GENETIC PACKING: software para empacotamento tridimensional heterogêneo em contêineres

Daniel Pamplona Soares – Acadêmico
Daniel Theisges dos Santos - Orientador

Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

Introdução

- Inicialmente ligada as operações militares;
- Uma indústria precisa transportar seus produtos de maneira eficaz;
- Valor do tempo (custos mais elevados e prazos mais curtos);

Introdução

- Problema amplamente estudado, pois possui interesses práticos e relevantes em aplicações industriais.

Objetivos

- Resolver o problema de empacotamento tridimensional heterogêneo utilizando AG (Algoritmo Genético).
  - Disponibilizar seleção de pacotes com tamanhos variáveis;
  - Disponibilizar seleção de tamanho de contêiner;
  - Realizar a alocação de pacotes dentro de contêineres considerando o maior preenchimento.

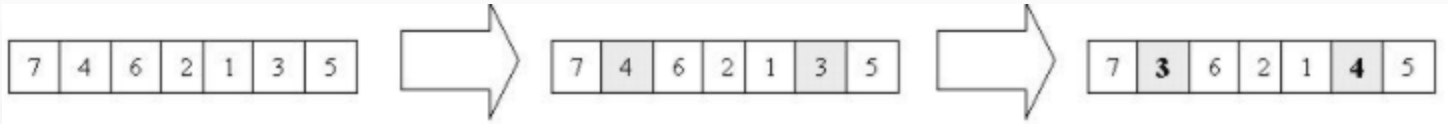
Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

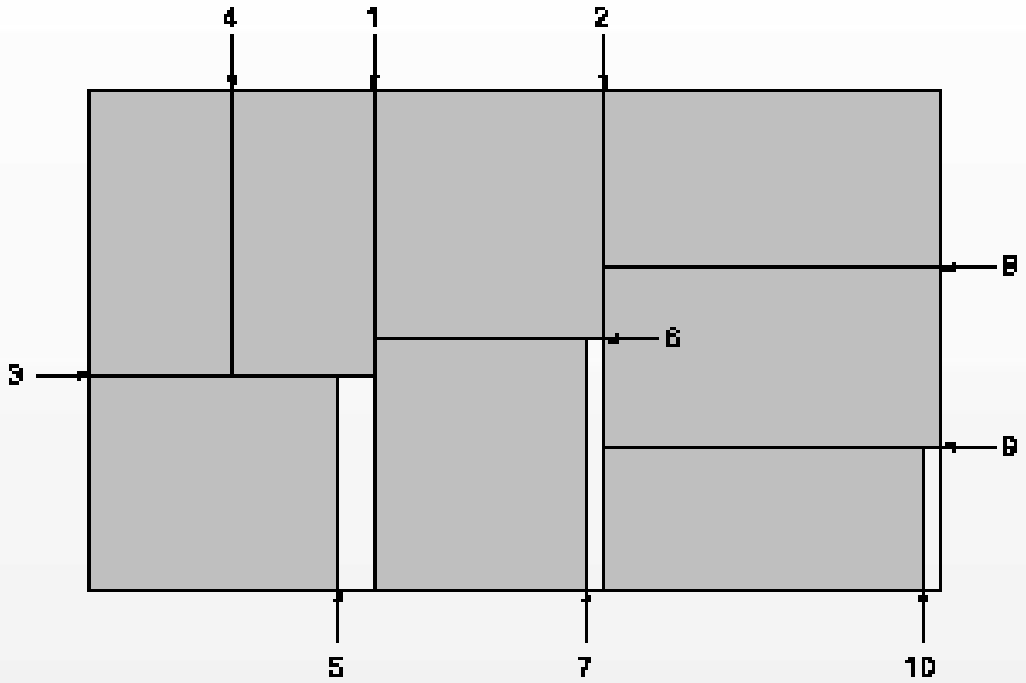
Problema de empacotamento

- NP-Difícil;
- Diversos tipos;
  - Unidimensional;
  - Bidimensional;
  - Tridimensional;
  - N-Dimensional.

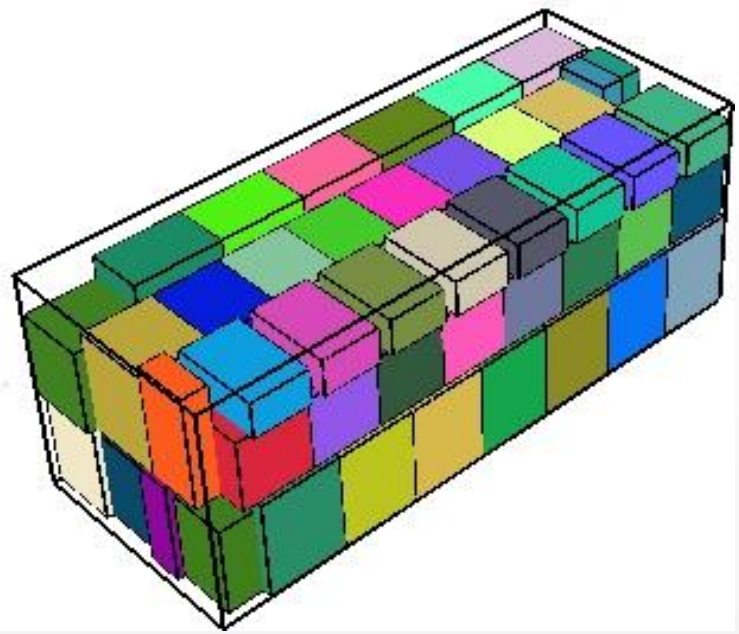
Unidimensional



Bidimensional



Tridimensional



Problema de empacotamento

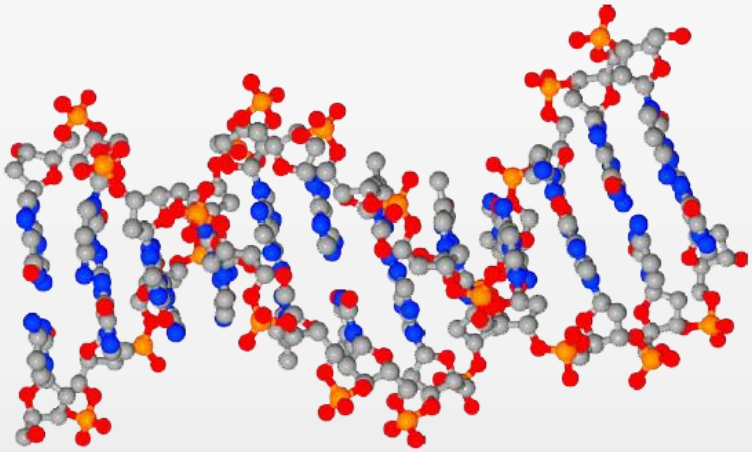
- Pode ser classificado por:
  - Dimensão;
  - Tipo de tarefa;
    - Diferenças dos tamanhos de entrada e saída.
  - Variedade dos objetos de entrada;
  - Variedade dos objetos de saída.
- Pode gerar subproblemas:
  - Agendamento, Fragilidade, Contêineres diversos.

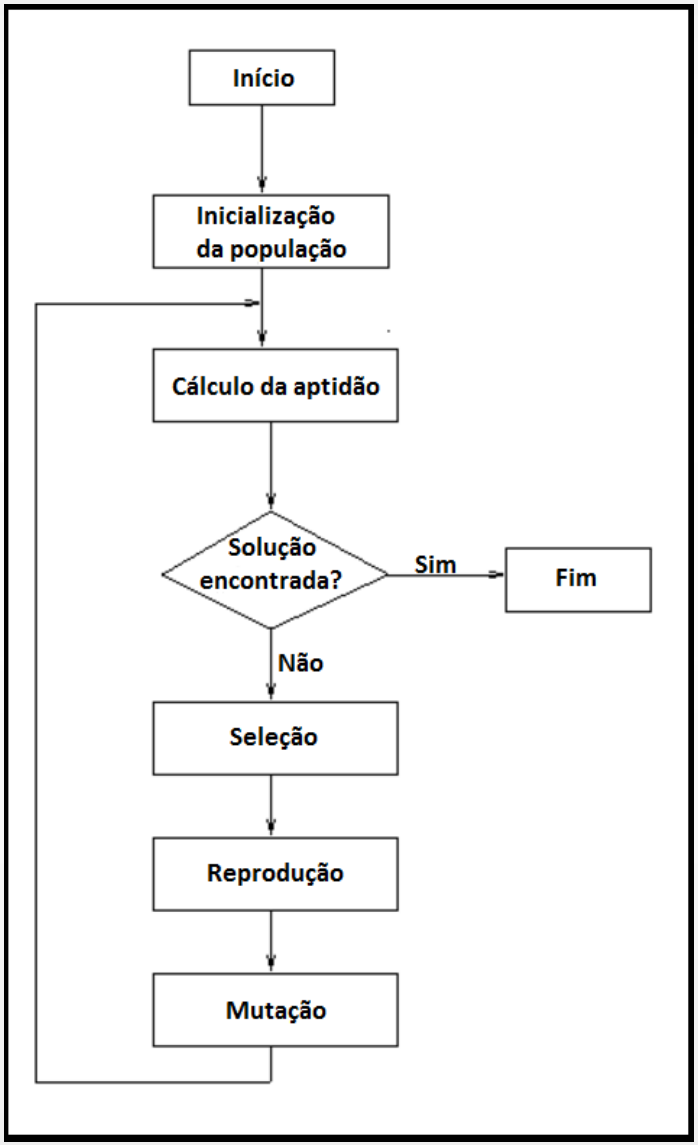
Algoritmos de aproximação

- Utilizado quando o caminho torna-se irrelevante;
- Problemas de otimização;
- Seu objetivo é o melhor estado final.

Algoritmos Genéticos

- Ramo dos Algoritmos Evolucionários;
- Baseado no processo biológico de evolução natural;
- Visa encontrar o máximo de uma solução.

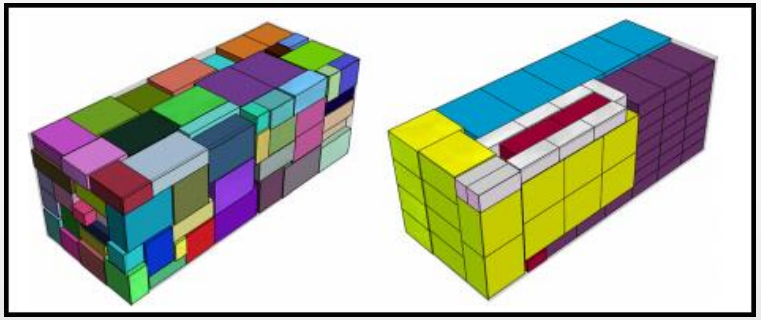




Fonte: Adaptado de Miranda (2011).

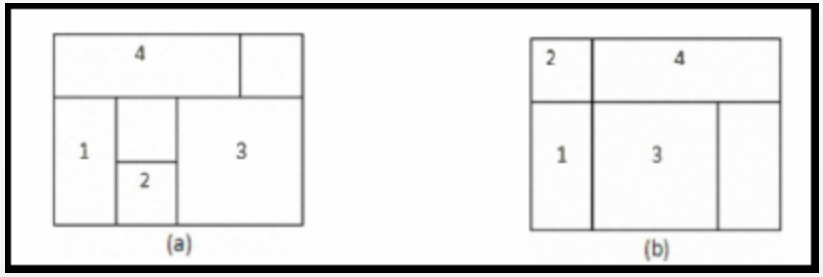
Trabalho correlato - 1

- A Biased Random Key Genetic Algorithm For 2D and 3D Bin Packing Problems (GONÇALVES, RESENDE, 2013);
- Chave genética randômica, visando resolver problemas industriais;
- Mantém os contêineres abertos enquanto existirem pacotes não encaixados.



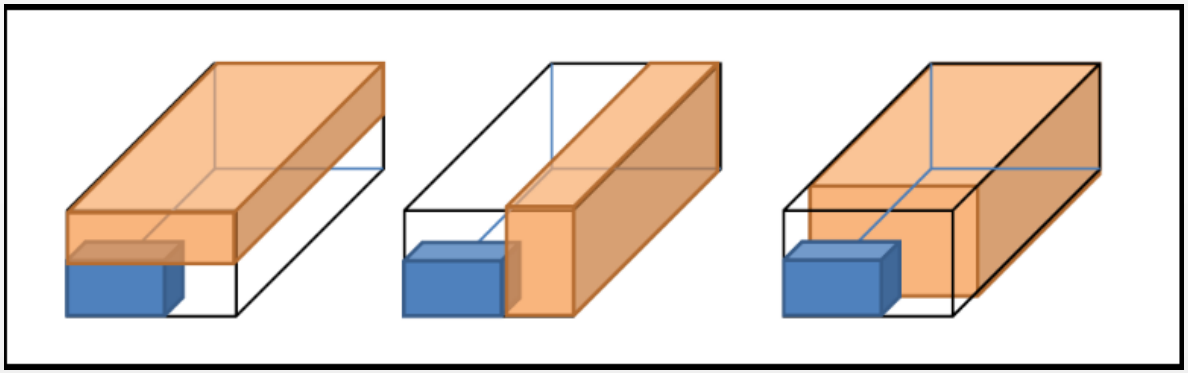
Trabalho correlato - 2

- A Hybrid Genetic Algorithm For 3D Bin Packing Problems (WANG, CHEN, 2010);
- Utilizou mesclagem de espaços vazios.



Trabalho correlato - 3

- A Genetic Algorithm For The Three-Dimensional Bin Packing Problem With Heterogeneous Bins (LI; ZHAO; ZHANG, 2014);
- Utilizou conceito de máximo espaço disponível (EMS – Empty Maximal Spaces)



Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

Requisitos Funcionais

- RF01 - Permitir a seleção de tamanho do contêiner;
- RF02 - Permitir seleção de tamanho dos pacotes;
- RF03 - Otimizar a alocação de pacotes dentro do contêiner, alocando a maior quantidade de pacotes possíveis;
- RF04 - Disponibilizar uma tela para configuração dos pacotes e do contêiner;

Requisitos Funcionais

- RF05 - Disponibilizar uma tela para exibir a aptidão atual atingida e o número de gerações do algoritmo;
- RF06 - Disponibilizar uma tela em três dimensões para visualização do resultado do algoritmo;
- RF07 - Permitir a rotação de câmera na exibição.

Requisitos Não-Funcionais

- RNF01 - Ser implementado utilizando Algoritmos Genéticos ;
- RNF02 - Ser implementado utilizando a plataforma Java 8 ;
- RNF03 - Ser implementado utilizando OpenGL.

Diagrama de casos de uso

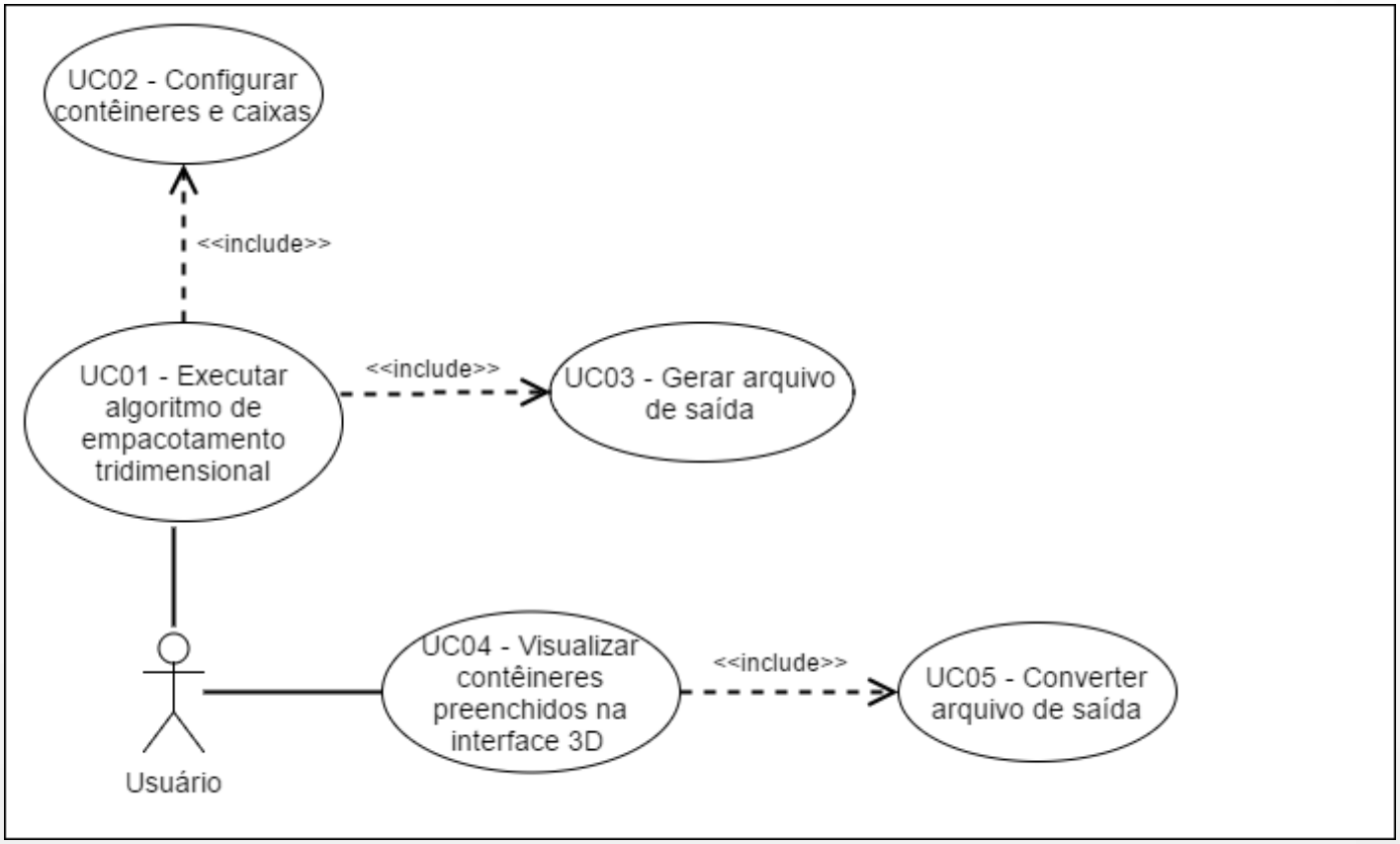


Diagrama de classes - Modelagem

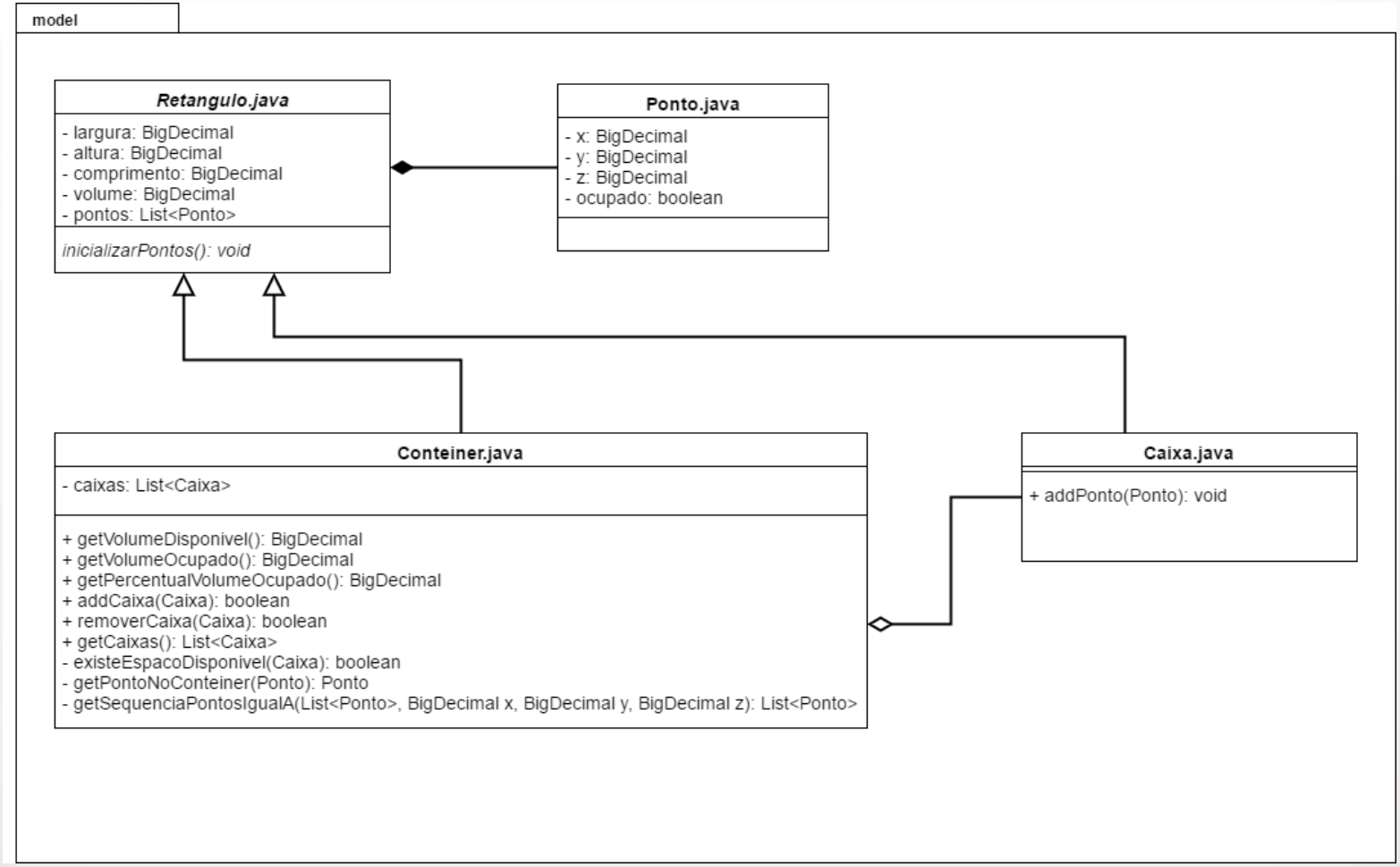


Diagrama de classes - AG

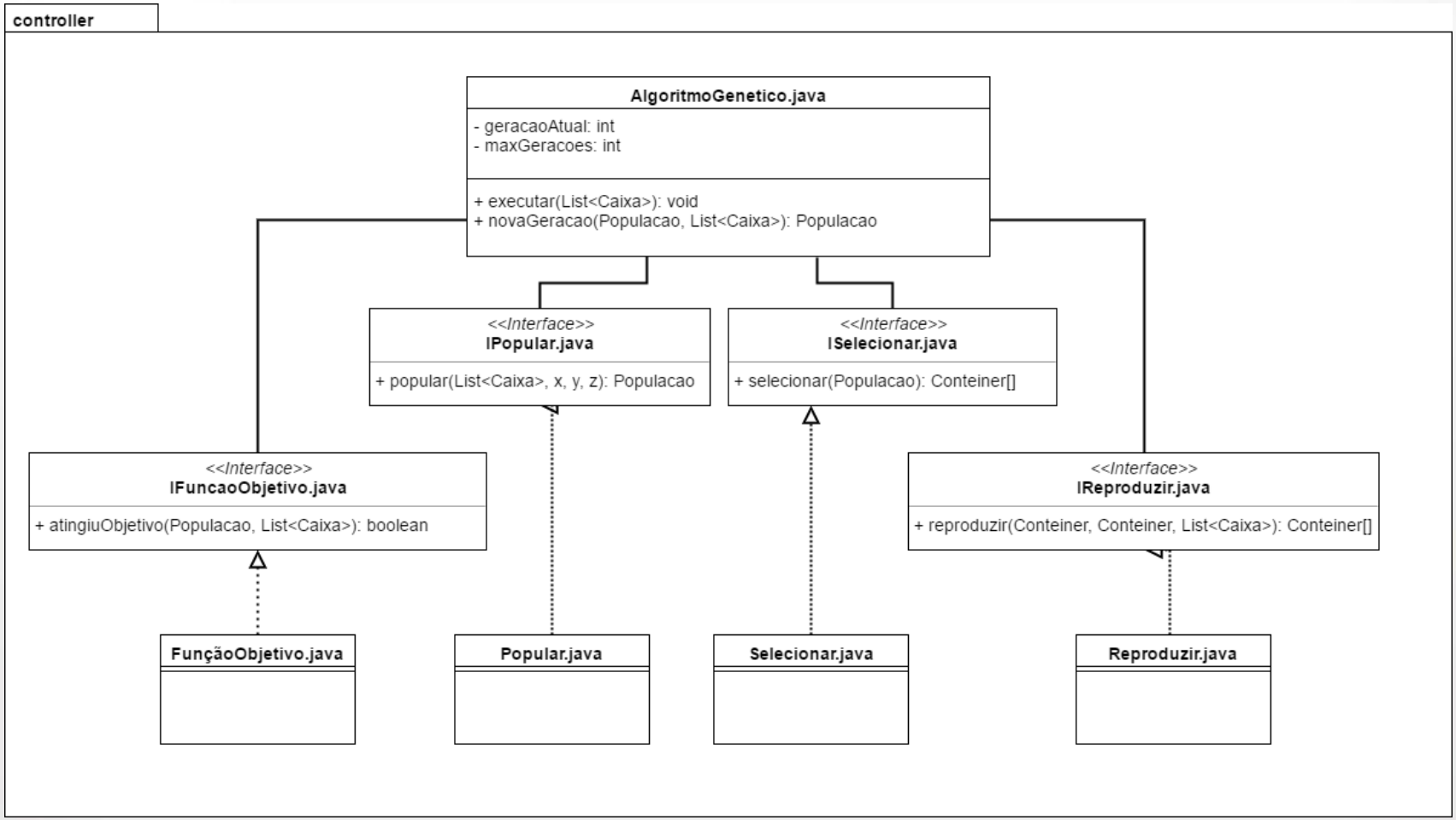
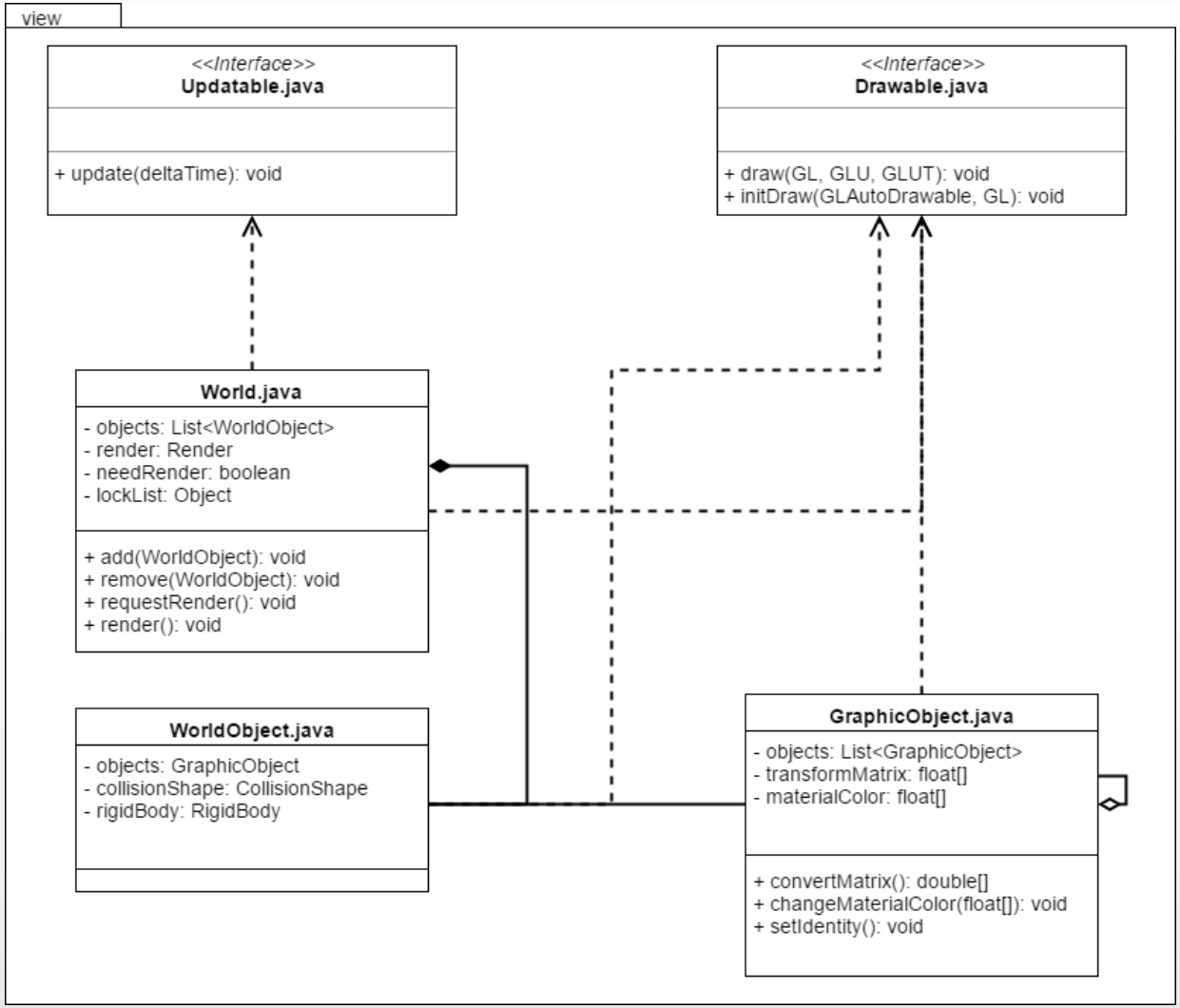
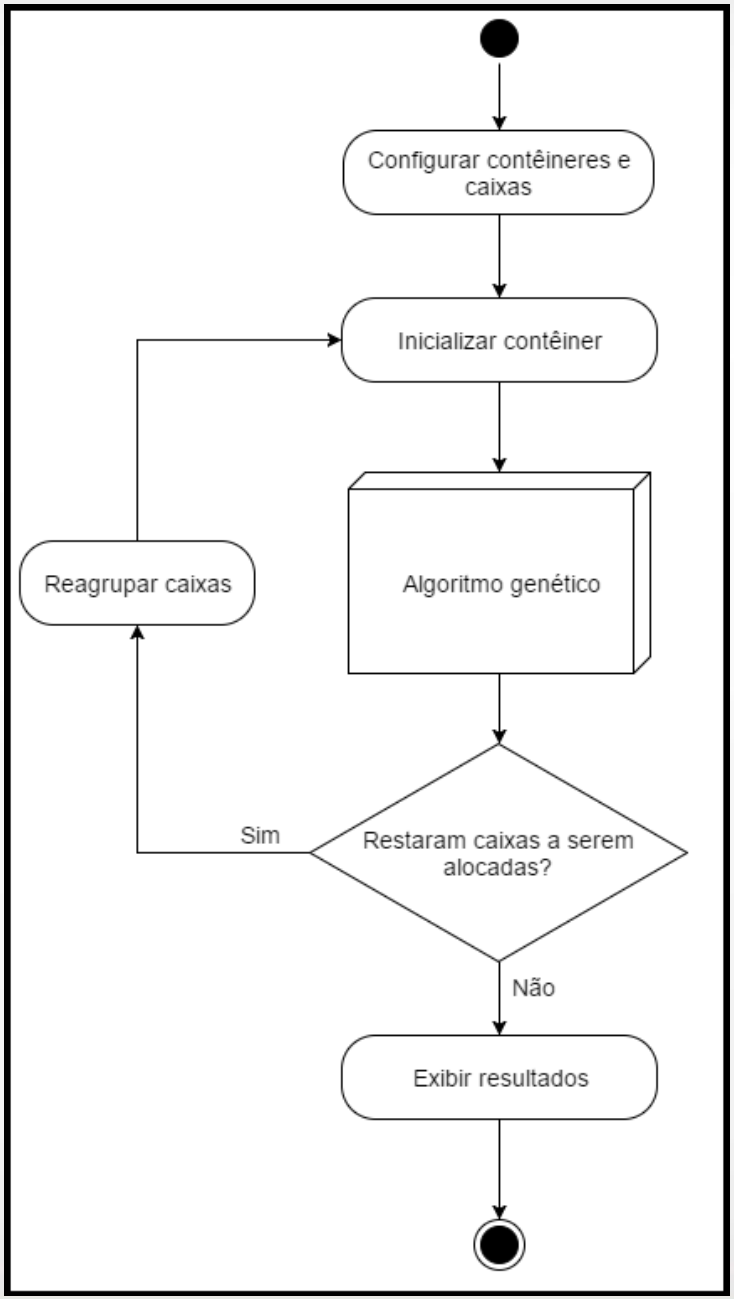
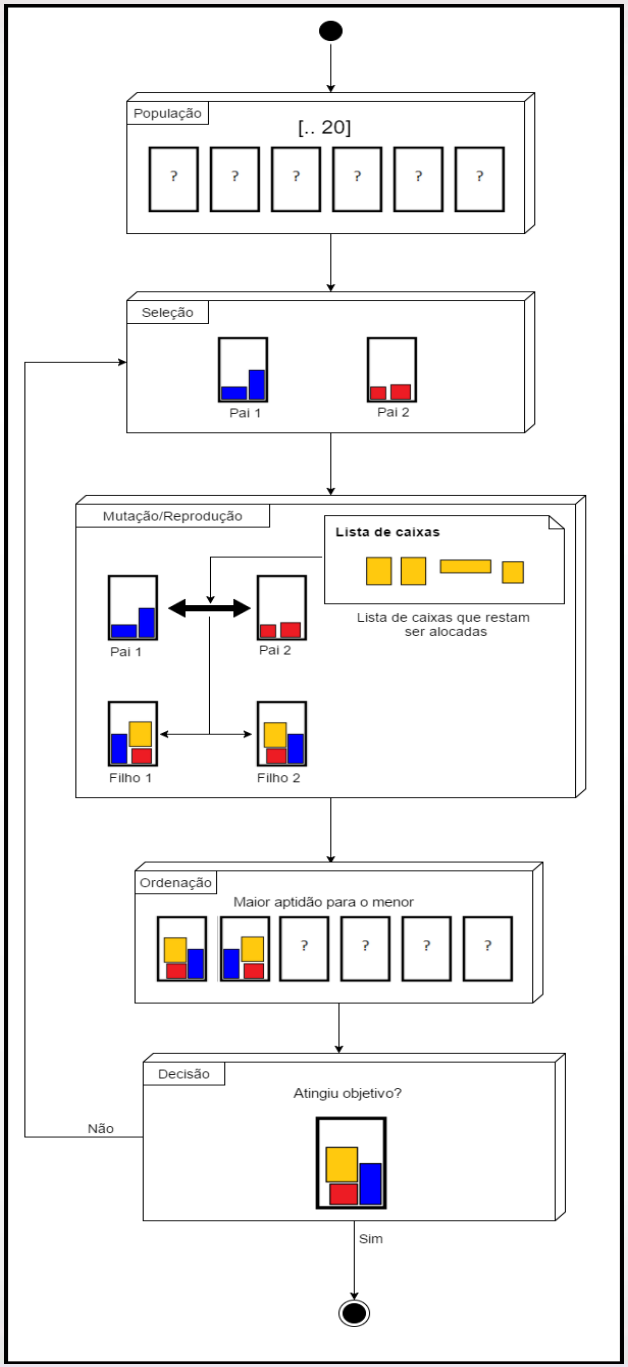


Diagrama de classes - View




Configurar contêineres e caixas
Inicializar contêiner
Algoritmo genético
Restaram caixas a serem alocadas?
Sim
Reagrupar caixas
Não
Exibir resultados


População
[.. 20]
?
?
?
?
?
?
Seleção
Pai 1
Pai 2
Mutação/Reprodução
Lista de caixas
Lista de caixas que restam ser alocadas
Pai 1
Pai 2
Filho 1
Filho 2
Ordenação
Maior aptidão para o menor
?
?
?
?
Decisão
Atingiu objetivo?
Não
Sim

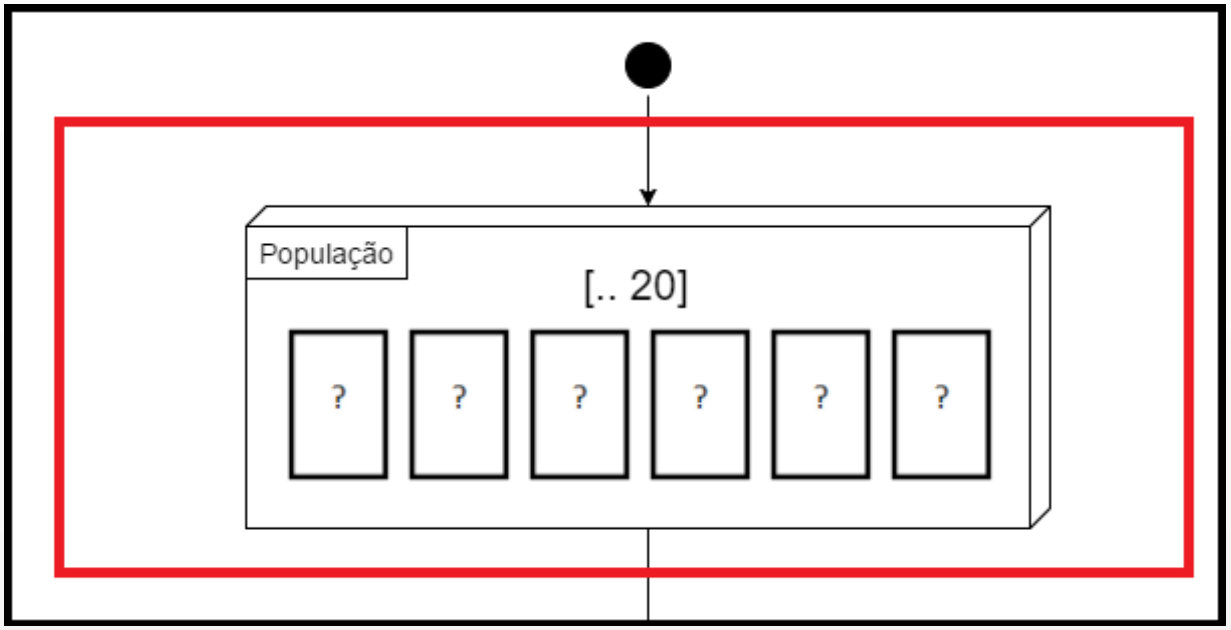
Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

Implementação

```
Populacao populacao = AlgoritmoGenetico.this.metodoPopulacao.popular(listaCaixas, x, y, z);
while (!AlgoritmoGenetico.this.funcaoObjetivo.atingiuObjetivo(melhorIndividuo, listaCaixas) && //
AlgoritmoGenetico.this.geracaoAtual <= AlgoritmoGenetico.this.maxGeracoes) {
    telaExecucao.setGeracaoAtual(AlgoritmoGenetico.this.geracaoAtual);
    populacao = AlgoritmoGenetico.this.novaGeracao(populacao, listaCaixas);
    melhorIndividuo = populacao.getIndividuo(0);
    final BigDecimal aptidao = melhorIndividuo.getPercentualVolumeOcupado();
    telaExecucao.setMelhorIndividuo(melhorIndividuo);
    System.out.println("Geração " + geracaoAtual + " | Aptidão: " + aptidao + " | Qtd Caixas: " + melhorIndividuo.getCaixas().size());
    telaExecucao.setAptidaoAtual(aptidao);
    AlgoritmoGenetico.this.geracaoAtual++;
}
telaExecucao.setExecucaoAtual(execucaoAtual);
telaExecucao.setIsAcabou(true);
telaExecucao.setAtingiuObjetivo(AlgoritmoGenetico.this.geracaoAtual < AlgoritmoGenetico.this.maxGeracoes);
telaExecucao.gravarExibirResultados();
Utils.esperar(150);
if (!alocouTodasCaixas(listaCaixas, melhorIndividuo)) {
    filtrarCaixasAlocadas(listaCaixas, populacao);
    AlgoritmoGenetico novaExecucao = new AlgoritmoGenetico(metodoPopulacao, metodoReproducao, metodoSelecao, funcaoObjetivo, x, y, z);
    novaExecucao.setExecucaoAtual(execucaoAtual);
    novaExecucao.incrementarExecucaoAtual();
    novaExecucao.executar(listaCaixas);
} else {
    informarTempoDeExecucao();
}
```

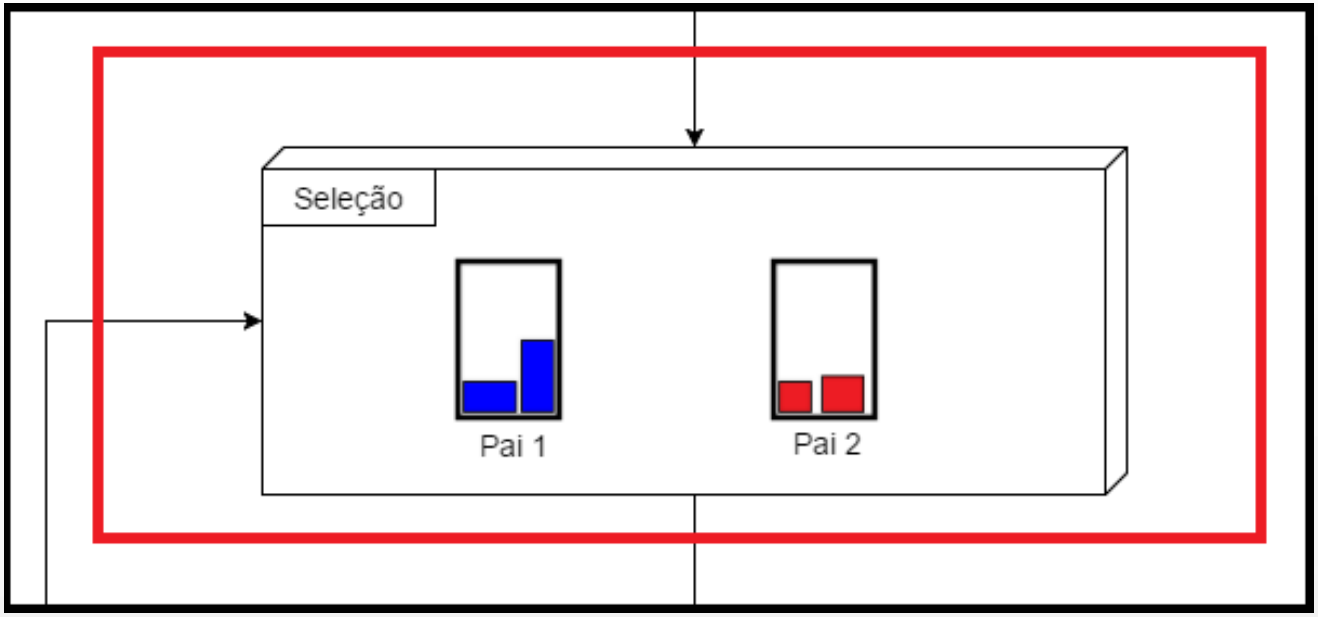
Algoritmos Genéticos – 1/5



Algoritmos Genéticos – 1/5

```
@Override
public Populacao popular(List<Caixa> listaCaixas, BigDecimal x, BigDecimal y, BigDecimal z) {
    final Populacao pop = new Populacao();
    while (pop.getTotalIndividuos() < Populacao.TAMANHO_POPULACAO) {
        final Conteiner c = new Conteiner(x, y, z);
        Collections.shuffle(listaCaixas);
        c.addCaixa(Utils.getCaixaNaoAlocadaDaLista(c, listaCaixas).clone());
        c.addCaixa(Utils.getCaixaNaoAlocadaDaLista(c, listaCaixas).clone());
        pop.addIndividuo(c);
    }
    pop.ordenarPopulacao();
    return pop;
}
```

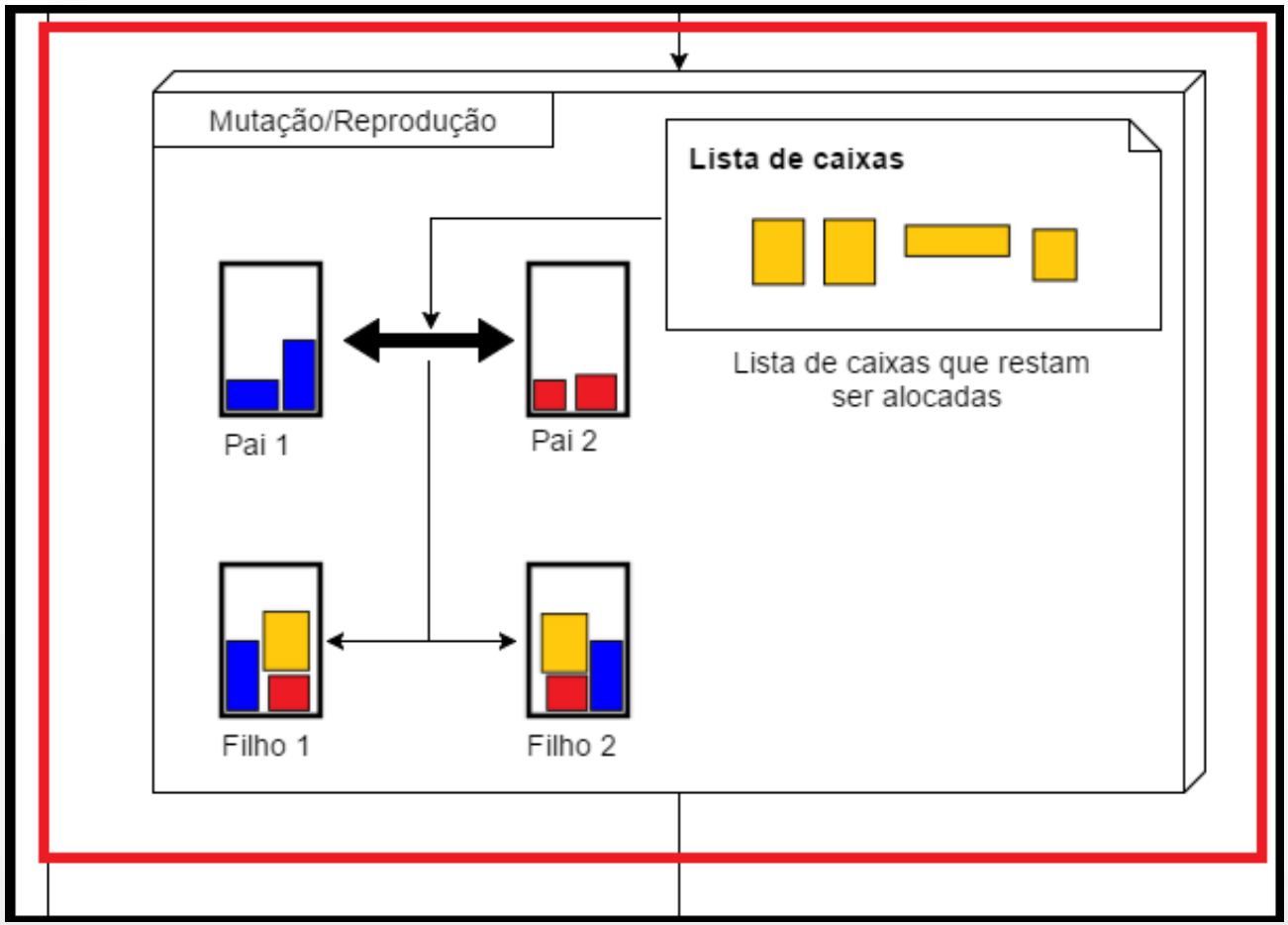
Algoritmos Genéticos – 2/5



Algoritmos Genéticos – 2/5

```
@Override
public Conteiner[] selecionar(Populacao populacao) {
    final Random r = new Random();
    final int totalIndividuos = populacao.getTotalIndividuos();

    final Conteiner[] pais = new Conteiner[2];
    pais[0] = populacao.getIndividuo(r.nextInt(totalIndividuos));
    pais[1] = populacao.getIndividuo(r.nextInt(totalIndividuos));

    return pais;

}
```

Algoritmos Genéticos – 3/5

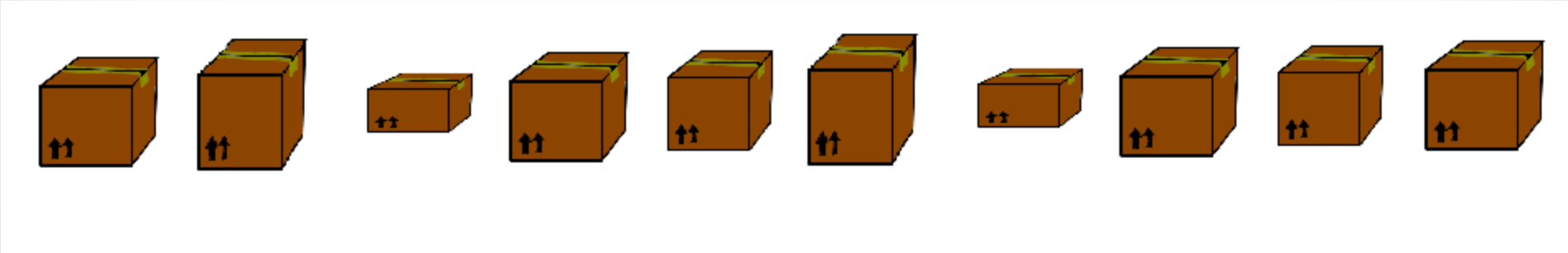


Algoritmos Genéticos – 3/5

```
@Override
public Conteiner[] reproduzir(Conteiner c1, Conteiner c2, List<Caixa> listaCaixas) {
    final ArrayList<Caixa> genesPai1 = (ArrayList<Caixa>) getListaClonada(c1.getCaixas());
    final ArrayList<Caixa> genesPai2 = (ArrayList<Caixa>) getListaClonada(c2.getCaixas());
    removerCaixasIguais(genesPai1, genesPai2);

    int pontoDeCorte1 = (int) (genesPai1.size() * PONTO_CORTE);
    int pontoDeCorte2 = (int) (genesPai2.size() * PONTO_CORTE);

    final Conteiner[] filhos = new Conteiner[2];
    filhos[0] = new Conteiner(c1.getLargura(), c1.getAltura(), c1.getComprimento());
    filhos[1] = new Conteiner(c2.getLargura(), c2.getAltura(), c2.getComprimento());

    mutar(pontoDeCorte1, genesPai1, filhos[0]);
    mutar(pontoDeCorte2, genesPai2, filhos[0]);
    mutar(pontoDeCorte2, genesPai2, filhos[1]);
    mutar(pontoDeCorte1, genesPai1, filhos[1]);

    int caixasAdd1 = (genesPai1.size() - pontoDeCorte1) + 1;
    int caixasAdd2 = (genesPai2.size() - pontoDeCorte2) + 1;
    for (int i = 0; i < caixasAdd1; i++) {
        filhos[0].addCaixa(Utils.getCaixaNaoAlocadaDaLista(filhos[0], listaCaixas).clone());
    }
    for (int i = 0; i < caixasAdd2; i++) {
        filhos[1].addCaixa(Utils.getCaixaNaoAlocadaDaLista(filhos[1], listaCaixas).clone());
    }
    return filhos;
}
```

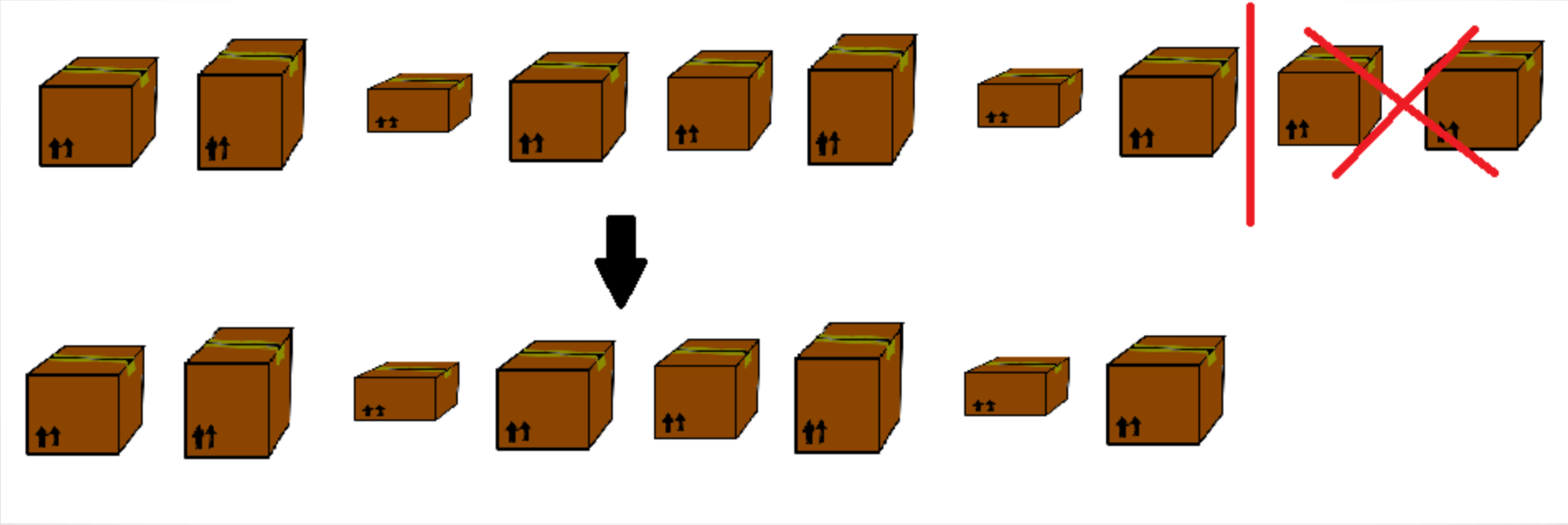
Ponto de corte

Genes:



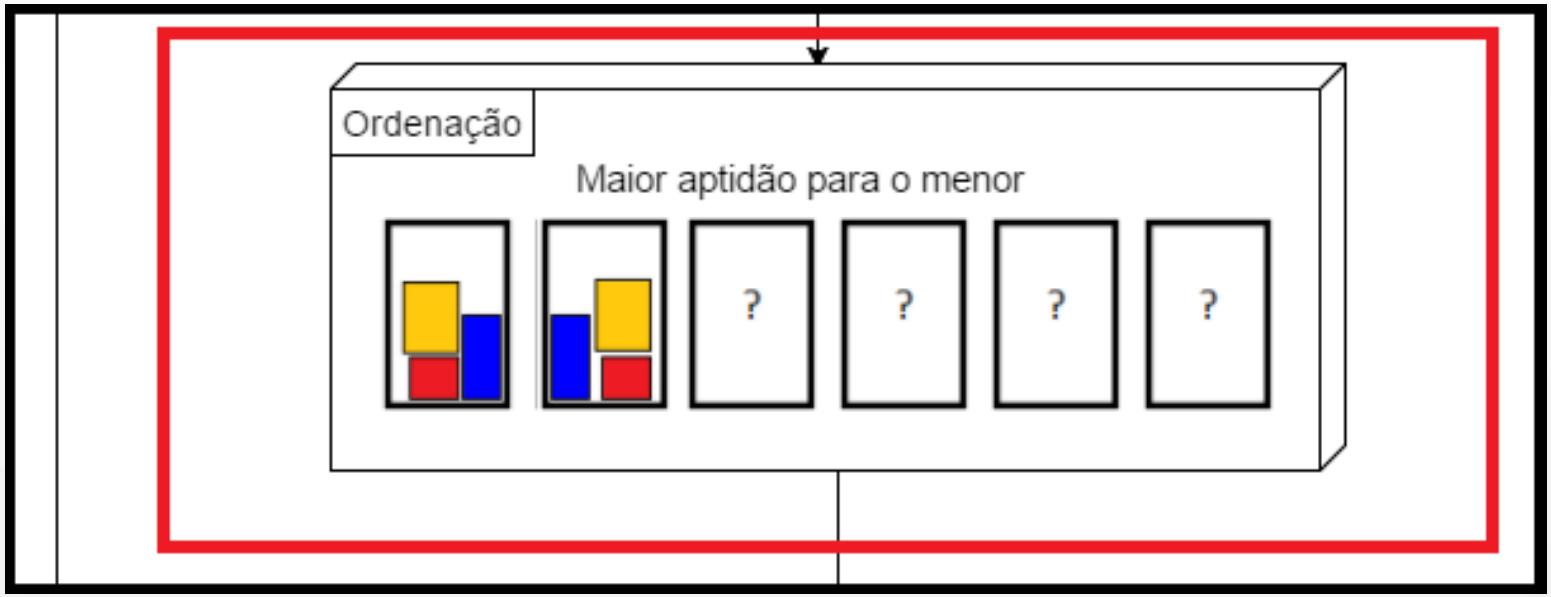
Ponto de corte

Genes:



- Duas estratégias implementadas

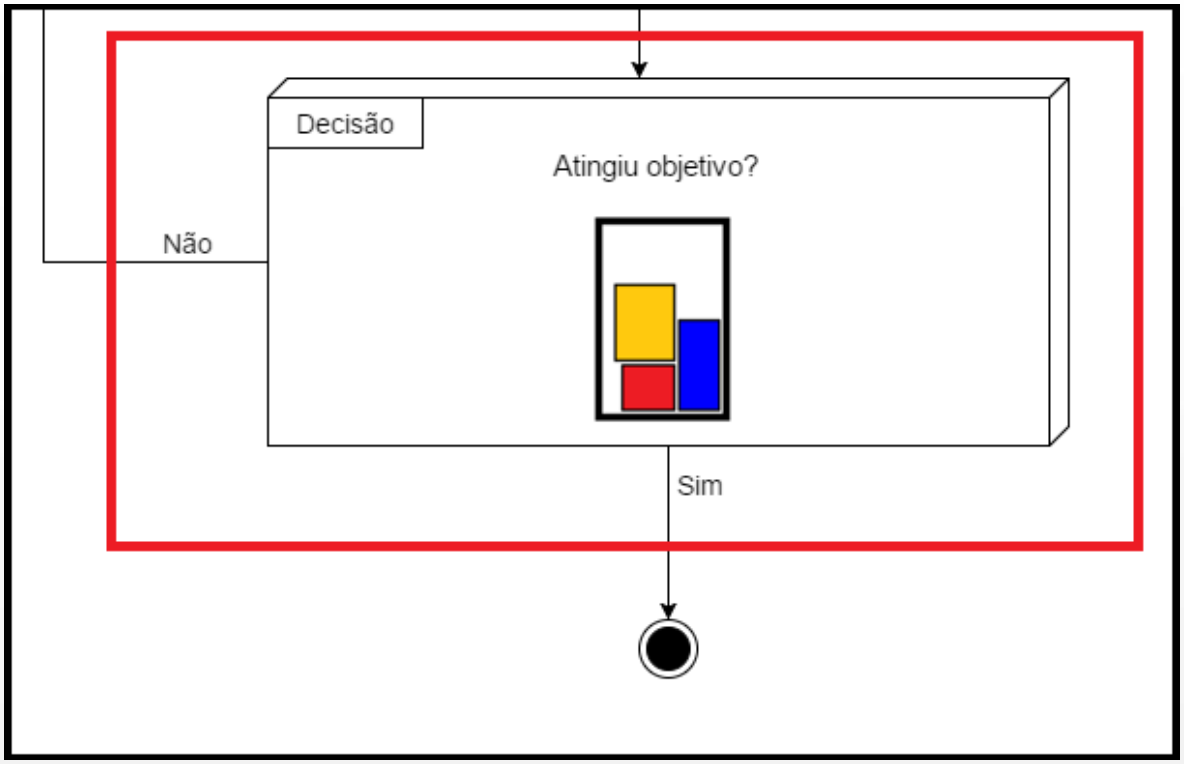
Algoritmos Genéticos – 4/5



Algoritmos Genéticos – 4/5

```
public void ordenarPopulacao() {
    boolean trocou = true;
    while (trocou) {
        trocou = false;
        for (int i = 0; i < individuos.length - 1; i++) {
            final Conteiner individuo1 = individuos[i];
            final Conteiner individuo2 = individuos[i + 1];
            final BigDecimal volume1 = individuo1 == null ? BigDecimal.ZERO : individuo1.getPercentualVolumeOcupado();
            final BigDecimal volume2 = individuo2 == null ? BigDecimal.ZERO : individuo2.getPercentualVolumeOcupado();
            if (Utils.isMenor(volume1, volume2)) {
                final Conteiner temp = individuo1;
                individuos[i] = individuo2;
                individuos[i + 1] = temp;
                trocou = true;
            }
        }
    }
}
```

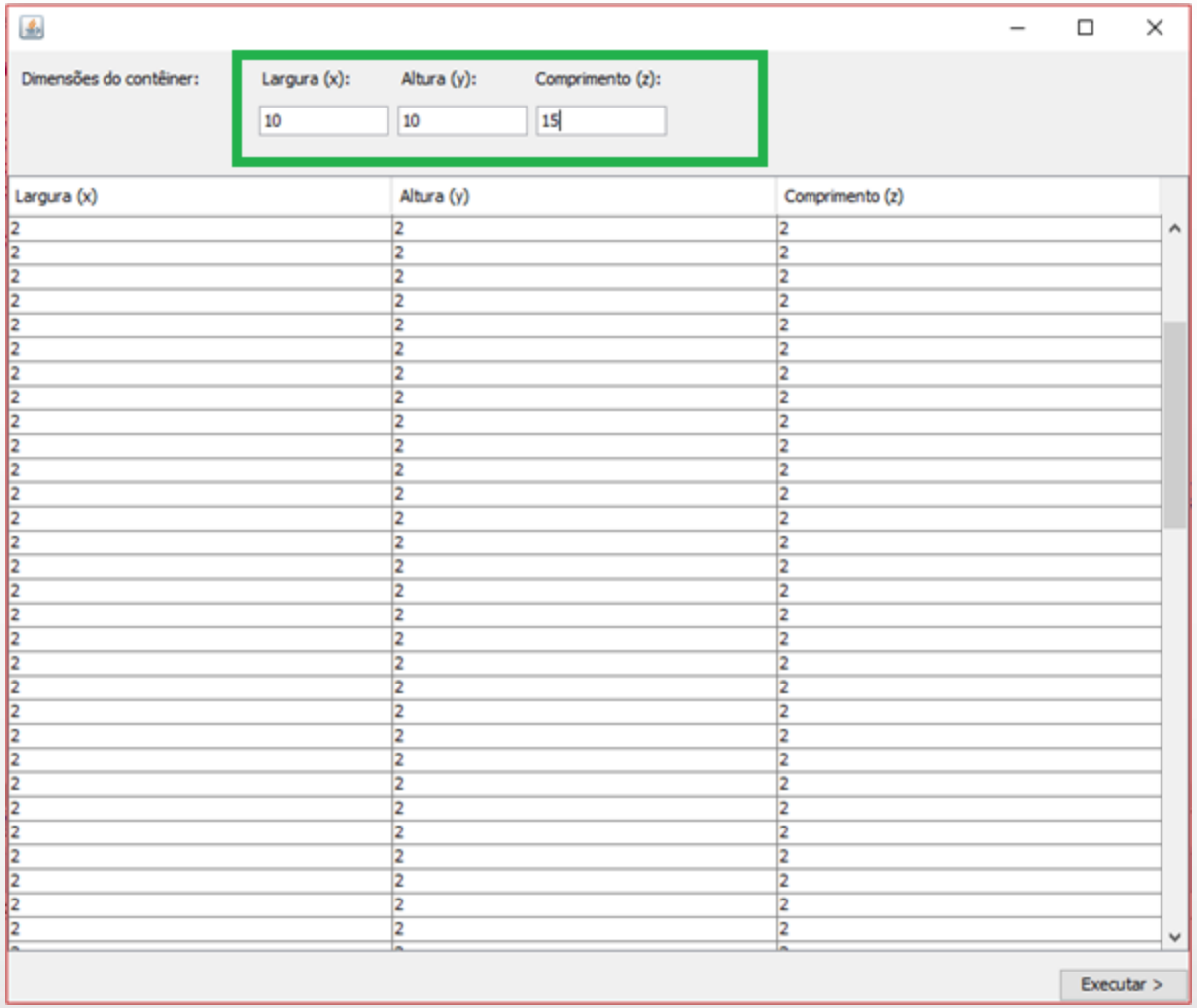
Algoritmos Genéticos – 5/5



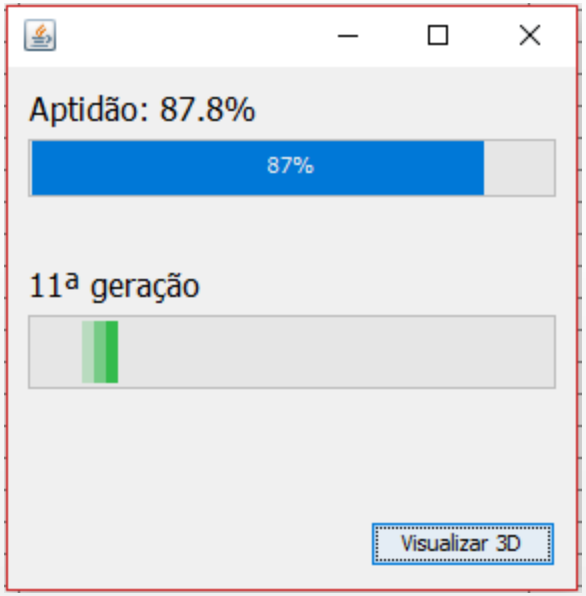
Algoritmos Genéticos – 5/5

```java
@Override
public boolean atingiuObjetivo(Conteiner melhorIndividuo, List<Caixa> listaCaixas) {
    return melhorIndividuo != null && (melhorIndividuo.atingiuTaxaVolume() || contemTodasCaixas(melhorIndividuo, listaCaixas));
}

private boolean contemTodasCaixas(Conteiner melhorIndividuo, List<Caixa> listaCaixas) {
    return melhorIndividuo.getCaixas().size() == listaCaixas.size();
}
```

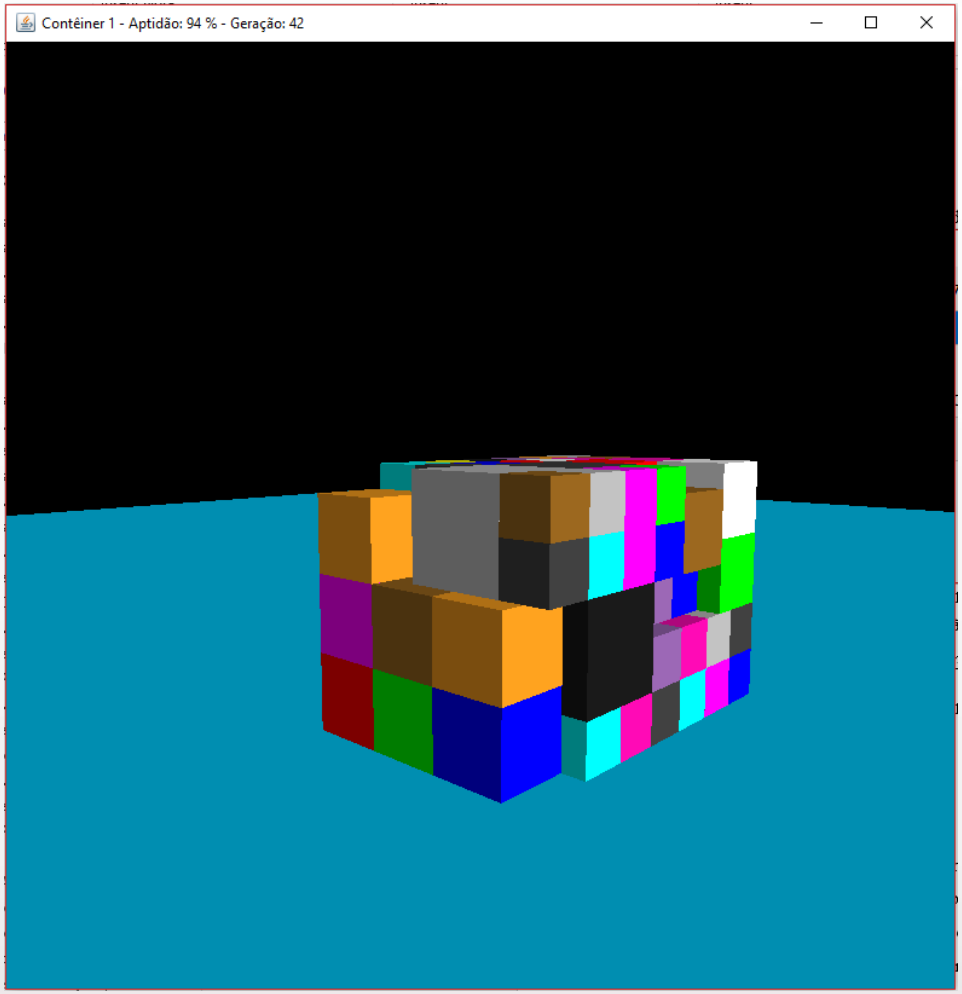
Operacionalidade da Implementação



Operacionalidade da Implementação



Operacionalidade da Implementação



Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

Resultados e Discussões

- Testes realizados em 5 classes diferentes:
  - Classe 1: 70% dos itens altos e profundos;
  - Classe 2: 70% dos itens largos e profundos;
  - Classe 3: 70% dos itens altos e largos;
  - Classe 4: 70% dos itens possuem grandes dimensões;
  - Classe 5: 70% dos itens possuem pequenas dimensões.

Resultados e Discussões

- Para as classes 1, 2 e 3 a média dos melhores tempos por configuração foi de 3,35 minutos;
- A média de aptidão do primeiro contêiner das melhores configurações ficou em 76,83%.

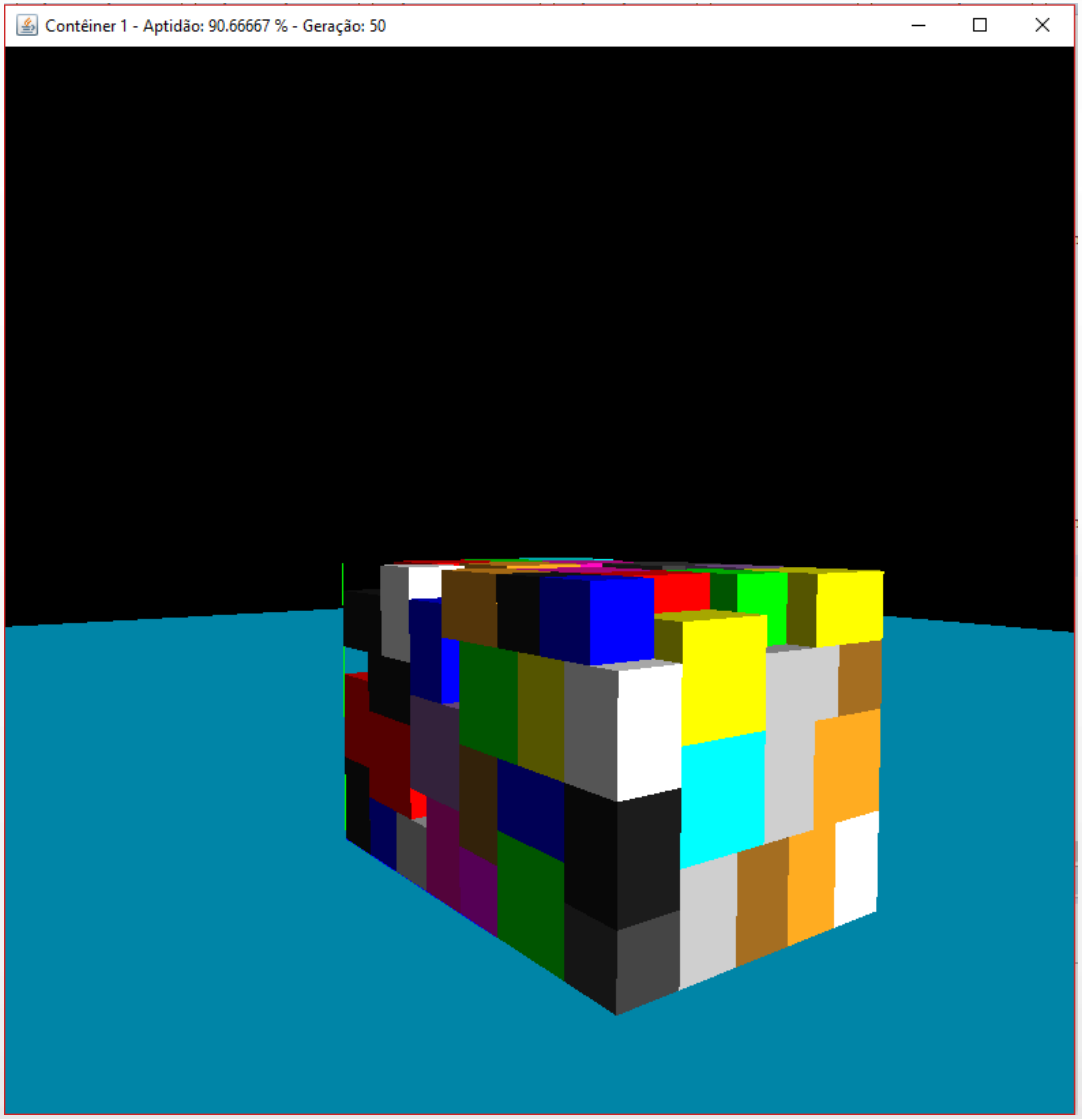
Resultados e Discussões

- Para a classe 4, a média de contêineres gerados nas melhores configurações foi de 11,5;
- O tempo médio de geração nas melhores configurações foi de 5,8 minutos.

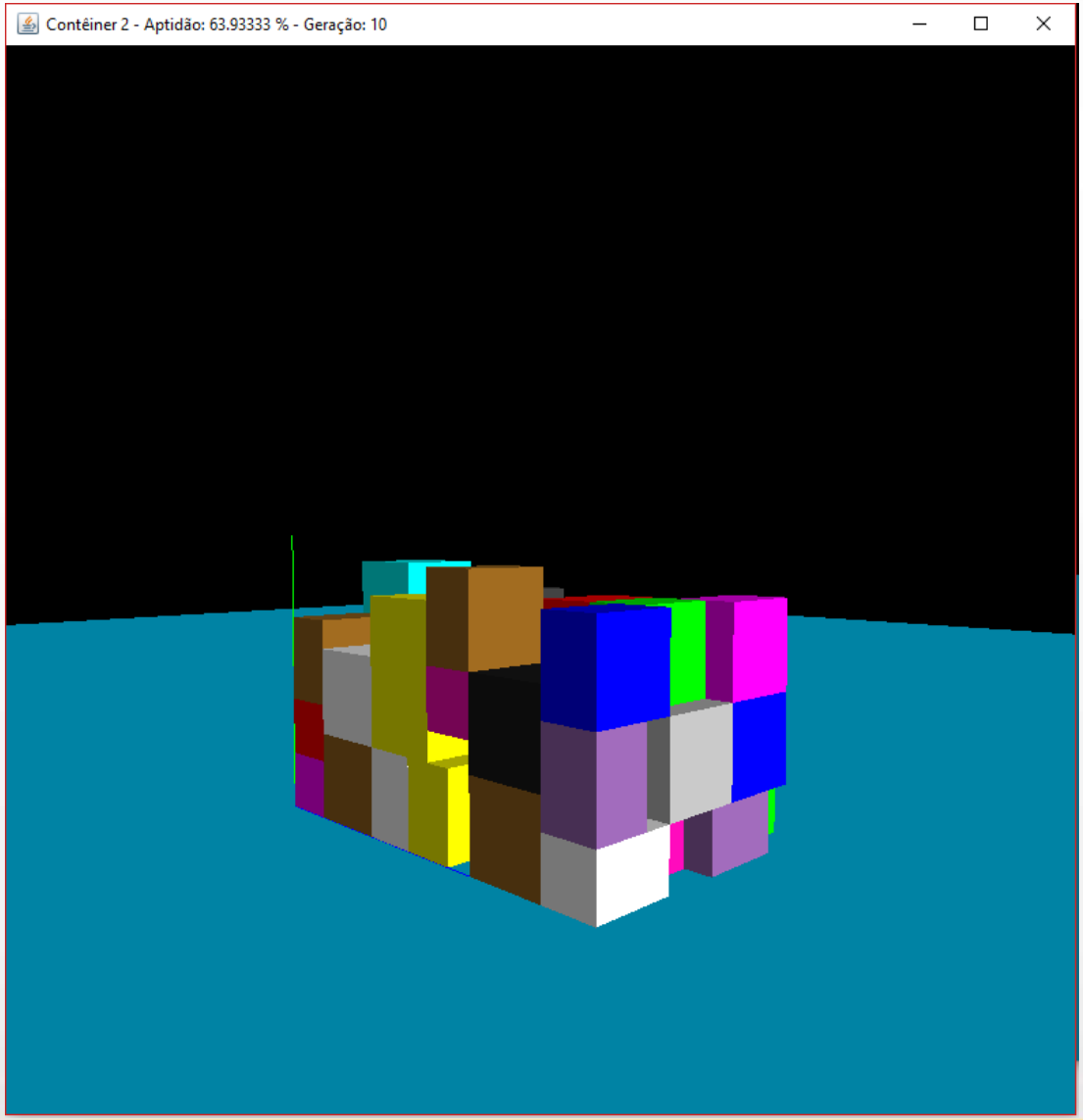
Resultados e Discussões

- Para a classe 5, a média gerações necessárias para atingir o objetivo nas melhores configurações foi de 10,25;
- O tempo médio de geração nas melhores configurações foi de 7,66 segundos.

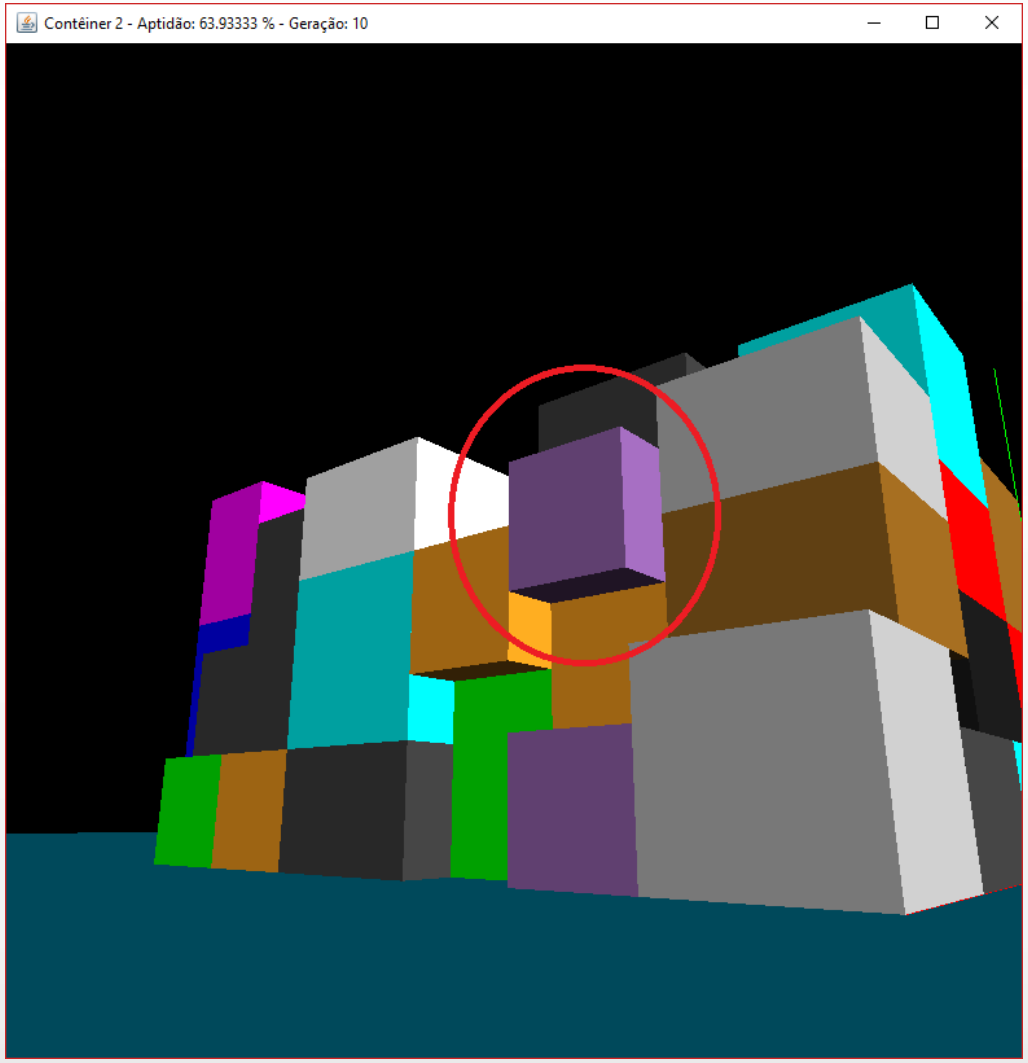
Resultados e Discussões



Resultados e Discussões



Resultados e Discussões



Sumário

- Introdução
- Fundamentação teórica
- Especificação
- Implementação
- Resultados e discussões
- Conclusões
- Demonstração

Conclusões

- Objetivo alcançado;
- Ferramentas se mostraram adequadas;
- Códigos do algoritmo genéticos reaproveitáveis.

Sugestões

- Melhorar visualização 3D;
- Rotação das caixas;
- Suportar outros tipos de contêineres.

FURB
UNIVERSIDADE DE BLUMENAU

Demonstração

FURB
UNIVERSIDADE DE BLUMENAU

Dúvidas?