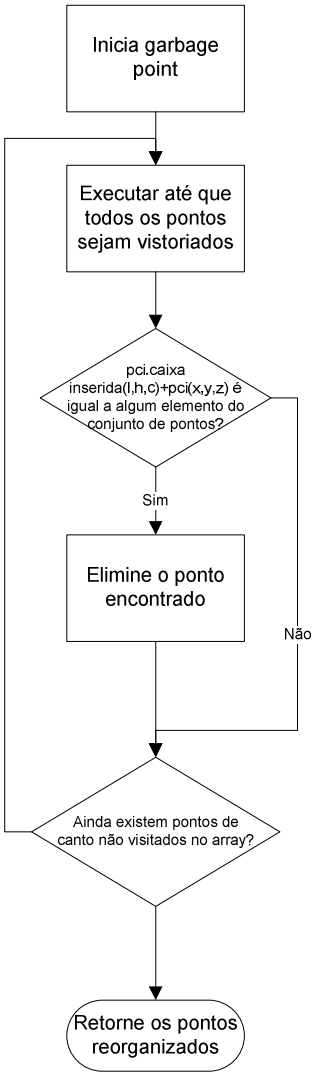


Figura 5. Duas instâncias para o Garbage Point Collector,tratamento de pontos iguais e pontos ocupados no momento da inserção.

5.3 Junção dos resultados

Na fase de ligação entre as duas etapas, uma interface atua após o último ciclo do roteirizador e após ele ter preenchido a tabela de roteamento e a ordem das entregas. Cada entrega tem um conjunto de caixas associada e a solução proposta foi subdividir o contêiner, de forma a comportar o empacotamento separado, assim isolando a carga referente a cada destino.

A implementação é simples e esse tipo de abordagem não é o modo otimizado, onde, por exemplo poderia ser considerado cada ponto de canto remanescente do empacotamento anterior e verificar se é possível aproveitar para uma nova organização.

Capítulo 6

Implementação

As linguagens de programação envolvidas no desenvolvimento deste trabalho foram Java e VRML. O motivo para a escolha das duas é o mesmo, o custo computacional para a interpretação, a portabilização e a possibilidade de expansão da realidade virtual para a visualização da informação via web.

6.1 Hardware/Software

Para o desenvolvimento e testes dos algoritmos foi utilizado um computador com a seguinte configuração:

Tabela 2. Configurações de Hardware e Software utilizados para desenvolvimento e testes

Sistema Operacional	**Microsoft Windows XP Professional**
OS Service Pack	Service Pack 3
DirectX	4.09.00.0904 (DirectX 9.0c)
CPU	DualCore , 2400 MHz (12 x 200)
Motherboard	Asus P5KPL
Motherboard Chipset	Intel Bearlake G31/G33
Memória RAM	2038 MB
Vídeo	Intel(R) G33/G31 Express Chipset Family (256 MB)
3D Accelerator	Intel Bearlake

6.2 UML

Aqui são apresentadas as principais classes envolvidas no desenvolvimento do projeto. A Figura 6 representa o diagrama para o roteirizador onde foi utilizada uma interface para grafos em Java chamada JUNG e a Figura 7 mostra o diagrama do empacotador, onde foi adotada a visualização por realidade virtual, tornando o processamento gráfico bem mais suave.

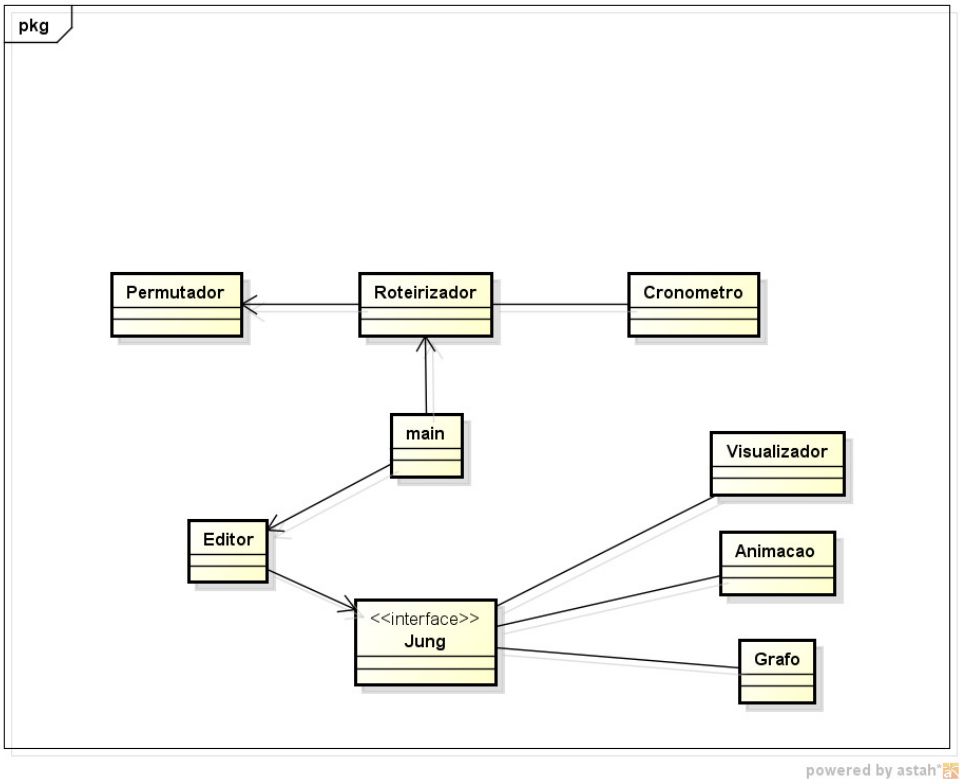


Figura 6. Diagrama de classes do ROTAK

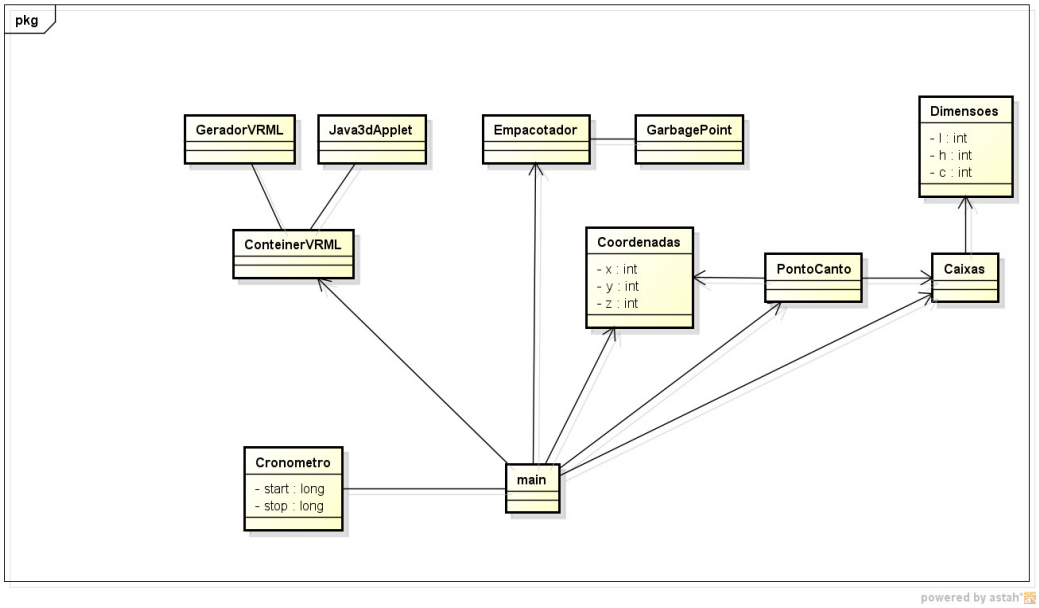


Figura 7. Diagrama de classes do ORGANIZAK

6.3 Descrição do construtor e interpretador VRML

Durante o desenvolvimento da interface gráfica do ORGANIZAK, houve a necessidade da criação de um ambiente para a visualização da informação, gerada pelo algoritmo de empacotamento. Em primeira implementação uma interface em Java3D, cujo código fonte principal está disponível no Apêndice G, era utilizada, porém apresentava um péssimo desempenho para rotação, translação e zoom. A solução foi apostar na utilização de um applet e unir com VRML, utilizando a biblioteca para Java vrml97.jar.

Os resultados foram excelentes. Com a geração de um arquivo de extensão .wrl e a criação de um visualizador, o processamento gráfico do ambiente 3D melhorou de forma que as propriedades de rotação, translaçãao e zoom ficaram 10 vezes mais rápidas.

A Figura 8 mostra o resultado do empacotamento de caixas de tamanhos iguais no interpretador VRML.

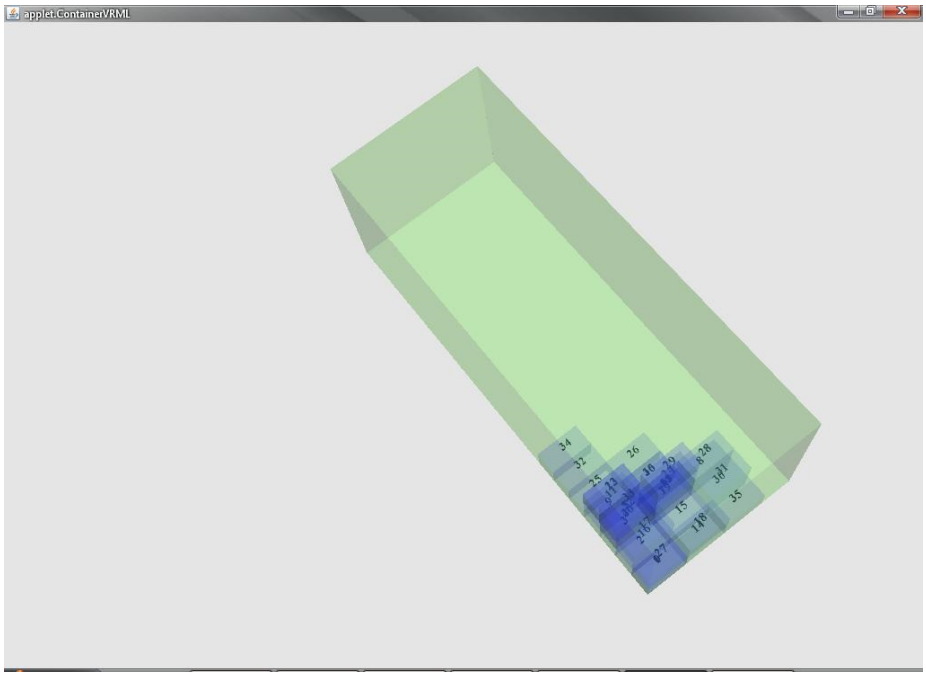


Figura 8. Contêiner gerado com realidade virtual. As caixas são numeradas para a melhor compreensão no momento do empacotamento.

6.4 Biblioteca gráfica para grafos JUNG

JUNG - Java Universal Network / Graph Framework - é uma biblioteca de software que fornece uma linguagem comum e extensível para a modelagem, análise e visualização de dados que pode ser representado como um grafo ou uma rede. Ela é escrita em Java, que permite que aplicativos baseados em JUNG façam uso da muitos recursos internos da API Java.

A arquitetura JUNG é projetada para suportar uma variedade de representações de entidades e suas relações, tais como grafos dirigidos e não dirigidos, grafo multi-modal, grafos com bordas paralelas e hipergrafos. Este framework fornece um mecanismo para anotar grafos, entidades e relações como metadados. Isso facilita a criação de ferramentas analíticas para conjuntos de dados complexos que podem examinar as relações entre as entidades, bem como os metadados ligados a cada entidade e relação.

JUNG inclui implementações de vários algoritmos de teoria dos grafos, mineração de dados e análise de redes sociais, incluindo agrupamento, filtragem, geração de gráficos aleatórios, blockmodeling, cálculo das distâncias

das redes e fluxos, e uma grande variedade de métricas (PageRank, HITS, intermediação , proximidade, etc.)

Ele também oferece um quadro de visualização que facilita a construção de ferramentas para a exploração interativa de dados da rede. Os usuários podem usar um dos algoritmos de layout fornecido, ou usar a estrutura para criar seus próprios layouts personalizados.

Como uma biblioteca de código aberto, Jung estabelece um quadro comum para o grafo de análise de rede e visualização. O Jung torna mais fácil o trabalho com dados de grafo e de rede.

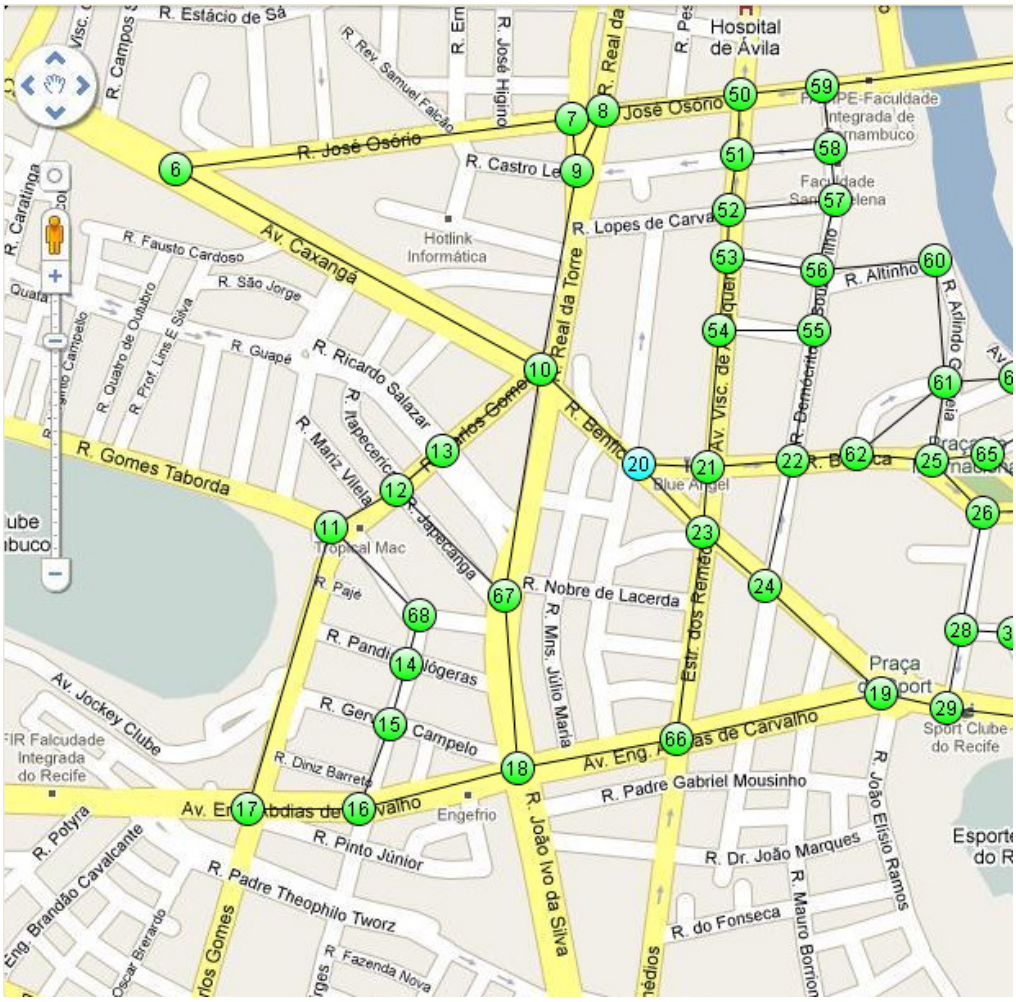


Figura 9. Exemplo de visualização de grafo viário.

Em vista disso, essa biblioteca foi escolhida para que a simulação do algoritmo ROTAK pudesse ser feita.

Capítulo 7

Resultados

7 ROTAK e ORGANIZAK

Nesta seção são apresentados os resultados obtidos na execução dos algoritmos de roteamento e empacotamento, ROTAK e ORGANIZAK.

7.1 ROTAK

Tendo a topologia citada anteriormente na seção 5.1.1 foi criada uma classe em Java, chamada Cronometro, com métodos de captura do tempo atual em milissegundos e nanossegundos. Foi utilizado apenas a captura em nanossegundos e posteriormente uma conversão em milissegundos pois, o resultado em tempo apresentava uma precisão com poucas casas decimais.

O comportamento do ROTAK para os 10 arranjos de 3 vértices alcançáveis durante 10 iterações foi, em média semelhante, ou seja, a complexidade não esteve no valor dos pesos e sim na quantidade de vértices. O mesmo se aplica no teste com 5 vértices alcançáveis para os 3 arranjos nas 10 iterações. Salvo para a segunda iteração do segundo arranjo onde, provavelmente algum lixo na memória ou um outro processo, requisitando prioridade naquele momento, pode ter aumentado consideravelmente o tempo de execução.

Tabela 3. Resultados de tempo em milissegundos para 10 iterações e 3 nós de destino

		1	2	3	4	5	6	7	8	9	10
Iteração	Quantidade de nós de destino/ Arranjo	5213	5134	5487	471011	10549	10467	2387	2549	9483	1597
	3										
1		2.0	3.0	3.0	3.0	3.0	3.0	2.0	3.0	3.0	3.0
2		1.0	3.0	2.0	2.0	2.0	4.0	2.0	2.0	3.0	2.0
3		4.0	5.0	3.0	4.0	5.0	1.0	3.0	1.0	1.0	2.0
4		4.0	7.0	4.0	4.0	2.0	2.0	4.0	2.0	3.0	4.0
5		1.0	2.0	3.0	3.0	4.0	3.0	1.0	3.0	3.0	3.0
6		1.0	2.0	1.0	5.0	3.0	3.0	1.0	3.0	1.0	1.0
7		4.0	7.0	4.0	2.0	4.0	4.0	3.0	1.0	1.0	1.0
8		4.0	3.0	3.0	2.0	1.0	1.0	3.0	3.0	3.0	3.0
9		1.0	5.0	3.0	4.0	4.0	1.0	1.0	3.0	3.0	3.0
10		1.0	1.0	3.0	3.0	4.0	4.0	2.0	1.0	1.0	1.0

Tabela 4. Arranjos de destinos sub-divididos para o conjunto de testes de 10 iterações e 3 nós de destino

Destinos	Arranjo Simples	Arranjo com sub-percurso
5-3-2-1	5213	5-1-2-5-1-3
5-3-4-1	5134	5-1-3-4
5-8-7-4	5487	5-4-8-7
4-11-7-10	471011	4-8-7-10-11
10-5-4-9	10549	10-11-8-5-4-8-9
10-7-4-6	10467	10-11-8-9-4-8-5-1-6-7
2-8-7-3	2387	2-5-1-3-4-8-7
2-5-4-9	2549	2-5-4-8-9
9-8-4-3	9483	9-4-8-5-1-3
1-5-9-7	1597	1-2-5-4-8-9-11-8-7

Tabela 5. Resultados de tempo em milissegundos para 10 iterações e 5 nós de destino

Iteração	Quantidade de nós de destino/Arranjo	Arranjo 1	Arranjo 2	Arranjo 3
		1-10-3-2-4-8-7	1-11-4-7-2-3-9	91187715
	5			
1		447.0	501.0	432.0
2		455.0	1129.0	505.0
3		504.0	442.0	463.0
4		455.0	493.0	485.0
5		469.0	443.0	436.0
6		481.0	459.0	440.0
7		449.0	479.0	487.0
8		495.0	489.0	436.0
9		494.0	517.0	448.0
10		529.0	437.0	475.0

Tabela 6. Arranjos de destinos sub-divididos para o conjunto de testes de 10 iterações e 5 nós de destino

Destinos	Arranjo Simples	Arranjo com sub-percurso
1-10-4-7-2-3-8	1-10-3-2-4-8-7	16710118513485125487
1-3-4-11-9-2-7	1-11-4-7-2-3-9	1-6-7-10-11-8-5-1-3-4-8-5-1-2-5-4-8-9-11
9-11-5-8-1-6-7	91187715	91187101185125

7.2 ORGANIZAK

A complexidade do ORGANIZAK tem como o pior caso O(n) que é impulsionado pelo número limite de itens disponíveis no conjunto de caixas. Como mostra o código do Apêndice C. E quanto mais caixas geradas, três vezes mais pontos de canto serão gerados. Como a heurística para a escolha da caixa e do ponto para a inserção são aleatórios, outro fator que pode prejudicar o desempenho é a quantidade de pontos de canto a serem removidos no garbage point(código no 0), que apresenta complexidade de O(2.n). Como o empacotador executa sempre uma chamada aos métodos de coleta do garbage point, a cada iteração, tem-se uma complexidade de $O(n^2)$.

Os resultados apresentados na Tabela 7 e Tabela 8 comprovam a complexidade do ORGANIZAK. O tempo computacional tende a aumentar em função de n^2 com o aumento do número de caixas.

Tabela 7. Resultados experimentais de tempo de execução, em milissegundos, do algoritmo ORGANIZAK com conjuntos de caixas iguais

Nº caixas/Iterações	30	50	55	70	80	90	100	120	150	180	200
1	265	438	438	547	609	687	766	907	1156	1375	1500
2	250	422	422	547	610	672	750	906	1125	1359	1500
3	250	437	421	531	609	688	750	906	1125	1360	1516
4	250	406	438	531	594	671	750	906	1125	1359	1531
5	250	407	437	547	609	672	765	907	1141	1375	1516
6	313	406	438	547	594	672	750	906	1125	1406	1531
7	250	406	422	531	609	688	766	906	1140	1422	1515
8	250	406	437	532	594	672	766	906	1141	1391	1516
9	250	422	422	531	609	687	750	907	1141	1375	1516
10	250	406	438	531	610	688	765	922	1140	1359	1531
11	250	422	437	547	609	687	750	906	1141	1391	1516
12	234	453	438	563	610	688	766	922	1156	1359	1515
13	250	422	422	546	609	687	750	906	1156	1360	1500
14	250	422	437	547	609	672	766	906	1125	1359	1516
15	250	422	438	547	610	687	765	922	1141	1359	1500
16	250	422	437	547	609	672	766	906	1156	1360	1515

17	250	422	438	547	609	687	765	907	1125	1359	1516
18	250	422	437	562	610	688	766	921	1125	1359	1516
19	266	422	438	547	609	687	750	891	1125	1375	1515
20	250	421	437	532	610	688	766	906	1125	1360	1500
21	250	407	438	546	609	687	750	907	1125	1359	1516
22	250	422	421	563	609	688	750	890	1125	1375	1500
23	250	437	438	562	610	672	750	906	1125	1360	1500
24	250	406	422	563	609	687	750	907	1125	1375	1516
25	250	422	437	547	609	672	750	906	1141	1375	1531
26	250	422	438	547	594	688	750	906	1140	1359	1531
27	250	422	437	562	610	687	766	906	1125	1359	1516
28	250	406	438	547	609	672	766	907	1125	1391	1531
29	250	422	437	531	609	688	828	906	1125	1375	1500
30	250	438	422	547	610	671	750	906	1125	1359	1516

Tabela 8. Resultados experimentais do tempo, em milissegundos, do algoritmo ORGANIZAK com conjuntos de caixas diferentes geradas aleatoriamente

Nº caixas/Iterações	30	50	55	70	80	90	100	120	150	180	200
1	1531	1578	1579	1593	1594	1593	1594	1641	1562	1641	1625
2	1516	1579	1578	1594	1500	1610	1594	1672	1578	1578	1640
3	1500	1546	1500	1609	1562	1640	1516	1656	1672	1609	1688
4	1593	1610	1609	1594	1594	1610	1515	1656	1625	1594	1672
5	1594	1547	1563	1531	1516	1531	1735	1657	1578	1672	1703
6	1594	1515	1609	1594	1547	1609	1953	1671	1672	1594	1687
7	1578	1610	1656	1578	1593	1594	1578	1688	1657	1578	1891
8	1594	1593	1594	1594	1610	1547	1687	1672	1687	1672	1906
9	1594	1610	1531	1578	1593	1531	1657	1594	1688	1625	1906
10	1593	1594	1578	1531	1594	1610	1656	1656	1640	1578	1813
11	1578	1593	1578	1531	1594	1593	1594	1594	1641	1593	1750
12	1594	1610	1594	1516	1609	1532	1687	1625	1640	1657	1828
13	1563	1593	1594	1562	1594	1531	1547	1656	1672	1578	1750
14	1578	1579	1672	1579	1578	1578	1781	1672	1594	1578	1859
15	1562	1609	1594	1515	1578	1594	1750	1593	1672	1672	1829
16	1532	1594	1578	1578	1532	1531	1656	1657	1641	1562	1843
17	1578	1562	1593	1578	1593	1594	1688	1672	1563	1688	1844
18	1593	1594	1594	1579	1578	1593	1734	1640	1672	1703	1813

19	1579	1625	1594	1578	1594	1610	1641	1594	1687	1641	1828
20	1578	1578	1594	1578	1500	1609	1781	1641	1625	1609	1843
21	1578	1578	1593	1578	1594	1531	1672	1656	1609	1641	1750
22	1609	1594	1532	1500	1578	1516	1625	1594	1672	1656	1782
23	1594	1594	1593	1562	1578	1641	1609	1656	1703	1672	1828
24	1562	1578	1625	1579	1594	1578	1657	1656	1657	1578	1844
25	1610	1594	1516	1500	1594	1500	1672	1641	1672	1672	1750
26	1640	1578	1578	1578	1578	1594	1671	1656	1656	1656	1735
27	1579	1578	1563	1578	1578	1515	1688	1578	1640	1609	1843
28	1562	1594	1562	1594	1547	1516	1672	1672	1672	1657	1813
29	1563	1578	1594	1562	1594	1562	1734	1672	1672	1656	1844
30	1609	1578	1562	1578	1593	1578	1688	1656	1656	1656	1796

Capítulo 8

Conclusão e Trabalhos Futuros

Durante os estudos e o desenvolvimento deste trabalho, foram abordadas duas áreas logísticas, o empacotamento tridimensional para a otimização do espaço utilizado e a roteirização. Foram simulados e analisados os casos para uma heurística simples, a escolha aleatória dos itens(caixas) e foram contempladas duas inovações:

- A idéia de um coletor de pontos de canto;
- A expansão do algoritmo de Floyd-Warshall;
- A criação de uma caixa de heurísticas;
- A criação de um gerador e interpretador de VRML para Java.

Ainda existem muitos temas a serem estudados e anexados a este trabalho para que venha a fazer testes de comparação entre os algoritmos e heurísticas mais utilizadas. E também para inspecionar novas técnicas.

Como melhoria e possíveis contribuições deste projeto estão

- Estudo de caso com empresa de logística em atividade para aplicar e desenvolver as técnicas e empacotamento e roteirização;
- Testes do algoritmo de roteamento com roteadores;
- Integração do algoritmo de roteamento no banco de dados geográficos PostGIS;
- Disponibilização das ferramentas para ensino de grafos e da caixa de heurísticas para testes de qualquer outro método de empacotamento que atenda aos requisitos.

Bibliografia

[1] BAIRD, H.S. Document Image Defect Models. Structured Document Image Analysis, vol. 2, n. 3, p. 546-556, 1992.

[2] BALTACIOGLU, E. THE DISTRIBUTER' S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A HUMAN INTELLIGENCE-BASED HEURISTIC APPROACH. 2001.135 f. Dissertação (Mestrado) - Air Force Institute of Technology. Ohio, EUA, 2001.

[3] JUNQUEIRA, L. & MORABITO, R., YAMASHITA, D. Modelos de otimização para problemas de carregamento de contêineres com considerações de estabilidade e de empilhamento. 2010. 18 f. Departamento de Engenharia de Produção / Universidade Federal de São Carlos (UFSCar). São Carlos, SP, 2010.

[4] MARINESCU, D. & PAUL, I. & BAICOIANU, A. A Topological Order for a Rectangular Three Dimensional Bin Packing Problem. 2008. 6 f. WSEAS. Transilvania University of Brasov. Heraklion, Greece, 2008

[5] MELLO, C.A.B. Synthesis of Images of Historical Documents for Web Visualization, Proceedings of IEEE International Multi-Media Modelling Conference, 2004, Brisbane, AU.

[6] SANTOS, A.M.R.P. Paletização Rectangular: Caracterização e Métodos de Resolução. 1995. 86 f. Dissertação (Mestrado) – Faculdade de Engenharia da Universidade do Porto. Porto, Portugal, 1995.

[7] SOMA, N. & SILVA, J. UM ALGORITMO POLINOMIAL PARA O PROBLEMA DE EMPACOTAMENTO DE CONTÊINERES COM ESTABILIDADE ESTÁTICA DA CARGA. 2002. 20 p. Instituto Tecnológico de Aeronáutica. São José dos Campos – SP, 2002.

[8] API JUNG Disponível em: http://jung.sourceforge.net/doc/api/index.html, Acesso em 20 de out. 2010.

[9] MANUAL JUNG. Disponível em : http://sourceforge.net/apps/trac/jung/wiki/JUNGManual Acessado em 20 de out. 2010.

[10] Cormen, Thomas H. Algoritmos – Rio de Janeiro: Campus Editora, Teoria e Prática. 2002.

[11] Carey, R. & Bell G. The annotated VRML 2.0 reference manual. ISBN 0-201-41974-2. 2002.

[12] VRML Guide. Disponível em: http://tecfa.unige.ch/guides/vrml/vrml97/spec/. Acessado em: 01 de Nov. 2010.

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

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

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

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

[17] J.L.C. Silva, N.Y. Soma e N. Maculan. A heuristic and a simulated annealing approach for the three-dimensional bin packing problem. Aceito para apresentação na IV Meta-heuristic International Conference, Porto, Portugal, julho, 2001

[18] R. Garey e D.S. Johnson, Computers and Intractability: A guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

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

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

[21] Martelo, S.; Monaci, M. e Vigo, D. (2003). An exact approach to strip-packing problem. INFORMS Journal on Computing, v. 15, p. 310–319.

Apêndice A

Matrizes de adjacências e predecessores do Rotak

D(0) = [0, 8, 2, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL]
[NIL, 0, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, NIL, NIL, 1, 0, NIL, 4, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 9, 0, 2, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(0) = [NIL, 1, 1, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, NIL, NIL, 5, NIL, NIL, 5, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 6, NIL, 6, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(1) = [0, 8, 2, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL]
[NIL, 0, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 9, 0, 2, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(1) = [NIL, 1, 1, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 6, NIL, 6, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(2) = [0, 8, 2, NIL, 9, 2, NIL, NIL, NIL, NIL, NIL]
[NIL, 0, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 9, 0, 2, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(2) = [NIL, 1, 1, NIL, 2, 1, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 6, NIL, 6, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(3) = [0, 8, 2, 5, 9, 2, NIL, NIL, NIL, NIL, NIL]
[NIL, 0, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 9, 0, 2, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(3) = [NIL, 1, 1, 3, 2, 1, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 6, NIL, 6, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(4) = [0, 8, 2, 5, 9, 2, NIL, 6, NIL, NIL, NIL]
[NIL, 0, NIL, NIL, 1, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, 2, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 9, 0, 2, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(4) = [NIL, 1, 1, 3, 2, 1, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 2, NIL, NIL, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, 6, NIL, 6, NIL, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, 8, NIL, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, 4, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(5) = [0, 8, 2, 5, 9, 2, 13, 6, NIL, NIL, NIL]
[6, 0, 8, 2, 1, 8, 5, 3, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, 2, NIL, NIL, NIL]
[14, 22, 16, 10, 9, 0, 2, 11, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[13, 21, 15, 9, 8, 15, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(5) = [NIL, 1, 1, 3, 2, 1, 5, 4, NIL, NIL, NIL]
[5, NIL, 1, 5, 2, 1, 5, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, 4, NIL, NIL, NIL]
[5, 1, 1, 5, 6, NIL, 6, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[5, 1, 1, 5, 8, 1, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, 4, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(6) = [0, 8, 2, 5, 9, 2, 4, 6, NIL, NIL, NIL]
[6, 0, 8, 2, 1, 8, 5, 3, NIL, NIL, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, 2, NIL, NIL, NIL]
[14, 22, 16, 10, 9, 0, 2, 11, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[13, 21, 15, 9, 8, 15, 1, 0, 1, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(6) = [NIL, 1, 1, 3, 2, 1, 6, 4, NIL, NIL, NIL]
[5, NIL, 1, 5, 2, 1, 5, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, 4, NIL, NIL, NIL]
[5, 1, 1, 5, 6, NIL, 6, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[5, 1, 1, 5, 8, 1, 8, NIL, 8, 8, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, 4, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(7) = [0, 8, 2, 5, 9, 2, 4, 6, NIL, 8, NIL]
[6, 0, 8, 2, 1, 8, 5, 3, NIL, 9, NIL]
[NIL, NIL, 0, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, 0, NIL, NIL, NIL, 1, NIL, NIL, NIL]
[5, 13, 7, 1, 0, 7, 4, 2, NIL, 8, NIL]
[14, 22, 16, 10, 9, 0, 2, 11, NIL, 6, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[13, 21, 15, 9, 8, 15, 1, 0, 1, 5, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, 0, NIL, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 1, NIL, NIL, 0]

P(7) = [NIL, 1, 1, 3, 2, 1, 6, 4, NIL, 7, NIL]
[5, NIL, 1, 5, 2, 1, 5, 4, NIL, 7, NIL]
[NIL, NIL, NIL, 3, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 4, NIL, NIL, NIL]
[5, 1, 1, 5, NIL, 1, 5, 4, NIL, 7, NIL]
[5, 1, 1, 5, 6, NIL, 6, 4, NIL, 7, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[5, 1, 1, 5, 8, 1, 8, NIL, 8, 7, NIL]
[NIL, NIL, NIL, 9, NIL, NIL, NIL, 4, NIL, NIL, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, 11, NIL, NIL, NIL]

D(8) = [0, 8, 2, 5, 9, 2, 4, 6, 7, 8, NIL]
[6, 0, 8, 2, 1, 8, 4, 3, 4, 8, NIL]
[17, 25, 0, 3, 12, 19, 5, 4, 5, 9, NIL]
[14, 22, 16, 0, 9, 16, 2, 1, 2, 6, NIL]
[5, 13, 7, 1, 0, 7, 3, 2, 3, 7, NIL]
[14, 22, 16, 10, 9, 0, 2, 11, 12, 6, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[13, 21, 15, 9, 8, 15, 1, 0, 1, 5, NIL]
[17, 25, 19, 3, 12, 19, 5, 4, 0, 9, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[14, 22, 16, 10, 9, 16, 2, 1, 2, 6, 0]

P(8) = [NIL, 1, 1, 3, 2, 1, 6, 4, 8, 7, NIL]
[5, NIL, 1, 5, 2, 1, 8, 4, 8, 7, NIL]
[5, 1, NIL, 3, 8, 1, 8, 4, 8, 7, NIL]
[5, 1, 1, NIL, 8, 1, 8, 4, 8, 7, NIL]
[5, 1, 1, 5, NIL, 1, 8, 4, 8, 7, NIL]
[5, 1, 1, 5, 6, NIL, 6, 4, 8, 7, NIL]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[5, 1, 1, 5, 8, 1, 8, NIL, 8, 7, NIL]
[5, 1, 1, 9, 8, 1, 8, 4, NIL, 7, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[5, 1, 1, 5, 8, 1, 8, 11, 8, 7, NIL]

D(9) = [0, 8, 2, 5, 9, 2, 4, 6, 7, 8, 8]
[6, 0, 8, 2, 1, 8, 4, 3, 4, 8, 5]
[17, 25, 0, 3, 12, 19, 5, 4, 5, 9, 6]
[14, 22, 16, 0, 9, 16, 2, 1, 2, 6, 3]
[5, 13, 7, 1, 0, 7, 3, 2, 3, 7, 4]
[14, 22, 16, 10, 9, 0, 2, 11, 12, 6, 13]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, NIL]
[13, 21, 15, 4, 8, 15, 1, 0, 1, 5, 2]
[17, 25, 19, 3, 12, 19, 5, 4, 0, 9, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[14, 22, 16, 5, 9, 16, 2, 1, 2, 6, 0]

P(9) = [NIL, 1, 1, 3, 2, 1, 6, 4, 8, 7, 9]
[5, NIL, 1, 5, 2, 1, 8, 4, 8, 7, 9]
[5, 1, NIL, 3, 8, 1, 8, 4, 8, 7, 9]
[5, 1, 1, NIL, 8, 1, 8, 4, 8, 7, 9]
[5, 1, 1, 5, NIL, 1, 8, 4, 8, 7, 9]
[5, 1, 1, 5, 6, NIL, 6, 4, 8, 7, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, NIL]
[5, 1, 1, 9, 8, 1, 8, NIL, 8, 7, 9]
[5, 1, 1, 9, 8, 1, 8, 4, NIL, 7, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[5, 1, 1, 9, 8, 1, 8, 11, 8, 7, NIL]

D(10) = [0, 8, 2, 5, 9, 2, 4, 6, 7, 8, 8]
[6, 0, 8, 2, 1, 8, 4, 3, 4, 8, 5]
[17, 25, 0, 3, 12, 19, 5, 4, 5, 9, 6]
[14, 22, 16, 0, 9, 16, 2, 1, 2, 6, 3]
[5, 13, 7, 1, 0, 7, 3, 2, 3, 7, 4]
[14, 22, 16, 10, 9, 0, 2, 11, 12, 6, 7]
[NIL, NIL, NIL, NIL, NIL, NIL, 0, NIL, NIL, 4, 5]
[13, 21, 15, 4, 8, 15, 1, 0, 1, 5, 2]
[17, 25, 19, 3, 12, 19, 5, 4, 0, 9, 1]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 0, 1]
[14, 22, 16, 5, 9, 16, 2, 1, 2, 6, 0]

P(10) = [NIL, 1, 1, 3, 2, 1, 6, 4, 8, 7, 9]
[5, NIL, 1, 5, 2, 1, 8, 4, 8, 7, 9]
[5, 1, NIL, 3, 8, 1, 8, 4, 8, 7, 9]
[5, 1, 1, NIL, 8, 1, 8, 4, 8, 7, 9]
[5, 1, 1, 5, NIL, 1, 8, 4, 8, 7, 9]
[5, 1, 1, 5, 6, NIL, 6, 4, 8, 7, 10]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 7, 10]
[5, 1, 1, 9, 8, 1, 8, NIL, 8, 7, 9]
[5, 1, 1, 9, 8, 1, 8, 4, NIL, 7, 9]
[NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, NIL, 10]
[5, 1, 1, 9, 8, 1, 8, 11, 8, 7, NIL]

D(11) = [0, 8, 2, 5, 9, 2, 4, 6, 7, 8, 8]

[6, 0, 8, 2, 1, 8, 4, 3, 4, 8, 5]

[17, 25, 0, 3, 12, 19, 5, 4, 5, 9, 6]

[14, 22, 16, 0, 9, 16, 2, 1, 2, 6, 3]

[5, 13, 7, 1, 0, 7, 3, 2, 3, 7, 4]

[14, 22, 16, 10, 9, 0, 2, 8, 9, 6, 7]

[19, 27, 21, 10, 14, 21, 0, 6, 7, 4, 5]

[13, 21, 15, 4, 8, 15, 1, 0, 1, 5, 2]

[15, 23, 17, 3, 10, 17, 3, 2, 0, 7, 1]

[15, 23, 17, 6, 10, 17, 3, 2, 3, 0, 1]

[14, 22, 16, 5, 9, 16, 2, 1, 2, 6, 0]

P(11) = [NIL, 1, 1, 3, 2, 1, 6, 4, 8, 7, 9]

[5, NIL, 1, 5, 2, 1, 8, 4, 8, 7, 9]

[5, 1, NIL, 3, 8, 1, 8, 4, 8, 7, 9]

[5, 1, 1, NIL, 8, 1, 8, 4, 8, 7, 9]

[5, 1, 1, 5, NIL, 1, 8, 4, 8, 7, 9]

[5, 1, 1, 5, 6, NIL, 6, 11, 8, 7, 10]

[5, 1, 1, 9, 8, 1, NIL, 11, 8, 7, 10]

[5, 1, 1, 9, 8, 1, 8, NIL, 8, 7, 9]

[5, 1, 1, 9, 8, 1, 8, 11, NIL, 7, 9]

[5, 1, 1, 9, 8, 1, 8, 11, 8, NIL, 10]

[5, 1, 1, 9, 8, 1, 8, 11, 8, 7, NIL]

Apêndice B

GarbagePoint.java

```
import java.util.ArrayList;

public class GarbagePoint {

    public GarbagePoint() {
    }

    public ArrayList<PontoCanto> coleta(ArrayList<PontoCanto> pontos,
ArrayList<PontoCanto> APE) {

        for (int j = 0; j < pontos.size(); j++) {
            for (int i = 0; i < APE.size(); i++) {
                if ((pontos.get(j).getPC().getX() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() +
pontos.get(j).getCaixa().getDimensoes().getComp() == APE.get(i).getPC().getZ())) {
                    APE.remove(i);
                    //System.out.println("Removeu ponto de canto. Condição 1");              }
                if ((pontos.get(j).getPC().getX() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() +
pontos.get(j).getCaixa().getDimensoes().getAlt() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 2");
                    APE.remove(i);              }
                if ((pontos.get(j).getPC().getX() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() +
pontos.get(j).getCaixa().getDimensoes().getAlt() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() +
pontos.get(j).getCaixa().getDimensoes().getComp() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 3");
                    APE.remove(i);              }
                if ((pontos.get(j).getPC().getX() +
pontos.get(j).getCaixa().getDimensoes().getLarg() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 4");
                    APE.remove(i);              }
                if ((pontos.get(j).getPC().getX() +
pontos.get(j).getCaixa().getDimensoes().getLarg() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() +
pontos.get(j).getCaixa().getDimensoes().getComp() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 5");
                    APE.remove(i);              }
                if ((pontos.get(j).getPC().getX() +
pontos.get(j).getCaixa().getDimensoes().getLarg() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() +
pontos.get(j).getCaixa().getDimensoes().getAlt() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 6");
                    APE.remove(i);              }
                if ((pontos.get(j).getPC().getX() +
```

```
pontos.get(j).getCaixa().getDimensoes().getLarg() == APE.get(i).getPC().getX())
                        && (pontos.get(j).getPC().getY() +
pontos.get(j).getCaixa().getDimensoes().getAlt() == APE.get(i).getPC().getY())
                        && (pontos.get(j).getPC().getZ() +
pontos.get(j).getCaixa().getDimensoes().getComp() == APE.get(i).getPC().getZ())) {
                    //System.out.println("Removeu ponto de canto. Condição 7");
                    APE.remove(i);              }          }       }
            for (int i = 0; i < APE.size(); i++) {
                PontoCanto pci, pcj;
                pci = APE.get(i);
                for (int j = i; j < APE.size(); j++) {
                    pcj = APE.get(j);
                    if (i != j) {
                        if (pci.getPC().getX() == pcj.getPC().getX()
                                && pci.getPC().getY() == pcj.getPC().getY()
                                && pci.getPC().getZ() == pcj.getPC().getZ()) {
                            APE.remove(pcj);
                            //System.out.println("Removeu ponto de canto idêntico");    }   }   } }
            return APE;   }   }
```

Apêndice C

Empacotador.java

```
import java.util.ArrayList;
import java.util.Random;

public class Empacotador {

    private int st = 1; //estado
    private ArrayList<PontoCanto> APCD; //Conjunto de pontos de canto disponíveis
    private ArrayList<PontoCanto> APCC; //Conjunto de pontos já utilizados e em sequência
    private ArrayList<Caixa> ACE; //Conjunto de caixas empacotadas
    private ArrayList<Caixa> ACD; //Conjuto de caixas disponíveis
    private Caixa ci;

    public Empacotador() {
    }

    /**
     * Retira uma caixa aleatoriamente de ACD
     * acrescenta em ACE e retorna a caixa escolhida
     */
    public Caixa escolheCaixa() {

        Caixa caixa = null;
        Random random = new Random();


        int randomIndice = random.nextInt(ACD.size());
        caixa = ACD.get(randomIndice);

        ACE.add(caixa);
        ACD.remove(randomIndice);
        //System.out.println("Caixa " + caixa.getID() + " escolhida");

        return caixa;

    }

    /**
     * Retira um ponto de cando do conjunto APC
     * acrescenta em APES e retorna o ponto escolhido
     */
    public PontoCanto escolhePonto() {

        PontoCanto pc = null;


        //Heurística Aleatória
        Random random = new Random();
        int randomIndice = random.nextInt(APCD.size());

        //System.out.println("Numero aleatório escolhido: " + randomIndice);

        pc = APCD.get(randomIndice);
```

```
        APCC.add(pc);
        APCD.remove(randomIndice);

        /*
         * Heurística À esquerda
         * Empacotar no ponto de canto mais à esquerda e no nível mais baixo
         */

        return pc;
    }

    public void carregaArrayCaixas(int limite) {

        ACD = new ArrayList<Caixa>();

        for (int i = 0; i < limite; i++) {
            Caixa caixa = new Caixa();
            caixa.setID(i);
            Dimensoes dimensoes = new Dimensoes(8, 5, 4);
            caixa.setDimensoes(dimensoes);
            ACD.add(caixa);
        }
    }

    public void carregaArrayCaixasRandom() {

        ACD = new ArrayList<Caixa>();
        Random random = new Random();
        int l, h, c;
        for (int i = 0; i < 200; i++) {
            Caixa caixa = new Caixa();
            caixa.setID(i);
            l = random.nextInt(10) + 1;
            h = random.nextInt(10) + 1;
            c = random.nextInt(10) + 1;
            Dimensoes dimensoes = new Dimensoes(l, h, c);
            caixa.setDimensoes(dimensoes);
            ACD.add(caixa);
        }
    }

    /**
     *
     */
    public void empacotar(Caixa conteiner, PontoCanto pci, int numeroCaixas) {

        carregaArrayCaixas(numeroCaixas);
        //carregaArrayCaixasRandom();

        APCD = new ArrayList<PontoCanto>();
        APCC = new ArrayList<PontoCanto>();

        ACE = new ArrayList<Caixa>();

        APCC.add(pci);

        APCD.add(pci);

        int L, H, C;

        L = conteiner.getDimensoes().getLarg();
        H = conteiner.getDimensoes().getAlt();
```

```
                C = conteiner.getDimensoes().getComp();

                /*
                 * Condição de parada:
                 * Quando não for mais possível empacotar mais nenhuma caixa, ou seja,
                 * quando qualquer caixa a ser empacotada ultrapassar o limite do conteiner
                 */

                //System.out.println("Tamanho do array de caixas antes do Loop: "+ACD.size());
                for (int i = 0; ACD.size()>0 ; i++) {
                    //System.out.println("Tamanho do array de caixas é: "+ACD.size()+" na iteração
nº: "+ i);

                    if (st == 1) {
                        ci = escolheCaixa();
                        pci = escolhePonto();
                        //System.out.println("Ponto com coordenadas: " + pci.getPC().getX() + "," +
pci.getPC().getY() + "," + pci.getPC().getZ() + " ESCOLHIDO. Condição st = 1");
                    } else if (st == 0) {
                        ci = escolheCaixa();
                        //System.out.println("Ponto com coordenadas: " + pci.getPC().getX() + "," +
pci.getPC().getY() + "," + pci.getPC().getZ() + " ESCOLHIDO. Condição st = 0");
                    }

                    /*
                     * Não ultrapassar o limite do bin
                     * sendo pci o ponto de canto atual e ci a caixa a ser inserida
                     */
                    Coordenadas coordenadap = new Coordenadas(pci.getPC().getX(),
pci.getPC().getY(), pci.getPC().getZ());
                    Coordenadas coordenadacx = new Coordenadas(L, 0, 0);
                    Coordenadas coordenadacy = new Coordenadas(0, H, 0);
                    Coordenadas coordenadacz = new Coordenadas(0, 0, C);

                    //System.out.println("Tamanho do array de pontos criados: " + APCC.size());
                    if ((distancia(coordenadap, coordenadacx) > ci.getDimensoes().getLarg())
                            && (distancia(coordenadap, coordenadacy) > ci.getDimensoes().getAlt())
                            && (distancia(coordenadap, coordenadacz) >
ci.getDimensoes().getComp())) {
                        pci.setCaixa(ci);
                        geraPC(pci, ci);
                        st = 1;
                    } else {

                        st = 0;
                    }

                     Runtime r = Runtime.getRuntime();
                     r.gc();
                }


                //System.out.println("Quantidade de pontos utilizados: " + APCC.size());
                //System.out.println("Quantidade de caixas empacotadas: " + ACE.size());

            }

            public ArrayList<PontoCanto> getAPCC() {
                return APCC;
            }

            public ArrayList<Caixa> getACE() {
```

```
        return ACE;
    }

    /**
     *
     * A partir do ponto escolhido, se o lado da caixa que
     * produziu o ponto de canto for maior que aquela a ser
     * inserida então o valor do novo ponto será:
     * (pci.x+ci.l,pci.y,pci.z),
     * (pci.x,pci.y+ci.h,pci.z),
     * (pci.x,pci.y,pci.z+ci.c)
     * senão
     * (pci.x+pci.caixa.l,pci.y,pci.z),
     * (pci.x,pci.y+pci.caixa.h,pci.z),
     * (pci.x,pci.y,pci.z+pci.caixa.c)
     *
     */
    public void geraPC(PontoCanto pci, Caixa caixa) {
        //escolha dos pontos de canto
        //se pci = <0,0,0>

        Coordenadas coordenada;
        ArrayList<PontoCanto> pontos = new ArrayList<PontoCanto>();

        if ((pci.getPC().getX() == 0) && (pci.getPC().getY() == 0) && (pci.getPC().getZ()
== 0)) {
            //cria novos pontos de canto

            coordenada = new Coordenadas(caixa.getDimensoes().getLarg(), 0, 0);
            PontoCanto pci1 = new PontoCanto(coordenada);
            pci1.setCaixa(caixa);
            //System.out.println("Ponto com coordenadas: " + pci1.getPC().getX() + "," +
pci1.getPC().getY() + "," + pci1.getPC().getZ() + " GERADO");
            APCD.add(pci1);
            pontos.add(pci1);
            coordenada = new Coordenadas(0, caixa.getDimensoes().getAlt(), 0);
            PontoCanto pci2 = new PontoCanto(coordenada);
            pci2.setCaixa(caixa);
            //System.out.println("Ponto com coordenadas: " + pci2.getPC().getX() + "," +
pci2.getPC().getY() + "," + pci2.getPC().getZ() + " GERADO");
            APCD.add(pci2);
            pontos.add(pci2);
            coordenada = new Coordenadas(0, 0, caixa.getDimensoes().getComp());
            PontoCanto pci3 = new PontoCanto(coordenada);
            pci3.setCaixa(caixa);
            //System.out.println("Ponto com coordenadas: " + pci3.getPC().getX() + "," +
pci3.getPC().getY() + "," + pci3.getPC().getZ() + " GERADO");
            APCD.add(pci3);
            pontos.add(pci3);
        } else {

            if (pci.getCaixa().getDimensoes().getLarg() < caixa.getDimensoes().getLarg()) {
                coordenada = new Coordenadas(pci.getPC().getX() +
pci.getCaixa().getDimensoes().getLarg(), pci.getPC().getY(), pci.getPC().getZ());
                PontoCanto pci1 = new PontoCanto(coordenada);
                pci1.setCaixa(caixa);
                //System.out.println("Ponto com coordenadas: " + pci1.getPC().getX() + "," +
pci1.getPC().getY() + "," + pci1.getPC().getZ() + " GERADO");
                APCD.add(pci1);
                pontos.add(pci1);
            } else {
                coordenada = new Coordenadas(pci.getPC().getX() +
```

```
caixa.getDimensoes().getLarg(), pci.getPC().getY(), pci.getPC().getZ());
                    PontoCanto pci1 = new PontoCanto(coordenada);
                    pci1.setCaixa(caixa);
                    //System.out.println("Ponto com coordenadas: " + pci1.getPC().getX() + "," +
pci1.getPC().getY() + "," + pci1.getPC().getZ() + " GERADO");
                    APCD.add(pci1);
                    pontos.add(pci1);
                }

                if (pci.getCaixa().getDimensoes().getComp() <
caixa.getDimensoes().getComp()) {
                    coordenada = new Coordenadas(pci.getPC().getX(), pci.getPC().getY(),
pci.getPC().getZ() + pci.getCaixa().getDimensoes().getComp());
                    PontoCanto pci2 = new PontoCanto(coordenada);
                    pci2.setCaixa(caixa);
                    //System.out.println("Ponto com coordenadas: " + pci2.getPC().getX() + "," +
pci2.getPC().getY() + "," + pci2.getPC().getZ() + " GERADO");
                    APCD.add(pci2);
                    pontos.add(pci2);
                } else {
                    coordenada = new Coordenadas(pci.getPC().getX(), pci.getPC().getY(),
pci.getPC().getZ() + caixa.getDimensoes().getComp());
                    PontoCanto pci2 = new PontoCanto(coordenada);
                    pci2.setCaixa(caixa);
                    //System.out.println("Ponto com coordenadas: " + pci2.getPC().getX() + "," +
pci2.getPC().getY() + "," + pci2.getPC().getZ() + " GERADO");
                    APCD.add(pci2);
                    pontos.add(pci2);
                }

                if (pci.getCaixa().getDimensoes().getAlt() < caixa.getDimensoes().getAlt()) {
                    coordenada = new Coordenadas(pci.getPC().getX(), pci.getPC().getY() +
pci.getCaixa().getDimensoes().getAlt(), pci.getPC().getZ());
                    PontoCanto pci3 = new PontoCanto(coordenada);
                    pci3.setCaixa(caixa);
                    //System.out.println("Ponto com coordenadas: " + pci3.getPC().getX() + "," +
pci3.getPC().getY() + "," + pci3.getPC().getZ() + " GERADO");
                    APCD.add(pci3);
                    pontos.add(pci3);
                } else {
                    coordenada = new Coordenadas(pci.getPC().getX(), pci.getPC().getY() +
caixa.getDimensoes().getAlt(), pci.getPC().getZ());
                    PontoCanto pci3 = new PontoCanto(coordenada);
                    pci3.setCaixa(caixa);
                    //System.out.println("Ponto com coordenadas: " + pci3.getPC().getX() + "," +
pci3.getPC().getY() + "," + pci3.getPC().getZ() + " GERADO");
                    APCD.add(pci3);
                    pontos.add(pci3);
                }

            }

            //System.out.println("Tamanho do array de pontos disponíveis antes do garbage: "
+ APCD.size());

            GarbagePoint garbagePoint = new GarbagePoint();
            APCD = garbagePoint.coleta(pontos, APCD);

            //System.out.println("Tamanho do array de pontos disponíveis depois do garbage:
" + APCD.size());

        }
```

```
        public Double distancia(Coordenadas p1, Coordenadas p2) {
            Double d = Math.sqrt(Math.pow(p2.getY() - p1.getY(), 2) + Math.pow(p2.getX() -
p1.getX(), 2) + Math.pow(p2.getZ() - p1.getZ(), 2));

            return d;
        }
    }
```

Apêndice D

ContainerVRML.java

```
import java.awt.BorderLayout;
import java.awt.Cursor;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Hashtable;

import javax.media.j3d.AmbientLight;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Bounds;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Group;
import javax.media.j3d.Morph;
import javax.media.j3d.Node;
import javax.media.j3d.SceneGraphObject;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3d;

import com.sun.j3d.loaders.Scene;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.picking.PickCanvas;
import com.sun.j3d.utils.picking.PickResult;
import com.sun.j3d.utils.picking.PickTool;
import org.jdesktop.j3d.loaders.vrml97.VrmlLoader;

public class ContainerVRML extends Java3dApplet implements MouseListener {
  PickCanvas pickCanvas = null;

  public ContainerVRML() {
  }

  public ContainerVRML(String[] args) {
    saveCommandLineArguments(args);
    initJava3d();
  }

  public void start() {
    if (pickCanvas == null)
      initJava3d();
  }

  protected void addCanvas3D(Canvas3D c3d) {
```

```
    setLayout(new BorderLayout());
    add(c3d, BorderLayout.CENTER);
    doLayout();

    if (m_SceneBranchGroup != null) {
      c3d.addMouseListener(this);

      pickCanvas = new PickCanvas(c3d, m_SceneBranchGroup);
      pickCanvas.setMode(PickTool.GEOMETRY_INTERSECT_INFO);
      pickCanvas.setTolerance(4.0f);
    }

    c3d.setCursor(new Cursor(Cursor.HAND_CURSOR));
  }

  public TransformGroup[] getViewTransformGroupArray() {
    TransformGroup[] tgArray = new TransformGroup[1];
    tgArray[0] = new TransformGroup();

    Transform3D viewTrans = new Transform3D();
    Transform3D eyeTrans = new Transform3D();

    BoundingSphere sceneBounds = (BoundingSphere) m_SceneBranchGroup
        .getBounds();

    Point3d center = new Point3d();
    sceneBounds.getCenter(center);
    double radius = sceneBounds.getRadius();
    Vector3d temp = new Vector3d(center);
    viewTrans.set(temp);

    double eyeDist = 1.0 * radius / Math.tan(Math.toRadians(40) / 2.0);
    temp.x = 0.0;
    temp.y = 0.0;
    temp.z = eyeDist;
    eyeTrans.set(temp);
    viewTrans.mul(eyeTrans);

    tgArray[0].setTransform(viewTrans);

    return tgArray;
  }

  protected BranchGroup createSceneBranchGroup() {
    BranchGroup objRoot = super.createSceneBranchGroup();

    Bounds lightBounds = getApplicationBounds();

    AmbientLight ambLight = new AmbientLight(true, new Color3f(1.0f, 1.0f,
        1.0f));
    ambLight.setInfluencingBounds(lightBounds);
    objRoot.addChild(ambLight);

    DirectionalLight headLight = new DirectionalLight();
    headLight.setInfluencingBounds(lightBounds);
    objRoot.addChild(headLight);

    TransformGroup mouseGroup = createMouseBehaviorsGroup();

    String vrmlFile = null;

    try {
```

```
            URL codebase = getWorkingDirectory();
            vrmlFile = codebase.toExternalForm() + "container.wrl";
          } catch (Exception e) {
            e.printStackTrace();
          }

          if (m_szCommandLineArray != null) {
            switch (m_szCommandLineArray.length) {
            case 0:
              break;

            case 1:
              vrmlFile = m_szCommandLineArray[0];
              break;

            default:
              System.err.println("Executando: ContainerVRML. Erro ao ler arquivo");
              System.exit(-1);
            }
          }

          BranchGroup sceneRoot = loadVrmlFile(vrmlFile);

          if (sceneRoot != null)
            mouseGroup.addChild(sceneRoot);

          objRoot.addChild(mouseGroup);

          return objRoot;
        }

        private TransformGroup createMouseBehaviorsGroup() {
          TransformGroup examineGroup = new TransformGroup();
          examineGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
          examineGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

          Bounds behaviorBounds = getApplicationBounds();

          MouseRotate mr = new MouseRotate(examineGroup);
          mr.setSchedulingBounds(behaviorBounds);
          examineGroup.addChild(mr);

          MouseTranslate mt = new MouseTranslate(examineGroup);
          mt.setSchedulingBounds(behaviorBounds);
          examineGroup.addChild(mt);

          MouseZoom mz = new MouseZoom(examineGroup);
          mz.setSchedulingBounds(behaviorBounds);
          examineGroup.addChild(mz);

          return examineGroup;
        }

        private BranchGroup loadVrmlFile(String location) {
          BranchGroup sceneGroup = null;
          Scene scene = null;

          VrmlLoader loader = new VrmlLoader();

          try {
            URL loadUrl = new URL(location);
            try {
```

```
        scene = loader.load(new URL(location));
      } catch (Exception e) {
        System.out.println("Exception :" + e);
        e.printStackTrace();
      }
    } catch (MalformedURLException badUrl) {

      try {

        scene = loader.load(location);
      } catch (Exception e) {
        System.out.println("Exception :" + e);
        e.printStackTrace();
      }
    }

    if (scene != null) {

      sceneGroup = scene.getSceneGroup();

      sceneGroup.setCapability(BranchGroup.ALLOW_BOUNDS_READ);
      sceneGroup.setCapability(BranchGroup.ALLOW_CHILDREN_READ);

      Hashtable namedObjects = scene.getNamedObjects();
      System.out.println("*** nome do objeto VRML: \n"
          + namedObjects);


      java.util.Enumeration enumValues = namedObjects.elements();
      java.util.Enumeration enumKeys = namedObjects.keys();

      if (enumValues != null) {
        while (enumValues.hasMoreElements() != false) {
          Object value = enumValues.nextElement();
          Object key = enumKeys.nextElement();

          recursiveSetUserData(value, key);
        }
      }
    }

    return sceneGroup;
  }

  void recursiveSetUserData(Object value, Object key) {
    if (value instanceof SceneGraphObject != false) {

      SceneGraphObject sg = (SceneGraphObject) value;
      sg.setUserData(key);

      if (sg instanceof Group) {
        Group g = (Group) sg;

        java.util.Enumeration enumKids = g.getAllChildren();

        while (enumKids.hasMoreElements() != false)
          recursiveSetUserData(enumKids.nextElement(), key);
      } else if (sg instanceof Shape3D || sg instanceof Morph) {
        PickTool.setCapabilities((Node) sg, PickTool.INTERSECT_FULL);
      }
    }
```

```
    }

    public void mouseClicked(MouseEvent e) {
      System.out.println("*** Clique ***");

      pickCanvas.setShapeLocation(e);
      PickResult[] results = pickCanvas.pickAllSorted();

      if (results != null) {
        for (int n = 0; n < results.length; n++) {
          PickResult pickResult = results[n];

          System.out
              .println("Clicou em " + n + ": " + pickResult);

          Node actualNode = pickResult.getObject();

          if (actualNode.getUserData() != null) {
            System.out.println("Sorted Object " + n + ": "
                + actualNode.getUserData());
          }
        }
      }

      PickResult pickResult = pickCanvas.pickClosest();

      if (pickResult != null) {
        System.out.println("Result: " + pickResult);

        Node actualNode = pickResult.getObject();

        if (actualNode.getUserData() != null) {
          System.out.println("Objeto selecionado: "
              + actualNode.getUserData());
        }
      }

    }

    public static void main(String[] args) throws IOException {


      //SplashScreen splash = new SplashScreen(1000);
      //splash.showSplash();

        geradorVRML vRML = new geradorVRML();
        vRML.geradorConteiner();

      ContainerVRML pickingTest = new ContainerVRML(args);

      new MainFrame(pickingTest, 600, 400);
    }
  }
```

Apêndice E

geradorVRML.java

```
import empacotador.Caixa;
import empacotador.Coordenadas;
import empacotador.Dimensoes;
import empacotador.Empacotador;
import empacotador.PontoCanto;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

public class geradorVRML {

    private String arquivo = "container.wrl";
    ArrayList<Coordenadas> coord = new ArrayList<Coordenadas>();
    ArrayList<Dimensoes> dim = new ArrayList<Dimensoes>();

    public void geradorConteiner() {
        String L, H, C;
        Empacotador empacotador = new Empacotador();
        Caixa conteiner = new Caixa();
        Dimensoes dimensoes = new Dimensoes(30, 80, 30);
        conteiner.setDimensoes(dimensoes);
        PontoCanto pontoCanto = new PontoCanto();
        Coordenadas coordenadas = new Coordenadas(0, 0, 0);
        pontoCanto.setPC(coordenadas);
        empacotador.empacotar(conteiner, pontoCanto,100);
        ArrayList<PontoCanto> APCC = new ArrayList<PontoCanto>();
        ArrayList<Caixa> ACE = new ArrayList<Caixa>();
        APCC = empacotador.getAPCC();
        ACE = empacotador.getACE();
        for (int i = 0; i < APCC.size()-1; i++) {
            coord.add(APCC.get(i).getPC());
            dim.add(ACE.get(i).getDimensoes());       }

        try {
            FileWriter f = new FileWriter(arquivo);

            PrintWriter pr = new PrintWriter(f);

            L = Float.toString(dimensoes.getLarg() / 10.f);
            C = Float.toString(dimensoes.getComp() / 10.f);
            H = Float.toString(dimensoes.getAlt() / 10.f);



            for (int i = 1; i <= 6; i++) {

                if ((i == 1)) {
                    //alinhado
                    pr.println("#VRML V2.0 utf8");
                    pr.println("DEF W" + i + " Transform {");
```

```
                pr.println("translation 3 4 1.5");
                pr.println("rotation 0 0 0 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ C +" "+ L +" }");
                pr.println("appearance Appearance {");
                pr.println("material Material {");
                pr.println("diffuseColor 0.4 0.8 0.3");
                pr.println("transparency 0.8 }}}}");
            } else if (i == 2) {
                //alinhado
                pr.println("DEF W" + i + " Transform {");
                pr.println("translation 0 4 1.5");
                pr.println("rotation 1 0 0 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ C +" "+ L +" }");
                pr.println("appearance Appearance {");
                pr.println("material Material {");
                pr.println("diffuseColor 0.4 0.8 0.3");
                pr.println("transparency 0.8 }}}}");
            } else if (i == 3) {
                pr.println("DEF W" + i + " Transform {");
                pr.println("translation 1.5 4 3");
                pr.println("rotation 0.5 0 -0.5 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ C +" "+ L +" }");
                pr.println("appearance Appearance {");
                pr.println("material Material {");
                pr.println("diffuseColor 0.4 0.8 0.3");
                pr.println("transparency 0.8 }}}}");
            } else if (i == 4) {
                pr.println("DEF W" + i + " Transform {");
                pr.println("translation 1.5 4 0");
                pr.println("rotation 0.5 0 0.5 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ C +" "+ L +" }");
                pr.println("appearance Appearance {");
                pr.println("material Material {");
                pr.println("diffuseColor 0.4 0.8 0.3");
                pr.println("transparency 0.8 }}}}");
            } else if (i == 5) {
                pr.println("DEF W" + i + " Transform {");
                pr.println("translation 1.5 0 1.5");
                pr.println("rotation 0.5 -0.5 0 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ H +" "+ L +" }");
                pr.println("appearance Appearance {");
                pr.println("material Material {");
                pr.println("diffuseColor 0.4 0.8 0.3");
                pr.println("transparency 0.8 }}}}");
            } else if (i == 6) {
                pr.println("DEF W" + i + " Transform {");
                pr.println("translation 1.5 8 1.5");
                pr.println("rotation 0.5 0.5 0 3.14159");
                pr.println("children Shape {");
                pr.println("geometry Box {");
                pr.println("size 0.001 "+ H +" "+ L +" }");
                pr.println("appearance Appearance {");
```

```
                        pr.println("material Material {");
                        pr.println("diffuseColor 0.4 0.8 0.3");
                        pr.println("transparency 0.8 }}}}");              }          }

                    geraItem(f, pr);

                    pr.close();
                    f.close();

                } catch (IOException ex) {
                    System.err.println(ex.getMessage());
                    System.exit(-1);       }    }

            public void geraItem(FileWriter f, PrintWriter pr) {

                pr = new PrintWriter(f);

                for (int i = 0; i < coord.size(); i++) {
                    String x, y, z;
                    String l, c, h;

                    l = Float.toString(((dim.get(i).getLarg()) / 10.f));
                    c = Float.toString(((dim.get(i).getComp()) / 10.f));
                    h = Float.toString(((dim.get(i).getAlt()) / 10.f));

                    if (i == 0) {
                        x = Float.toString(((coord.get(i).getX() + dim.get(i).getLarg())) / (2.f * 10.f));
                        y = Float.toString(((coord.get(i).getY() + dim.get(i).getAlt())) / (2.f * 10.f));
                        z = Float.toString(((coord.get(i).getZ() + dim.get(i).getComp())) / (2.f * 10.f));
                    } else {
                        x = Float.toString((((coord.get(i).getX() + dim.get(i).getLarg())) / (2.f * 10.f)) +
((coord.get(i).getX())) / (2.f * 10.f));
                        y = Float.toString((((coord.get(i).getY() + dim.get(i).getAlt())) / (2.f * 10.f)) +
((coord.get(i).getY())) / (2.f * 10.f));
                        z = Float.toString((((coord.get(i).getZ() + dim.get(i).getComp())) / (2.f * 10.f))
+ ((coord.get(i).getZ())) / (2.f * 10.f));           }

                    pr.println("DEF W7 Transform {");
                    pr.println("translation " + x + " " + y + " " + z);
                    pr.println("rotation 0 0 0 3.14159");
                    pr.println("children Shape {");
                    pr.println("geometry Box {");
                    pr.println("size " + l + " " + h + " " + c +"}");
                    pr.println("appearance Appearance {");
                    pr.println("material Material {");
                    pr.println("diffuseColor 0.1 0.1 0.8");
                    pr.println("transparency 0.8 }}}}");

                    pr.println("DEF W8 Transform {");
                    pr.println("translation " + x + " " + y + " " + z);
                    pr.println("children Shape {");
                    pr.println("appearance Appearance { material Material { diffuseColor 0 0 0}}");
                    pr.println("geometry Text {");
                    pr.println("string [\"" + i + "\"]");
                    pr.println("fontStyle FontStyle { "
                            + "style  \"BOLD\" "
                            + "size .2 }");
                    pr.println("}}}");       }    }    }
```

Apêndice F

FloydWarshallAperfeicoado.java

```
import java.util.ArrayList;

public class FloydWarshallAperfeicoado {

    private static int matrixSize = 0;
    private static int Rows = 0;
    private static int Columns = 0;
    private static int[][] MatrizOriginal;
    private static int[][] MatrizCustosRotas;
    private static int[][] pred;
    private static String infinityString = "NIL";
    private static int infinity = 100;
    private static int[] percursoSimles;
    private static int[] percursoIdeal;
    private static ArrayList<Integer> percurso;
    private static int[][] pares;

    public FloydWarshallAperfeicoado() {

        doFloydWarshall(MatrizOriginal, matrixSize);
        System.out.println("Terminada a execução de Floyd-Warshall com
aperfeiçoamento.");    }

    private void doFloydWarshall(int[][] oldW, int matrixSize) {
        printMatrix(MatrizOriginal, matrixSize, 0);
        int[][] newW = new int[matrixSize][matrixSize];
        MatrizCustosRotas = new int[matrixSize][matrixSize];
        for (int i = 0; i < newW.length; i++) {
            for (int j = 0; j < newW.length; j++) {
                newW[i][j] = oldW[i][j];
                if ((newW[i][j]) == infinity || (i == j)) {
                    pred[i][j] = infinity;
                } else if ((i != j) && newW[i][j] < infinity) {
                    pred[i][j] = i + 1;    }    }    }

        printMatrixPrecedentes(pred, matrixSize, 0);
        for (int k = 0; k < matrixSize; k++) {
            for (int i = 0; i < matrixSize; i++) {
                for (int j = 0; j < matrixSize; j++) {
                    //newW[i][j] = min(i, j, k, oldW);
                    if (newW[i][j] <= newW[i][k] + newW[k][j]) {
                        pred[i][j] = pred[i][j];
                    } else {
                        pred[i][j] = pred[k][j];
                    }
                    if ((newW[i][k] + newW[k][j]) < (newW[i][j])) {
                        newW[i][j] = newW[i][k] + newW[k][j];
                    }
                }
            }
            oldW = newW;
            printMatrix(newW, matrixSize, k + 1);
            printMatrixPrecedentes(pred, matrixSize, k + 1);
```

```
        }

        for (int i = 0; i < newW.length; i++) {
            for (int j = 0; j < newW.length; j++) {
                MatrizCustosRotas[i][j] = newW[i][j];
            }

        }
    }

    private String pad(int distance) {
        String value = String.valueOf(distance);
        int pad = 3 - value.length();
        for (int i = 0; i < pad; i++) {
            value = " " + value;
        }
        return value;
    }


    private void printMatrix(int[][] matrix, int matrixSize, int matrixNumber) {
        System.out.print("D(" + matrixNumber + ") = ");
        for (int i = 0; i < matrixSize; i++) {
            if (i != 0) {
                System.out.print(" ");     }
            System.out.print("[ ");
            if (matrix[i][0] == infinity) {
                System.out.print(infinityString);
            } else {
                System.out.print(pad(matrix[i][0]));
            }
            for (int j = 1; j < matrixSize; j++) {
                System.out.print(", ");
                if (matrix[i][j] == infinity) {
                    System.out.print(infinityString);
                } else {
                    System.out.print(pad(matrix[i][j]));        }      }
            System.out.println(" ]");
        }
        System.out.println(" ");    }

    private void printMatrixPrecedentes(int[][] matrix, int matrixSize, int matrixNumber) {
        System.out.print("P(" + matrixNumber + ") = ");
        for (int i = 0; i < matrixSize; i++) {
            if (i != 0) {
                System.out.print(" ");            }
            System.out.print("[ ");
            if (matrix[i][0] == infinity) {
                System.out.print(infinityString);
            } else {
                System.out.print(pad(matrix[i][0]));            }
            for (int j = 1; j < matrixSize; j++) {
                System.out.print(", ");
                if (matrix[i][j] == infinity) {
                    System.out.print(infinityString);
                } else {
                    System.out.print(pad(matrix[i][j]));         }        }
            System.out.println(" ]");        }
        System.out.println(" ");    }

    public static void imprimeArranjo(ArrayList<Integer> a) {
        int[] indices;
```

```
            int flag = 0;
            PermutationGenerator x = new PermutationGenerator(a.size());
            StringBuffer permutation;
            System.out.println("");
            System.out.println("------------------------------");
            System.out.println("");
            while (x.hasMore()) {
               permutation = new StringBuffer();
               indices = x.getNext();
               for (int i = 0; i < indices.length; i++) {
                  if ((a.get(indices[0]) == a.get(0))) {
                     flag = 1;
                     permutation.append(a.get(indices[i]));
                  } else {
                     flag = 0;              }          }
               if (flag == 1) {
                  System.out.println(permutation.toString());   }   }  }

         public static void ImprimePercurso(ArrayList<Integer> a, Integer inicio) {
            int[] indices;
            int custo = 0;
            int custop = 0;
            int menor = infinity;
            percursoSimles = new int[a.size()];
            percursoIdeal = new int[a.size()];
            percurso = new ArrayList<Integer>();
            pares = new int[a.size()][2];


            System.out.println("");
            System.out.println("------------------------------");
            System.out.println("");

            PermutationGenerator x = new PermutationGenerator(a.size());

            while (x.hasMore()) {
               indices = x.getNext();
               for (int i = 0; i < indices.length - 1; i++) {
                  if ((a.get(indices[0]) == a.get(0))) {
                     pares[i][0] = a.get(indices[i]);
                     System.out.print(pares[i][0]);
                     pares[i][1] = a.get(indices[i + 1]);
                     if (percursoSimles.length == indices.length) {
                        percursoSimles[i] = pares[i][0];
                        percursoSimles[i + 1] = pares[i][1];
                     }
                     System.out.print("-" + pares[i][1]);
                     System.out.print(" ");
                     if (MatrizCustosRotas[pares[i][0] - 1][pares[i][1] - 1] != infinity) {
                        custop = MatrizCustosRotas[pares[i][0] - 1][pares[i][1] - 1];
                     } else {
                        custop = 0;                  }
                     System.out.print("Custo parcial : " + custop + " ");
                     custo += custop;
                     if (((i + 1) % 3 == 0)) {
                        if (custo < menor) {
                           menor = custo;
                           for (int z = 0; z < percursoSimles.length; z++) {
                              percursoIdeal[z] = percursoSimles[z];      }     }
                        System.out.print(" - Custo: " + custo);
                        System.out.println("");
                        custo = 0;     }   }   }   }
```

```
        System.out.println("------------------------------");
        System.out.println("");
        System.out.print("Caminho de menor percurso: ");
        for (int z = 0; z < percursoIdeal.length; z++) {
            System.out.print(percursoIdeal[z]);       }
        System.out.println("");
        System.out.println("------------------------------");

        System.out.println("");
        System.out.println("------------------------------");
        System.out.println("O menor custo encontrado foi: " + menor);
        System.out.println("");

        System.out.println("------------------------------");
        System.out.print("Sub-percurso escolhido: ");

        //busca(4, 2);

        for (int i = 0; i < percursoIdeal.length - 1; i++) {
            busca((percursoIdeal[i]), (percursoIdeal[i + 1]));       }

        for (int i = 0; i < percurso.size(); i++) {
            if (((i + 1) < percurso.size()) && (percurso.get(i) != percurso.get(i + 1))) {
                System.out.print(percurso.get(i));           }
            if ((i + 1) == percurso.size()) {
                System.out.print(percurso.get(i));           }       }
        System.out.println("");    }

    public static void busca(int o, int d) {
        if (o == d) {
            //System.out.print(o);
            percurso.add(o);
        } else if (pred[o - 1][d - 1] == 0) {
            System.out.println("Nenhum caminho de " + o + " para " + d + " existente!");
        } else {
            busca(o, pred[o - 1][d - 1]);
            //System.out.print(d);
            percurso.add(d);       }    }

    public static void main(String[] args) {

        float tempo = 0;
        ArrayList<Float> resultados = new ArrayList<Float>();

        matrixSize = 11;
        System.out.println("TAMANHO DA MATRIZ:  " + matrixSize);
        Rows = matrixSize;
        Columns = matrixSize;
        MatrizOriginal = new int[Rows][Columns];
        pred = new int[Rows][Columns];
        MatrizOriginal[0][0] = 0;               MatrizOriginal[0][1] = 8;
        MatrizOriginal[0][2] = 2;               MatrizOriginal[0][3] = infinity;
        MatrizOriginal[0][4] = infinity;        MatrizOriginal[0][5] = 2;
        MatrizOriginal[0][6] = infinity;        MatrizOriginal[0][7] = infinity;
        MatrizOriginal[0][8] = infinity;        MatrizOriginal[0][9] = infinity;
        MatrizOriginal[0][10] = infinity;       MatrizOriginal[1][0] = infinity;
        MatrizOriginal[1][1] = 0;               MatrizOriginal[1][2] = infinity;
        MatrizOriginal[1][3] = infinity;        MatrizOriginal[1][4] = 1;
        MatrizOriginal[1][5] = infinity;        MatrizOriginal[1][6] = infinity;
        MatrizOriginal[1][7] = infinity;        MatrizOriginal[1][8] = infinity;
        MatrizOriginal[1][9] = infinity;        MatrizOriginal[1][10] = infinity;
```

```
MatrizOriginal[2][0] = infinity;        MatrizOriginal[2][1] = infinity;
MatrizOriginal[2][2] = 0;               MatrizOriginal[2][3] = 3;
MatrizOriginal[2][4] = infinity;        MatrizOriginal[2][5] = infinity;
MatrizOriginal[2][6] = infinity;        MatrizOriginal[2][7] = infinity;
MatrizOriginal[2][8] = infinity;        MatrizOriginal[2][9] = infinity;
MatrizOriginal[2][10] = infinity;       MatrizOriginal[3][0] = infinity;
MatrizOriginal[3][1] = infinity;        MatrizOriginal[3][2] = infinity;
MatrizOriginal[3][3] = 0;               MatrizOriginal[3][4] = infinity;
MatrizOriginal[3][5] = infinity;        MatrizOriginal[3][6] = infinity;
MatrizOriginal[3][7] = 1;               MatrizOriginal[3][8] = infinity;
MatrizOriginal[3][9] = infinity;        MatrizOriginal[3][10] = infinity;
MatrizOriginal[4][0] = 5;               MatrizOriginal[4][1] = infinity;
MatrizOriginal[4][2] = infinity;        MatrizOriginal[4][3] = 1;
MatrizOriginal[4][4] = 0;               MatrizOriginal[4][5] = infinity;
MatrizOriginal[4][6] = 4;               MatrizOriginal[4][7] = infinity;
MatrizOriginal[4][8] = infinity;        MatrizOriginal[4][9] = infinity;
MatrizOriginal[4][10] = infinity;       MatrizOriginal[5][0] = infinity;
MatrizOriginal[5][1] = infinity;        MatrizOriginal[5][2] = infinity;
MatrizOriginal[5][3] = infinity;        MatrizOriginal[5][4] = 9;
MatrizOriginal[5][5] = 0;               MatrizOriginal[5][6] = 2;
MatrizOriginal[5][7] = infinity;        MatrizOriginal[5][8] = infinity;
MatrizOriginal[5][9] = infinity;        MatrizOriginal[5][10] = infinity;
MatrizOriginal[6][0] = infinity;        MatrizOriginal[6][1] = infinity;
MatrizOriginal[6][2] = infinity;        MatrizOriginal[6][3] = infinity;
MatrizOriginal[6][4] = infinity;        MatrizOriginal[6][5] = infinity;
MatrizOriginal[6][6] = 0;               MatrizOriginal[6][7] = infinity;
MatrizOriginal[6][8] = infinity;        MatrizOriginal[6][9] = 4;
MatrizOriginal[6][10] = infinity;       MatrizOriginal[7][0] = infinity;
MatrizOriginal[7][1] = infinity;        MatrizOriginal[7][2] = infinity;
MatrizOriginal[7][3] = infinity;        MatrizOriginal[7][4] = 8;
MatrizOriginal[7][5] = infinity;        MatrizOriginal[7][6] = 1;
MatrizOriginal[7][7] = 0;               MatrizOriginal[7][8] = 1;
MatrizOriginal[7][9] = 7;               MatrizOriginal[7][10] = infinity;
MatrizOriginal[8][0] = infinity;        MatrizOriginal[8][1] = infinity;
MatrizOriginal[8][2] = infinity;        MatrizOriginal[8][3] = 3;
MatrizOriginal[8][4] = infinity;        MatrizOriginal[8][5] = infinity;
MatrizOriginal[8][6] = infinity;        MatrizOriginal[8][7] = infinity;
MatrizOriginal[8][8] = 0;               MatrizOriginal[8][9] = infinity;
MatrizOriginal[8][10] = 1;              MatrizOriginal[9][0] = infinity;
MatrizOriginal[9][1] = infinity;        MatrizOriginal[9][2] = infinity;
MatrizOriginal[9][3] = infinity;        MatrizOriginal[9][4] = infinity;
MatrizOriginal[9][5] = infinity;        MatrizOriginal[9][6] = infinity;
MatrizOriginal[9][7] = infinity;        MatrizOriginal[9][8] = infinity;
MatrizOriginal[9][9] = 0;               MatrizOriginal[9][10] = 1;
MatrizOriginal[10][0] = infinity;       MatrizOriginal[10][1] = infinity;
MatrizOriginal[10][2] = infinity;       MatrizOriginal[10][3] = infinity;
MatrizOriginal[10][4] = infinity;       MatrizOriginal[10][5] = infinity;
MatrizOriginal[10][6] = infinity;       MatrizOriginal[10][7] = 1;
MatrizOriginal[10][8] = infinity;       MatrizOriginal[10][9] = infinity;
MatrizOriginal[10][10] = 0;
new FloydWarshallAperfeicoado();
ArrayList<Integer> a = new ArrayList<Integer>();
a.add(10);//nó inicial       a.add(11);       a.add(5);
a.add(8);       a.add(9);       a.add(4);       a.add(1);
//10-5-4-7-6-1-3
for (int i = 0; i < 10; i++) {
    Chronometer.start();
    imprimeArranjo(a);
    ImprimePercurso(a, a.get(2));
    Chronometer.stop();
    tempo = Chronometer.elapsedTime()/1000000;
    resultados.add(tempo);
```

```
                    //long fim = System.currentTimeMillis();

                    System.out.println("Terminada a execução de Floyd-Warshall com
aperfeiçoamento.");
                    //System.out.println("Tempo decorrido: " + (fim - inicio) + "ms");
                    System.out.println("Tempo decorrido: " + tempo + "ms");
                }

                for (int i = 0; i < resultados.size(); i++) {
                    System.out.println(resultados.get(i));     }   }  }
```

Apêndice G

Main.java (classe principal Java3d)

```
import java.awt.Color;
import java.awt.DisplayMode;
import java.awt.Graphics2D;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.Point;
import java.awt.geom.GeneralPath;
import java.awt.image.BufferStrategy;
import javax.swing.ImageIcon;
import javax.swing.JFrame;

public class main extends JFrame {
   public main() {
      setUndecorated(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setResizable(false);
      setIgnoreRepaint(true);
      myDevice.setFullScreenWindow(this);
      myDevice.setDisplayMode(myMode);
      createBufferStrategy(2);
      addKeyListener(mover);
      setCursor(getToolkit().getDefaultToolkit().createCustomCursor(new     ImageIcon("
").getImage(), new Point(0, 0), "invisble"));
      Loop();  }
   public static void main(String[] args) {
      main m = new main();  }
   public void Loop() {
      BufferStrategy myBuffer = getBufferStrategy();
      while (true) {
         Graphics2D g = (Graphics2D) myBuffer.getDrawGraphics();
         draw(g);
         myBuffer = getBufferStrategy();
         g.dispose();
         myBuffer.show();
         try {
            Thread.sleep(10);
         } catch (InterruptedException ex) {
            ex.printStackTrace();
         } } }
   public void draw(Graphics2D g2d) {
      GeneralPath myP = new GeneralPath();
      myDrawer.draw3D(g2d, myCube.getPolygons());
      g2d.setColor(Color.WHITE);
      g2d.drawString("Pressione  SETA  CIMA(ZOOM  +)   /   SETA  BAIXO(ZOOM  -)  /
ESC ", 20.0f, 20.0f);
      g2d.drawString("Pressione  Q/A Rotação no eixo Z  ", 20.0f, 40.0f);
      g2d.drawString("Pressione  W/S Rotação no eixo X  ", 20.0f, 60.0f);
      g2d.drawString("Pressione  E/D Rotação no eixo Y  ", 20.0f, 80.0f);
      g2d.drawString("Empacotamento 3D: Solução otimizada  ", 20.0f, 100.0f);   }
```

```
        private final int width = 800;
        private final int height = 600;
        private GraphicsEnvironment myenv = GraphicsEnvironment.getLocalGraphicsEnvironment();
        private GraphicsDevice myDevice = myenv.getDefaultScreenDevice();
        private DisplayMode myMode = new DisplayMode(width, height, 32, DisplayMode.REFRESH_RATE_UNKNOWN);
        private Vetor3D[] v = new Vetor3D[]{new Vetor3D(0, 0, 0), new Vetor3D(100, 0, 0), new Vetor3D(100, 150, 0), new Vetor3D(0, 150, 0)};
        private Poligono3D myPol = new Poligono3D(v);
        private Poligono3D drawPol = new Poligono3D();
        private Transforma3D trans = new Transforma3D(0.0f, -100.0f, -400.0f);
        private JanelaVisualizacao myWindow = new JanelaVisualizacao(0, 0, width, height, (float) Math.toRadians(75));
        private Movimento mover = new Movimento(trans);
        private Desenho3D myDrawer = new Desenho3D(width, height, trans, myWindow);
        private Cubo3D myCube = new Cubo3D(200);
    }
```