UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

LIP – SISTEMA PARA OTIMIZAR O CARREGAMENTO DE PRODUTOS PALETIZADOS EM CAMINHÕES

DANIEL GIELOW JUNIOR

BLUMENAU
2017

DANIEL GIELOW JUNIOR

LIP – SISTEMA PARA OTIMIZAR O CARREGAMENTO DE PRODUTOS PALETIZADOS EM CAMINHÕES

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Daniel Theisges dos Santos, Mestre - Orientador

BLUMENAU
2017

LIP – SISTEMA PARA OTIMIZAR O CARREGAMENTO DE PRODUTOS PALETIZADOS EM CAMINHÕES

Por

DANIEL GIELOW JUNIOR

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Daniel Theisges dos Santos, Mestre – Orientador, FURB
Membro:	Profª. Andreza Sartori, Doutora – FURB
Membro:	Prof. Roberto Heinzle, Doutor – FURB

Blumenau, 03 de julho de 2017

Dedico este trabalho a minha mãe que sempre me incentivou nos estudos e sonhou com a conclusão do meu curso de graduação.

AGRADECIMENTOS

Ao meus pais e irmãos, pelo incentivo para que eu chegasse até esta etapa da minha vida.

A minha namorada Carolina, que de forma especial me deu força e apoio nos momentos mais difíceis.

Ao meu orientador Daniel Theisges, pela dedicação e incentivo que tornaram possível a conclusão deste trabalho.

E aos meus amigos e colegas de curso Camila, Filipe, Jean, Juliano, Luis e Philip que tornaram esses anos de estudos mais agradáveis.

A única coisa que importa é colocar em prática, com sinceridade e seriedade, aquilo em que se acredita.

Dalai Lama

RESUMO

Problemas de empacotamento possuem uma relação forte com a área de logística, quando trata da alocação de um conjunto de itens em um recipiente, considerando variáveis como a otimização de espaço e ordem de entrega. Este trabalho apresenta o desenvolvimento de uma aplicação para auxiliar no processo de carregamento de produtos paletizados em caminhões. A aplicação, denominada LIP, realiza o processamento dos pedidos importados pelo usuário, e disponibiliza o resultado de como estes produtos devem ser dispostos dentro do caminhão, respeitando algumas restrições para o transporte. Essas restrições são: ordem de entrega, balanceamento dos produtos dentro do caminhão e o volume máximo permitido. Para o desenvolvimento da aplicação, foram utilizados métodos de aproximação utilizando técnicas de inteligência artificial. Destaca-se também o uso da biblioteca three.js para a visualização 3D do resultado alcançado. Foram feitos experimentos no intuito de validar o funcionamento e a eficiência do sistema. Os resultados obtidos demonstram que a aplicação desenvolvida é capaz de encontrar bons resultados, e realizar o processamento de cenários reais de forma satisfatória.

Palavras-chave: Problema de carregamento de paletes. Algoritmos genéticos. Inteligência artificial. Algoritmos aproximados. Busca local. A*.

ABSTRACT

Packaging problems have a strong relationship with the logistics area, when dealing with the allocation of a set of items in a container, considering variables such as space optimization and delivery order. This work presents the development of an application to assist the loading process of palletized products in trucks. The application, denominated LIP, performs the processing of orders imported by the user, and provides the result of how these products must be arranged inside the truck, respecting some restrictions for transportation. These restrictions are: delivery order, balancing of the products inside the truck and the maximum volume allowed. For the development of the application, approximation methods were used using techniques of artificial intelligence. Also noteworthy is the use of the three.js library for the 3D visualization of the result reached. Experiments were carried out in order to validate the functioning and efficiency of the system. The obtained results demonstrate that the developed application is able to find out good results, although with some limitations, and perform the real scenario processing in a satisfactory way.

Key-words: Pallet loading problem. Genetic algorithm. Artificial intelligence. Approximation algorithms. Local search. A*.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG – Algoritmos Genéticos

API – Application Programming Interface

BPP – Bin Packing Problem

CLP – Container Loading Problem

DPLP – Distributor's (Single) Pallet Loading Problem

EA – Enterprise Architect

HTML – HyperText Markup Language

IA – Inteligência Artificial

IIPP – Identical Item Packing Problem

JSON – JavaScript Object Notation

MPLP – Manufacturer's Pallet Loading (Packing) Problem

PLP – Pallet Load Problem

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

SCPP – Single Container Packing Problem

SLOPP – Single Large Object Placement Problem

SUMÁRIO

1 INTRODUÇÃO

A inteligência artificial (IA) é um assunto estudado a décadas. Seu termo nasceu oficialmente em 1956 no famoso encontro de Dartmouth e ao longo do tempo o estudo da IA proporcionou grande progresso na resolução de problemas e diversos métodos foram desenvolvidos (RUSSELL; NORVIG, 2013). Atualmente a IA possui uma série de áreas e técnicas de atuação, como por exemplo os algoritmos de busca (LIMA et al., 2008).

Algoritmos de Busca são técnicas de IA aplicadas a problemas de alta complexidade teórica que não são resolvidos com técnicas de programação convencionais, principalmente as de natureza puramente numérica (DIRENE, 2016). Segundo Zambiasi (2010), os algoritmos de busca podem ser classificados em: busca cega e busca heurística.

Dentro dos algoritmos de busca heurística destacam-se os Algoritmos Genéticos (AG). Propostos inicialmente por John Holland na década de 70, se expandiram por toda a comunidade científica por oferecer boas soluções em problemas extremamente complexos (LINDEN, 2006), tais como os problemas NP-difíceis. Problemas NP-difíceis são problemas com complexidade exponencial, ou seja, o esforço para encontrar sua solução cresce de forma exponencial com o tamanho do problema (ZIVIANI, 2006). Dentro da área da logística pode-se encontrar vários problemas NP-difíceis, como por exemplo, o clássico problema do caixeiro viajante, que consiste em encontrar a menor rota a ser percorrida com diversos pontos de passagem (SILVEIRA, 2001).

Outros problemas NP-difícil relacionados com a área de logística são os problemas de empacotamento, que consistem em preencher um recipiente com a maior quantidade de itens possíveis (WÄSCHER et al., 2007). Exemplo desse tipo de problema é o carregamento de paletes (*Pallet Loading Problem* - PLP), que consiste em organizar as caixas de produtos sobre paletes, visando otimizar a ocupação de área disponível, reduzindo então custos em operações em transporte e armazenagem (CAVALCANTI JÚNIOR, 2009). Esta atividade faz parte do processamento de pedidos e tem o seu custo considerado pequeno quando comparados aos demais processos logísticos. Contudo, este processo é considerado uma atividade primária por ser um elemento crítico em termos de tempo necessário para levar os produtos até o cliente (BALLOU, 2006).

Diante deste contexto, foi desenvolvido uma aplicação para a otimização do carregamento de produtos paletizados em caminhões utilizando algoritmos aproximados como técnica de busca, possibilitando a visualização 3D do resultado encontrado.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver uma aplicação para otimizar o carregamento de produtos paletizados em caminhões e disponibilizar a visualização 3D do resultado.

Os objetivos específicos são:

a) desenvolver um AG para otimizar o carregamento de produtos paletizados em caminhões, tratando restrições de ordem de entrega, volume e balanceamento da carga;
b) desenvolver um algoritmo para calcular a posição 3D dos produtos dentro do palete, para tratar o problema de empacotamento;
c) desenvolver uma aplicação web para configurar parâmetros relacionados ao caminhão, paletes, caixas e execução dos algoritmos;
d) desenvolver uma aplicação para visualização 3D do resultados do algoritmos.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo apresenta a introdução e os objetivos do trabalho. O segundo capítulo apresenta a fundamentação teórica sobre a área da logística, problema de paletes e algoritmos aproximados, além de trabalhos similares ao LIP. O terceiro capítulo mostra o desenvolvimento do trabalho com requisitos, especificação, implementação, operacionalidade do sistema e os resultados obtidos. Por fim, o quarto capítulo relata as conclusões e extensões que poderiam ser desenvolvidas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos relacionados ao trabalho. A seção 2.1 expõe uma visão geral sobre a área da logística e o problema de empacotamento. A seção 2.2 dedica-se a explanar o problema de carregamento de paletes. A seção 2.3 oferece uma visão sobre algoritmos de aproximação e justifica a sua importância para este trabalho. Por fim, a seção 2.4 apresenta os trabalhos correlatos.

2.1 LOGÍSTICA

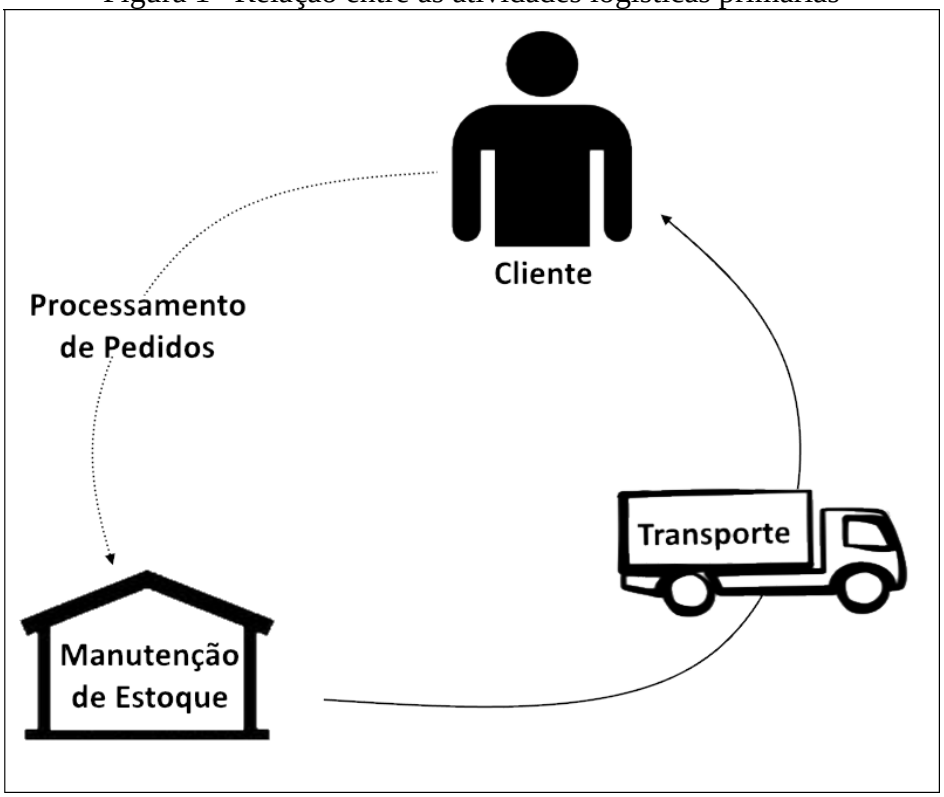
A utilização da logística existe desde o início da civilização, a sua origem e desenvolvimento está diretamente ligada às atividades militares e necessidades resultantes das guerras (MARQUES, 2008). A logística relacionada a estratégia militar caracterizava-se pela movimentação e coordenação das tropas, armamentos e munições para os locais necessários (COELIS, 2008).

Todo este processo logístico foi construído com o objetivo de abastecer, transportar e alojar tropas, proporcionando os recursos necessários no local certo e na hora certa (COELIS, 2008). Atualmente o conceito da logística foi expandido para ser aplicado à gestão empresarial (COELIS, 2008).

Bowersox e Closs (2001) afirmam que a logística empresarial se tornou singular e está presente em qualquer atividade de produção ou de marketing. A maioria dos consumidores em nações industriais altamente desenvolvidas, quando vão as lojas, esperam encontrar os produtos recém-fabricados disponíveis (BOWERSOX; CLOSS, 2001). Diante deste cenário, a implementação das melhores práticas logísticas tornou-se uma das áreas operacionais mais desafiadoras e interessantes da administração (BOWERSOX; CLOSS, 2001).

Basicamente, como pode ser visto na Figura 1, existem três atividades primárias que caracterizam a logística: transportes, manutenção de estoques e processamento de pedidos. “Essas atividades são consideradas primárias porque ou elas contribuem com maior parcela do custo total da logística ou elas são essenciais para a coordenação e o cumprimento da tarefa logística” (BALLOU, 1993, p.24).

Figura 1– Relação entre as atividades logísticas primárias



Fonte: adaptado de Ballou (2006).

Nas palavras de Ballou (2006), o transporte é essencial, pois nenhuma empresa moderna consegue operar sem providenciar a movimentação da sua matéria-prima ou de seus produtos acabados. O transporte também representa um dos elementos mais importantes em termos de custos logísticos para as empresas, visto que a movimentação de cargas pode absorver de um a dois terços dos custos logísticos totais, além de impactar diretamente no atendimento dos pedidos dos clientes.

A manutenção de estoque, assim como o transporte é de extrema importância, pois é destinado a gerenciar os recursos designados a atender futuras demandas (MENDONÇA, 2013). Enquanto o transporte é responsável por garantir o produto no lugar disponível para o cliente, os estoques são responsáveis por diminuir o tempo do transporte, agindo como amortecedores entre a oferta e a demanda (MENDONÇA, 2013).

O processamento de pedidos tem o seu custo considerado pequeno quando comparados aos custos do transporte ou de manutenção de estoque. Contudo, este processo é considerado uma atividade primária por ser um elemento crítico em termos de tempo necessário para levar os produtos até o cliente (BALLOU, 2006).

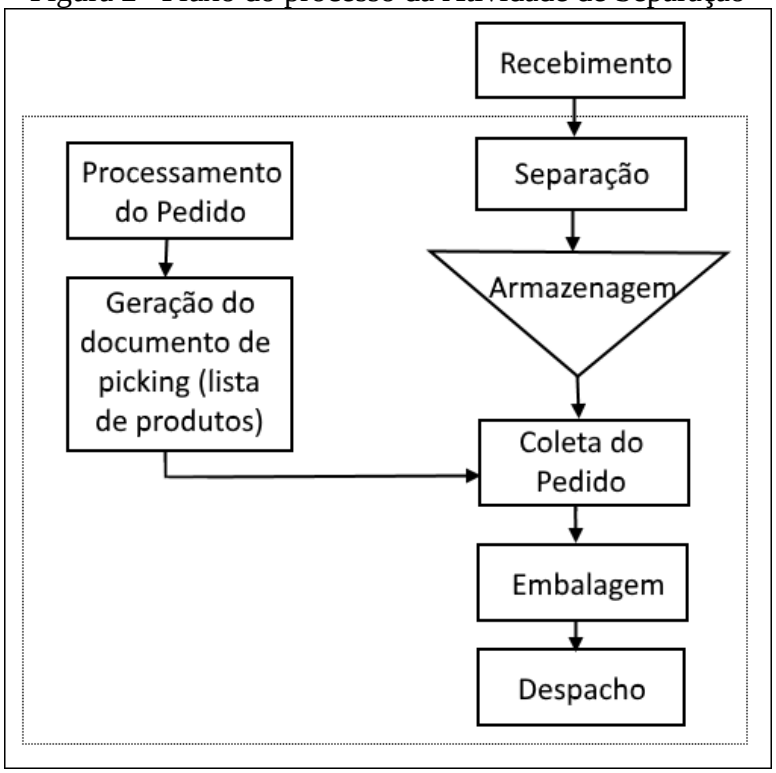
Dentro da atividade do processamento de pedidos destaca-se o processo de separação dos produtos. Este processo consiste em agrupar materiais, peças e produtos em função dos pedidos de clientes (BOWERSOX; CLOSS, 2001, p. 350).

2.1.1 PROCESSO DE SEPARAÇÃO

O processo de separação, também conhecido como *picking* consiste em coletar produtos na área de armazenagem e disponibiliza-los na área de expedição (MOURA, 1998). Ainda segundo Moura (1998) o objetivo desta atividade é fornecer os pedidos aos consumidores, na quantidade correta, sem avarias e no prazo determinado.

O processo de separação é considerado como uma das atividades mais importantes dentro de um armazém, pois dependendo do armazém pode ser responsável de 30% a 40% do custo de mão-de-obra (LEITÃO, 2007). Em conjunto com o custo de execução, o tempo dessa atividade tem uma influência considerável no tempo de ciclo do pedido, ou seja, o tempo entre o recebimento de um pedido do cliente até a sua entrega (LEITÃO, 2007). De uma forma simplificada, a Figura 2 apresenta como o processo de separação está inserido dentro das atividades do armazém:

Figura 2– Fluxo do processo da Atividade de Separação



Fonte: adaptado de Leitão (2007).

Embora possa parecer um processo simples, tem sido um grande desafio para as empresas, pois trata-se de uma atividade cara e que apresenta muitos problemas (ESTEVES; ALVES, 2013). De acordo com Esteves e Alves (2013), as falhas durante o processo de separação, acabam se desencadeando para as etapas seguintes da cadeia de abastecimento, causado grandes prejuízos financeiros para as empresas.

Com o aumento das exigências dos consumidores e a competição entre as empresas no mercado, tornou-se necessários a adoção de sistemas para flexibilizar a atividade de separação, a fim de suportar os níveis de serviço e qualidade necessária (LEITÃO, 2007).

Segundo Alegre (2005), a tarefa de separação começa com um processo de paletização que converte os pedidos dos clientes em uma lista de separação, contendo o produto, quantidade e a sequência da coleta. O responsável pela separação, realiza a coleta dos itens no armazém e os transporta até o local de distribuição.

2.1.2 PROBLEMA DE EMPACOTAMENTO

No problema de empacotamento, também conhecido como *bin packing problem* (BPP), há uma lista de vários pequenos objetos, os quais são chamados de itens, e vários objetos maiores os quais são chamados de recipientes. A intenção é alocar todos os itens dentro dos recipientes, de forma a minimizar o número necessário de recipientes a serem utilizados (OLIVEIRA; WÄSCHER, 2007). Este problema é aplicável em diversas situações práticas no dia-a-dia, assim como na indústria, onde uma combinação ideal dos itens pode levar a uma diminuição considerável de custos (PEDRUZZI et al., 2016).

De acordo com Silveira (2011), o problema de empacotamento é classificado como um problema combinatório NP-difícil. Problemas como estes são inviáveis de serem resolvidos de forma exata e eficiente. Silveira (2011) para estes problemas são utilizados métodos como métodos heurísticos e algoritmos aproximados, que trocam a solução ótima pela garantia de uma solução aproximada e computável.

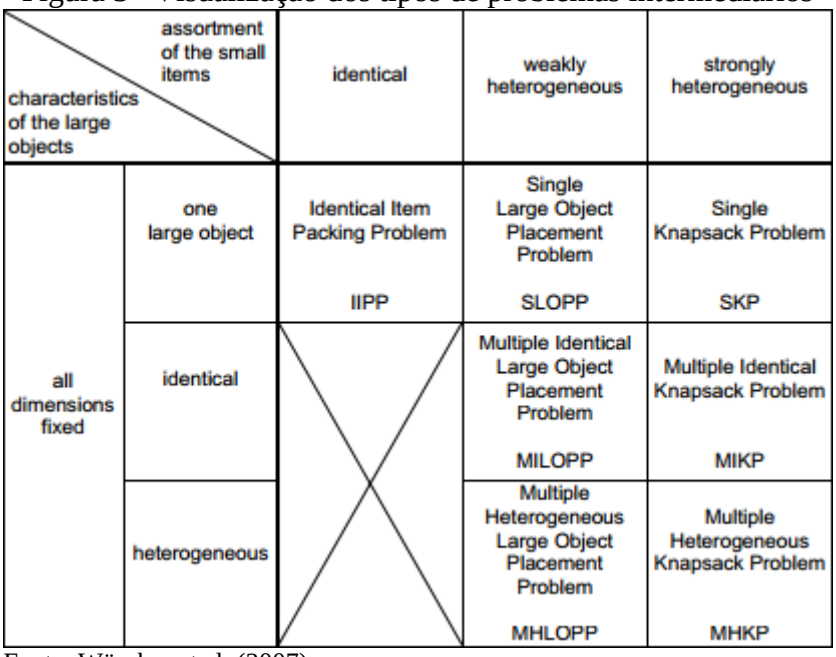
Bischoff e Marriott (1990) realizaram levantamento bibliográfico trazendo algumas técnicas de aproximação existentes para o problema de empacotamento. Estas abordagens podem não obter a solução ótima, mas garantem um limite superior próximo da solução ideal:

a) *First Fit*: Tenta adicionar o item no primeiro recipiente que pode acomoda-lo
b) *Next Fit*: Tenta primeiro ajustar o item no recipiente atual, se malsucedido, ele atribui um novo recipiente para o item ser colocado.
c) *Best Fit*: Esta técnica tenta adicionar o item no recipiente que terá a capacidade mínima restante, se o objeto puder ser alocado.
d) *Worst Fit*: Ao contrário do *Best Fit* tenta adicionar o item no recipiente que terá a maior capacidade restante.

Uma classificação para vários tipos de problemas de empacotamento foi apresentada por Wäscher et al. (2007), onde os tipos de problemas de empacotamento são caracterizados pela variação do número dos recipientes utilizados, por suas dimensões, e pelas características

dos itens envolvidos, desde itens com as mesmas dimensões até itens fortemente heterogêneos (Figura 3).

Figura 3– Visualização dos tipos de problemas intermediários



Fonte: Wäscher et al. (2007).

Para o problema de empacotamento, este trabalho foca no Problema de empacotamento em um único container (*Single Large Object Placement Problem - SLOPP*) e no Problema de empacotamento de itens idênticos (*Identical Item Packing Problem – IIPP*), visto que estes tipos estão relacionados com o PLP (WÄSCHER et al., 2007).

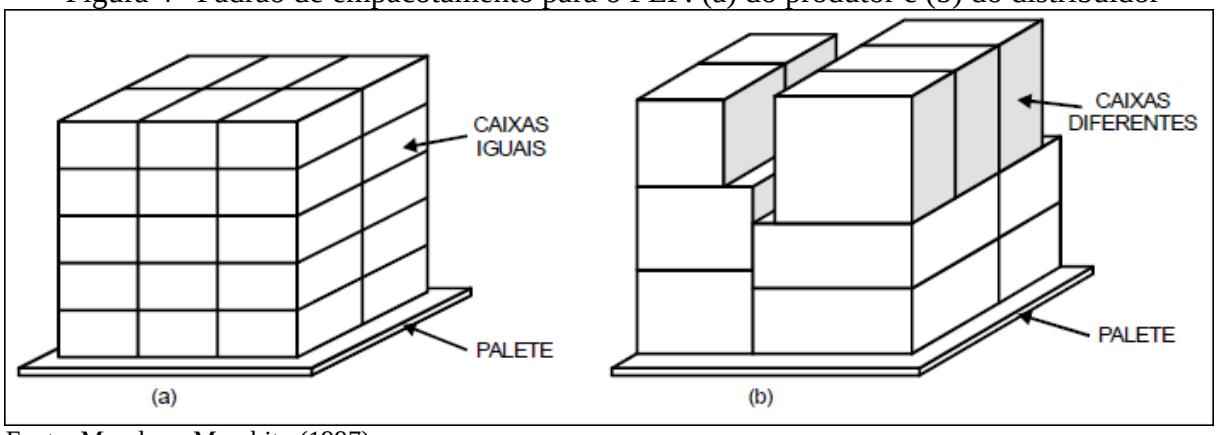
2.2 PROBLEMA DE CARREGAMENTO DE PALETES

O PLP "consiste em organizar as caixas de produtos sobre o palete visando otimizar a ocupação da área disponível.[...]. Em muitos casos estes produtos são colocados de forma que não utilizam o espaço disponível no palete. Isso gera maior custo em operações de transporte e armazenagem" (CAVALCANTI JÚNIOR, 2009, p. 5).

Ceccilio e Morabito (2004) afirmam que o PLP é considerado NP-difíceis. A solução para este problema, quando utilizados métodos exatos, não são encontradas dentro de um limite de tempo razoável. Ceccilio e Morabito (2004) ainda comentam que a necessidade de uma resposta imediata, faz com que métodos aproximados sejam utilizados na maioria das abordagens propostas.

Para Morales e Morabito (1997) existem dois problemas associados ao PLP: o problema de carregamento de paletes do produtor (Figura 4a), onde os paletes são carregados somente com caixas iguais; e o problema de carregamento de paletes do distribuidor (Figura 4b), onde existem caixas de diferentes dimensões.

Figura 4– Padrão de empacotamento para o PLP: (a) do produtor e (b) do distribuidor



Fonte: Morales e Morabito (1997).

A seguir, são apresentados os conceitos apresentadores por Morales e Morabito (1997) diante a tipologia proposta por Wäscher et al. (2007).

2.2.1 Problema de Carregamento de Paletes do Produtor

Para Wäscher et al., (2007), o IIPP consiste em atribuir o maior número possível de itens pequenos idênticos em um único objeto maior. Devido ao fato dos itens serem idênticos é aceito que os itens são carregados em camadas e que todas as caixas têm a mesma orientação vertical.

Wäscher et al., (2007) explicam que diante dessa certeza o problema é reduzido a um bidimensional, uma clássica representação deste problema é o Problema de Carregamento de Paletes do Produtor (*Manufacturer's Pallet Loading (Packing) Problem* - MPLP). No MPLP, o produto fabricado é embalado em caixas de iguais dimensões (l, w, h) onde l representa o comprimento, w representa a largura e h a altura da caixa. Estas caixas devem ser carregadas em camadas horizontais sobre paletes de dimensões (L, W, H) onde L e W são o comprimento e a largura do palete, respectivamente, e H é a altura máxima permitida no palete (MORALES; MORABITO, 1997).

A Figura 5 apresenta um veículo sendo carregado com o MPLP, pode ser observado que é carregado o mesmo item em todas as baias do caminhão e o caminhão possui uma grande capacidade carregamento.

Figura 5– Veículo de carregamento do produtor

Fonte: elaborado pelo autor.

2.2.2 Problema de Carregamento de Paletes do Distribuidor

Para o problema conhecido como SLOPP, de acordo com Wäscher et al. (2007), na sua versão bidimensional é considerado um conjunto de itens fracamente heterogêneos, que serão adicionados em um recipiente maior. O número de vezes que um determinado item será inserido pode ser limitado, e o objetivo do problema é maximizar o número de itens empacotados.

A versão tridimensional deste problema pode ser encontrada na literatura como *single container packing problem* (SCPP). Este problema exige o carregamento de um grande número de caixas fracamente heterogêneas dentro de um recipiente, de modo que o volume das caixas empacotadas seja maximizado (WÄSCHER et al., 2007).

O SCPP também pode ser atribuído ao Problema de Carregamento de Paletes do Distribuidor (*Distributor's (Single) Pallet Loading Problem* - DPLP). Para Xavier (2012), no DPLP o carregamento é realizado com diversas mercadorias com diferentes tamanhos e formas.

Por ser um problema voltado para a visão do distribuidor, para o DPLP outra questão que surge é o fato de que o distribuidor possui mercadorias de diversos clientes. Sendo assim, restrições adicionais como ordem de entrega, estabilidade, condições de carregamento e o peso devem ser levadas em consideração (XAVIER, 2012).

Diferente do MPLP, onde todas as caixas são iguais, não pode ser aplicado o carregamento em camadas, uma vez que os itens possuem dimensões diferentes (WÄSCHER et al., 2007).

A Figura 6 apresenta um caminhão já carregado com o DPLP, podemos observar que este caminhão possui uma pequena capacidade de paletes e que em um único palete há vários itens de tamanhos diferentes.

Figura 6– Veículo de carregamento do distribuidor

Fonte: elaborado pelo autor.

2.3 ALGORITMOS DE APROXIMAÇÃO

Assim como o problema de empacotamento, existem vários problemas NP-Completos importantes para simplesmente serem deixados de lado por ser inviável encontrar uma solução ótima (CORMEN, 2009). Quando um problema é NP-Completo, existem poucas chances de um algoritmo encontrar a solução exata em tempo polinomial, porém, ainda é possível encontrar uma solução aproximada. Quando um algoritmo é capaz de retornar soluções aproximadas este é denominado um algoritmo de aproximação (CORMEN, 2009).

O uso de um algoritmo aproximado dá-se quando há um maior interesse na execução em tempo polinomial do algoritmo do que a necessidade de obter uma reposta ótima para um determinado problema (BORDINI, 2016). Estes algoritmos se beneficiam das características do problema que estão tentando resolver para conseguir encontrar uma solução próxima da ótima. Deste modo, os algoritmos são referidos como dependentes do problema (KANN, 1992).

Segundo Bracht (2004), todo algoritmo de aproximação possui um fator de aproximação, que nos diz quão longe a solução encontrada está em relação a solução ótima. Ainda com o autor o fator de aproximação pode ser definido da seguinte forma:

> Dado um algoritmo A para um problema de otimização, se I é uma instância para este problema, denotamos por A(I) o valor da solução devolvida pelo algoritmo A aplicado à instância I e denotamos por OPT(I) o correspondente valor para uma solução ótima de I. Para problemas de minimização (maximização), diremos que um algoritmo A tem um fator de aproximação α, ou é α-aproximado, se $A(I)/OPT(I) \leq \alpha$, com $\alpha > 1$ ($A(I)/OPT(I) \geq \alpha$, com $\alpha < 1$), para toda instância I (BRACHT, 2004, p. 5).

Ao realizar um algoritmo aproximado, é necessário provar que o fator de aproximação é justo. Para isso deve-se obter uma instância na qual a solução encontrada pelo algoritmo e sua solução ótima sejam idênticas, ou tão bom quanto se queira (XAVIER, 2003). Segundo Bracht (2004) podemos definir um fator de aproximação da seguinte forma:

> Dado um algoritmo A, para um problema de minimização (maximização), com fator de aproximação α, dizemos que α é um fator de aproximação justo se existir, para qualquer $\varrho \geq 0$, pelo menos uma instância I, tal que $A(I)/OPT(I) \leq \alpha - \varrho$ ($A(I)/OPT(I) \geq \alpha + \varrho$) (BRACHT, 2004, p. 5).

De acordo com Bracht (2004), existem várias técnicas para o desenvolvimento de algoritmos de aproximação. Como por exemplo arredondamento de soluções via programação linear, dualidade em programação linear e método primal-dual, algoritmos probabilísticos e sua desaleatorização, programação semidefinida.

Segundo Fingler (2013), quando um algoritmo de aproximação não existe ou a qualidade da solução não é suficiente e os métodos exatos não são viáveis, um dos métodos heurísticos pode ser utilizado e obter resultados satisfatórios. Entre estes métodos, os mais comuns são as meta-heurísticas, algoritmos independentes do problema que iterativamente buscam soluções seguindo um comportamento.

2.3.1 BUSCA HEURÍSTICA

Também conhecidos como busca da melhor escolha, a busca heurística é uma instância do algoritmo geral da busca em árvore em que um nó é expandido com base em uma função de avaliação (CORMEN, 2009). Tal método normalmente é um algoritmo iterativo em que cada iteração conduz a busca por uma nova solução que, eventualmente, poderá ser melhor que a melhor solução encontrada previamente.

Russell e Norvig (2013) citam que uma implementação conhecida de busca heurística é a busca gulosa. Este algoritmo visa expandir o nó que está mais próximo do objetivo, com o fundamento de que isso pode conduzir a uma solução rapidamente. Ou seja, ele sempre faz a escolha que parece ser a melhor no momento, com a expectativa que essa escolha conduza a

uma solução globalmente ideal. Assim, este algoritmo avalia os nós usando apenas a função heurística: f(n) – h(n).

Esta estratégia é perigosa, uma vez que o nó aparentemente perto da solução pode levar a um beco sem saída. Uma solução para isso seria considerar os nós visitados anteriormente (LUGER, 2004).

Segundo Russell e Norvig (2013), o algoritmo A* baseia-se nesta ideia, o algoritmo avalia os nós da busca, combinando g(n), que representa o custo para alcançar o nó, e h(n), que representa o custo de ir do nó até o objetivo: f(n) = g(n) + h(n). Desta forma, temos em f(n) o custo estimado da solução mais barata através de *n*.

Seguindo este princípio, o algoritmo A*, quando se está tentando encontrar a solução de menor custo, tenta primeiro o nó com menor valor de g(n) + h(n). Segundo Russell e Norvig (2013), este algoritmo é mais razoável, pois, desde que a função heurística f(n) satisfaça certas condições o algoritmo A* é completo e ótimo.

Contudo, Russell e Norvig (2013) afirmam que a complexidade de A* muitas vezes torna impraticável insistir em encontrar uma solução ótima. Variantes desta abordagem encontram rapidamente soluções subótimas. Estes algoritmos fazem uma sequência de decisões que otimizam alguma escolha local, sem garantir que possam conduzir a melhor solução global. A solução é construída passo a passo, onde cada passo da execução do algoritmo a solução é construído através de uma decisão que localmente é a melhor possível (WILLIAMSON; SHMOYS, 2011).

Para Kann (1992) outra forma de aproximação é através de uma busca local, onde cada etapa da execução do algoritmo, tenta-se encontrar soluções melhores a partir a solução atual. Russell e Norvig (2013) afirmam que, dentro da família dos algoritmos de busca local estão os algoritmos genéticos, que inspirados na biologia evolutiva são úteis para resolver problemas de otimização.

2.3.2 ALGORITMOS GENÉTICOS

Os AGs utilizam técnicas heurísticas e pertencem ao grupo dos algoritmos evolucionários que utiliza os princípios de seleção natural e evolução propostos por Darwin (LINDEN, 2006). Inspirados na forma como os seres vivos sobrevivem e passam seu material genético para as próximas gerações, tem como propósito encontrar soluções aproximadas em problemas de otimização e busca (LINDEN, 2006).

Dentro dos conceitos da teoria da evolução proposta por Darwin e dos mecanismos da genética, surge a teoria dos AG. Este algoritmo foi inventado por John Holland na década de

70 (LINDEN, 2006). Holland estudou a evolução das espécies e propôs um método heurístico que, quando implementado poderia oferecer boas soluções para problemas extremamente difíceis e que não podiam ser resolvidos computacionalmente na época (LINDEN, 2006).

Desde então, o AG começou a se expandir por toda a comunidade científica, ajudando a resolver uma série de problemas importantes (LINDEN, 2006). Estes problemas possuem como cenário a inviabilidade de avaliar todas as combinações possíveis, podendo atingir um número altíssimo de estados possíveis, que torna inviável a sua avaliação (ALMEIDA; KAGAN, 2010).

Para implementar um AG, deve-se realizar os seguintes passos (ZINI, 2009):

a) Escolhe-se uma população inicial, normalmente formada por indivíduos criados aleatoriamente;

b) Avalia toda a população através de um cálculo de aptidão seguindo algum critério determinado por uma função que mede a qualidade do indivíduo, desta forma, os melhores indivíduos são aqueles que apresentam um melhor resultado na função de aptidão;

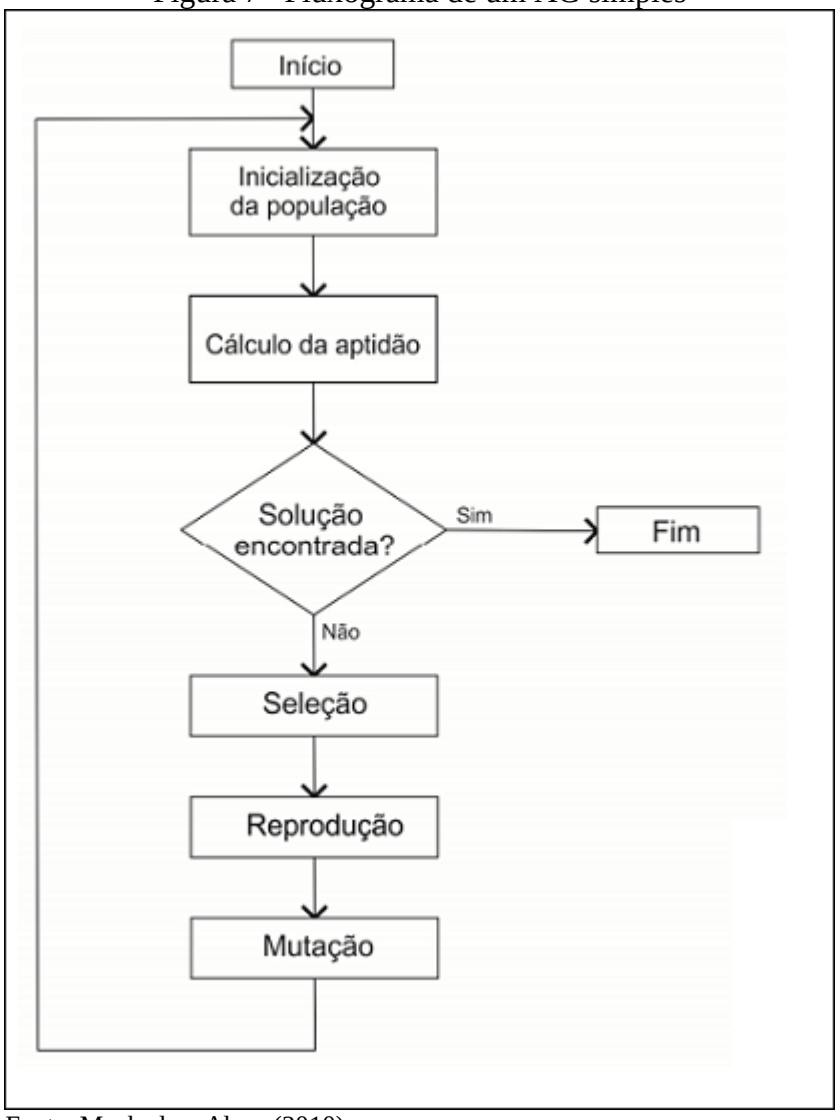
c) Realiza-se uma seleção dos indivíduos com base em uma estratégia pré-estabelecida, para servir como base na criação de uma nova população de indivíduos;

d) Aplica-se então as operações genéticas de reprodução e mutação sobre os indivíduos selecionados, gerando uma nova população.

Os passos acima são repetidos até que um indivíduo de qualidade aceitável seja encontrado, um número preestabelecido de passos seja atingido ou o algoritmo não consiga mais mostrar evolução, ou seja, não se consegue melhorar a solução já encontrada.

A Figura 7 apresenta um fluxograma demonstrando a estrutura básica de um algoritmo genético.

Figura 7– Fluxograma de um AG simples



Fonte: Machado e Alves (2010).

A execução de um AG inicia com a criação de uma população inicial. A população é o conjunto de cromossomos que estão sendo considerados como uma solução e que serão utilizados para criar novos indivíduos para análise. A qualidade da população inicial é um ponto chave para o sucesso do processo evolutivo, ela deve conter uma amostra relevante do espaço da busca, para que através da recombinação seja possível convergir para uma solução (POZO et al., 2002).

O tamanho da população pode também impactar na eficiência do AG, visto que populações muito pequenas tem grandes chances de perder a diversidade necessária para convergir para uma solução. Contudo em populações grandes, o AG poderá perder parte da sua eficiência avaliando a função de aptidão em todos os indivíduos, além de consumir mais recursos computacionais (POZO et al., 2002).

2.3.2.1 AVALIAÇÃO

Segundo Bueno (2009), a função de aptidão é o componente mais importante de qualquer AG, ela é a responsável por avaliar a aptidão de cada indivíduo através de uma determinada função. Nesta etapa é medido quão próximo um indivíduo está da solução desejada ou quão boa é esta solução. É fundamental que esta função diferencie na proporção correta as más soluções das boas, visto que, se houver pouca precisão, uma boa solução pode ser descartada durante o processo de seleção do algoritmo, além de gastar mais tempo explorando soluções pouco promissoras.

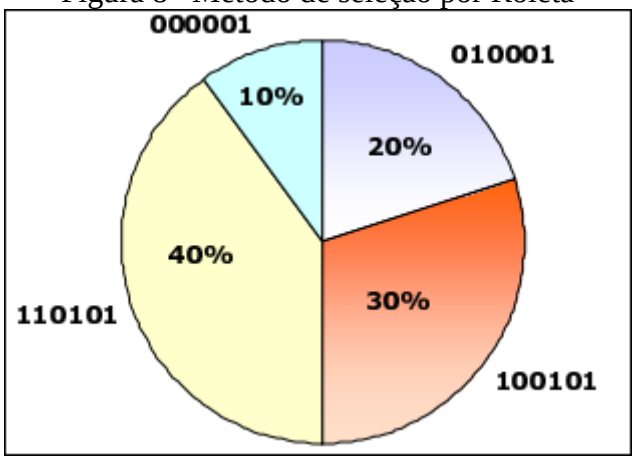
As funções de seleção podem ser utilizadas tanto para escolher os cromossomos que serão aplicados os operadores genéticos, quanto na escolhe dos cromossomos mais adaptados para passar a próxima geração (BUENO, 2009).

2.3.2.2 SELEÇÃO

O operador de seleção simula o processo de seleção natural, onde os pais mais aptos geram mais filhos, ao mesmo tempo que os pais menos aptos podem gerar descendentes (LINDEN, 2007). Desta forma, indivíduos com a função de avaliação mais alta são privilegiados, sem desprezar por completo indivíduos menos aptos, pois estes podem conter características genéticas que sejam favoráveis a criação de um indivíduo que seja a melhor solução (LINDEN, 2007). Para isso diversas formas de seleção foram desenvolvidas, entre elas o método de seleção por Roleta e o método de seleção por Torneio (POZO et al., 2002).

No método de seleção por roleta, quanto melhor for o cromossomo, mais chance ele tem de ser selecionado. Conforme pode ser observado na Figura 8, cada indivíduo é representado proporcionalmente a sua aptidão na roleta, assim, para os indivíduos mais aptos é dado uma porção maior da roleta, enquanto indivíduos com uma aptidão inferior recebem uma porção relativamente menor (POZO et al., 2002).

Figura 8– Método de seleção por Roleta



Fonte: Pozo et al. (2002).

A seleção por torneio é a mais simples de ser implementada, a sua execução resume-se na escolha de pelo menos dois indivíduos aleatoriamente dos quais é escolhido o com maior aptidão. Mesmo assim é bastante utilizado, pois evita a convergência prematura da população, além de utilizar menos recurso computacional (POZO et al., 2002).

2.3.2.3 MUTAÇÃO

No processo de mutação é modificado aleatoriamente alguma característica do cromossomo sobre o qual é aplicada, conforme pode ser visualizado na Figura 9. Esta troca é um importante fenômeno para a diversidade e evolução, e acaba por criar novas características que não existiam ou existiam em pequena quantidade em uma população (POZO et al., 2002).

Figura 9– Exemplo do processo de mutação

Fonte: adaptado de Linden (2007).

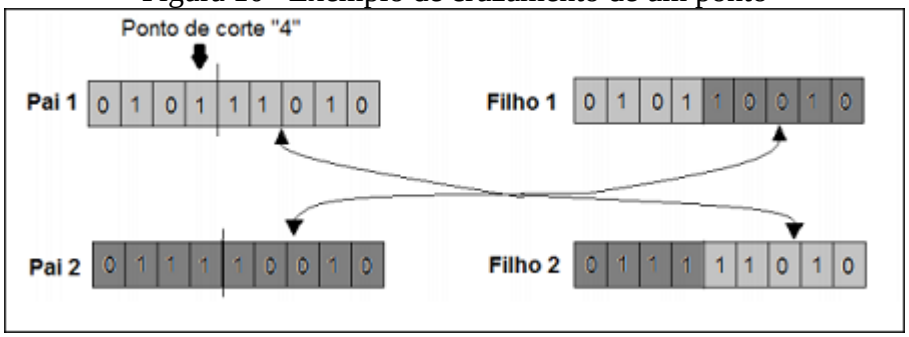
O operador de mutação é aplicado aos indivíduos com uma taxa de mutação geralmente pequena, pois a mutação excessiva em uma população pode gerar várias soluções irregulares, impedindo a evolução da população (POZO et al., 2002). Segundo Linden (2017) taxas de mutações muito altas, fazem com que o AG se comporte de forma estranha, agindo como uma técnica de *random walk*, na qual a solução é determinada de forma aleatória.

2.3.2.4 CRUZAMENTO

O operador de cruzamento é responsável pela reprodução dos cromossomos e troca de genes, garantindo a diversidade de constante evolução da população. Esta etapa basicamente consiste em selecionar pontos de cruzamento dos cromossomos pai, separar estes cromossomos, e trocar as partes destes cromossomos para posteriormente realizar a mutação (BUENO, 2009). Abaixo podemos visualizar algumas formas de cruzamento apresentadas por Linden (2006):

a) um ponto: é escolhido um ponto de corte entre os dois cromossomos, a partir deste ponto as informações genéticas dos pais serão trocadas gerando novos filhos (Figura 10);

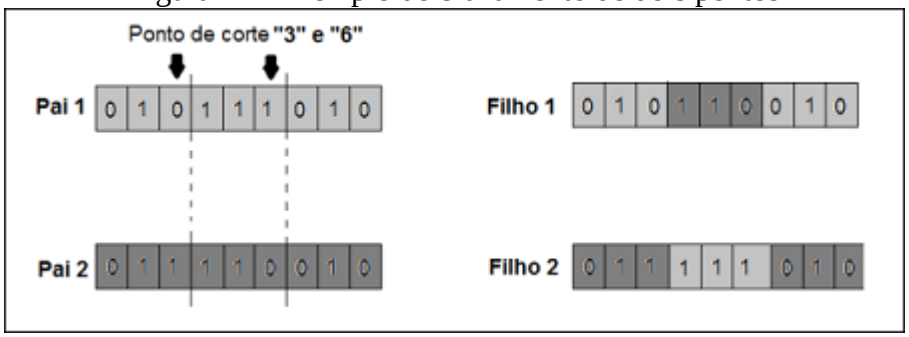
Figura 10– Exemplo de cruzamento de um ponto



Fonte: adaptado de Linden (2007).

b) dois pontos: semelhante ao cruzamento de um ponto, acrescentando apena mais um ponto de corte. Como pode ser visualizado na Figura 11, o primeiro filho será formado pela parte do primeiro pai sem os pontos de corte e pela parte entre os pontos de corte do segundo pai, o segundo filho será formado pelas partes restantes;

Figura 11– Exemplo de cruzamento de dois pontos

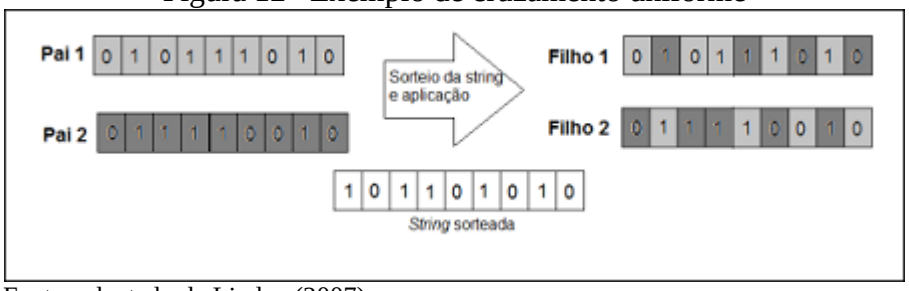


Fonte: adaptado de Linden (2007).

c) Uniforme: realiza uma técnica de sorteio para ser capaz de combinar todo e qualquer esquema existente. Para cada gene é sorteado um número de zero a um. Quando o número for igual a um o primeiro filho recebe o gene da posição do primeiro pai, e o segundo filho recebe o gene da posição do segundo pai. Caso o

número sorteado seja igual a zero, as atribuições serão invertidas. A Figura 12 apresenta um exemplo de cruzamento uniforme para melhor entendimento;

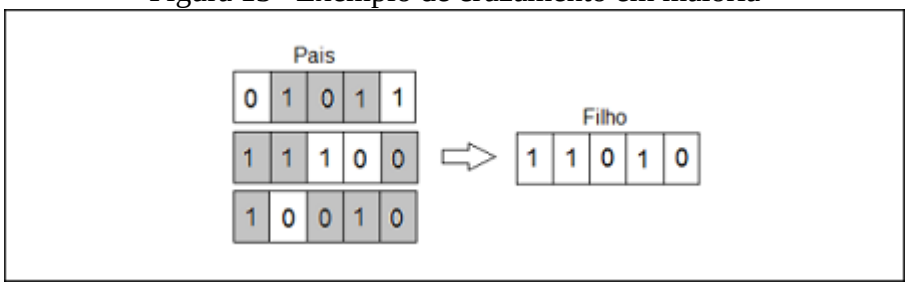
Figura 12– Exemplo de cruzamento uniforme



Fonte: adaptado de Linden (2007).

d) cruzamento baseado em maioria: é uma forma de cruzamento não muito utilizada, pois o AG tende a convergir rapidamente. Esta técnica de cruzamento consiste em sortear vários pais e fazer com cada *bit* do filho seja igual a maioria dos pais, como pode ser observado na Figura 13.

Figura 13– Exemplo de cruzamento em maioria



Fonte: adaptado de Linden (2007).

2.4 TRABALHOS CORRELATOS

Nesta seção serão apresentados três trabalhos correlatos. Na seção 2.4.1 será abordado o artigo de Gonçalves e Resende (2012). A seção 2.4.2 apresentará o trabalho de conclusão de curso de Cavalcanti Júnior (2009) e na seção 2.4.3 será apresentado o artigo de Silva e Soma (2002).

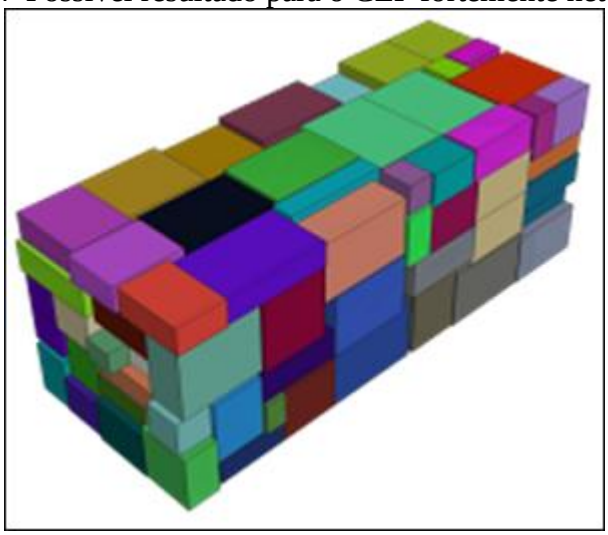
2.4.1 A parallel multi-population biased random-key genetic algorithm for container loading problem

Gonçalves e Resende (2012) apresentam neste trabalho a utilização de um AG para a solução do problema de carregamento de containers (*Container Loading Problem* - CLP), acrescentando também restrições de rotação de estabilidade durante o processo. A técnica utilizada propõe otimizar o carregamento deste um conjunto homogêneo até um conjunto mais heterogêneo de caixas dentro de um container.

Como técnica de otimização utilizou-se o algoritmo genético de chave-randômica, que apresenta uma variação no processo de mutação. Ao invés de utilizar a tradicional mutação gene-por-gene, nesta técnica um pequeno conjunto de novos membros é acrescentado na população. Estes novos indivíduos são gerados a partir da mesma distribuição da população inicial, tendo como objetivo evitar uma convergência prematura da população.

No processo de geração da população destaca-se a utilização da estratégia de multi-população. Neste método são evoluídas várias populações paralelamente. Após um número predeterminado de gerações, o algoritmo seleciona os dois melhores cromossomos a partir da união de todas as populações, para então inseri-los em todas as populações. A Figura 14 mostra uma solução possível para o CLP.

Figura 14–Possível resultado para o CLP fortemente heterogêneo



Fonte: Gonçalves e Resende (2012).

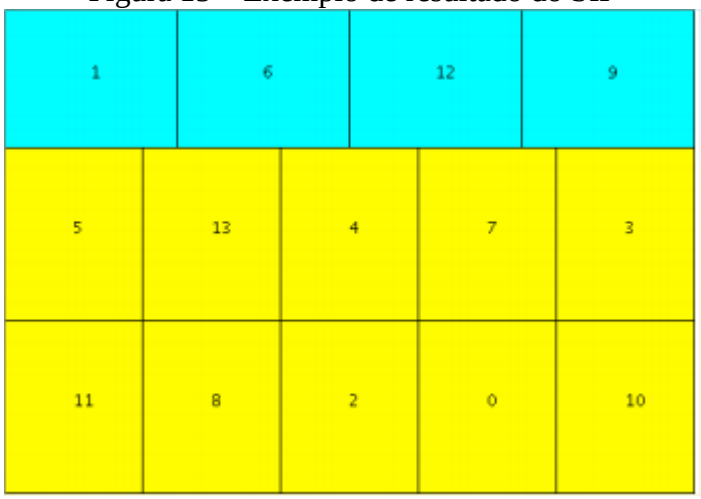
Para análise de resultados, 1500 casos de testes foram executados, desde cargas mais fracas até fortemente heterogêneas. Gonçalves e Resende (2012) compararam os resultados da solução proposta com outras 13 técnicas existentes, constatando que a solução proposta superou as demais em eficiência. Embora a solução também tenha superado as demais em tempo de processamento, esta questão não foi levada em consideração, visto que as execuções dos algoritmos foram realizadas em máquinas com poder de processamento diferentes.

2.4.2 SIP – Sistema inteligente de carregamento de paletes

Cavalcanti Júnior (2009) desenvolveu um sistema para mostrar o posicionamento das caixas sobre um palete, otimizando a sua ocupação. Este problema é conhecido como problema de carregamento de paletes, após um estudo inicial dos algoritmos e técnicas existentes uma abordagem utilizando AGs foi desenvolvida.

No algoritmo desenvolvido por Cavalcanti Júnior (2009) os indivíduos são representados pelo conjunto de caixas sobre o palete. Uma caixa constitui um gene com dois atributos representando as suas coordenadas (x,y) e uma representando a sua orientação (“0” = horizontal e “1” = vertical). A função objetivo do algoritmo é calculada pela quantidade de sobreposição de caixas e o percentual da área do palete ocupada pelas caixas. A Figura 15 apresenta um resultado obtido através do sistema desenvolvido para o PLP, onde estão dispostas 14 caixas, ocupando 100% da área de um palete.

Figura 15 – Exemplo de resultado do SIP



Fonte: Cavalcanti Júnior (2009, p.44).

Inicialmente foram realizados pequenos experimentos para determinar o melhor método de cruzamento para o algoritmo. Após determinado o melhor método de cruzamento foram realizados experimentos utilizando três tamanhos de população (20, 40 e 60 indivíduos), limitados a 2000 gerações. Os resultados foram obtidos através de uma média do resultado de 30 instâncias de testes.

Comparando os resultados o autor concluiu que, apesar de ter um tempo maior de execução o conjunto de testes utilizando uma população de 60 indivíduos convergiu mais rapidamente para a solução ótima. Este também foi o único considerado totalmente válido, pois atingiu 0 de média de sobreposição nas simulações realizadas.

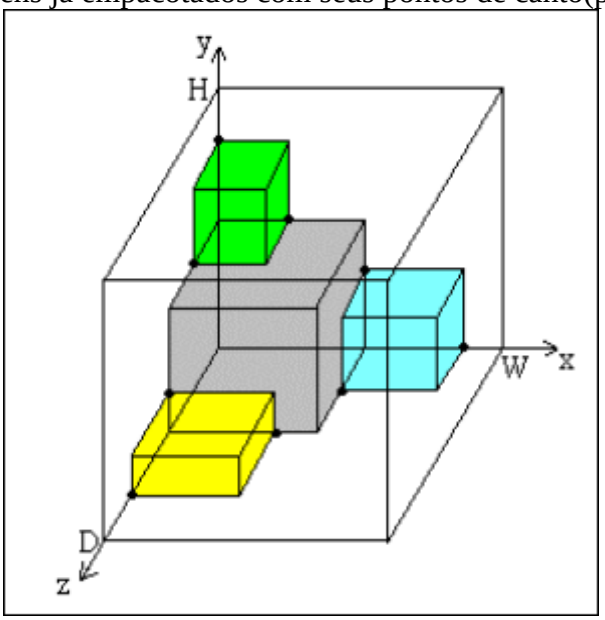
2.4.3 Um algoritmo genético aplicado ao problema de empacotamento de bins tridimensionais

Silva e Soma (2002) desenvolveram um AG para o problema de empacotamento de bins tridimensionais, onde há uma quantidade variada de itens (caixas retangulares) de diferentes tamanhos a serem empacotados ortogonalmente em um número limitados de bins (containers ou caixas retangulares maiores).

A motivação para realização do presente trabalho apontada pelos autores é a quantidade de aplicações nos meios industriais, que variam do carregamento de cargas em aviões ao sequenciamento de tarefas em ambientes multi-processados. O estudo parte então de uma pesquisa sobre heurísticas existentes na literatura, seguindo com a proposta de desenvolvimento de um AG para resolução do 3D-BPP.

O AG desenvolvido faz uma de uma heurística do volume (HV) desenvolvida exclusivamente na resolução do 3D-BPP. A heurística HV leva em consideração o espaço já ocupado dentro de um bin, o espaço não ocupado e os itens que ainda devem ser empacotados. Para o espaço não ocupado, são determinados alguns pontos nos quais os itens podem ser inseridos, 3D-Corner (ponto de canto). Na Figura 16 apresenta um conjunto de itens já alocados e seus respectivos pontos de canto.

Figura 16 – Itens já empacotados com seus pontos de canto(pontos pretos)



Fonte: Silma e Soma (2002).

Os experimentos computacionais para a solução proposta foram realizados em conjunto com outros três algoritmos existentes na literatura em nove cenários diferentes, com a quantidade de itens variando de 10 a 90. Os cenários de testes utilizados interferem nas dimensões dos itens e também na sua aleatoriedade.

O algoritmo desenvolvido pode encontrar soluções bem melhores do que as soluções que foram comparadas. Quanto ao tempo de processamento as soluções comparadas tem um tempo de processamento bem inferior ao algoritmo desenvolvido, duas das soluções existentes demoram de 1 a 10 segundos para encontrar a solução, enquanto a algoritmo desenvolvido levou em média 24,4 e 43,6 segundos.

3 DESENVOLVIMENTO

As seções a seguir descrevem os requisitos, a especificação, a implementação e a operacionalidade do trabalho, abordando sucintamente as ferramentas e etapas utilizadas no desenvolvimento do projeto. Ao fim, são mostrados os resultados obtidos com este trabalho.

3.1 REQUISITOS

A aplicação de otimização de produtos paletizados em caminhões deverá:

a) possuir uma tela para realizar as configurações de execução dos algoritmos de paletização (requisito funcional - RF);
b) possuir um AG para realizar o processamento dos pedidos (RF);
c) possuir um algoritmo A* para realizar o cálculo 3D dos paletes (RF);
d) possuir um algoritmo de busca local para realizar o cálculo 3D dos paletes (RF);
e) possuir uma tela para realizar as configurações necessárias dos produtos que serão calculados pela aplicação (requisito funcional - RF);
f) possuir uma tela para realizar as configurações necessárias dos veículos que serão utilizados pela aplicação (requisito funcional - RF);
g) possuir uma tela para visualizar o resultado do processamento em 3D (RF);
h) possuir uma tela para realizar a importação de novos pedidos no sistema (RF);
i) utilizar a linguagem javascript para o desenvolvimento da aplicação para navegadores web (requisito não-funcional - RNF);
j) utilizar a linguagem C# para o desenvolvimento do AG (RNF);
k) utilizar a linguagem C# para o desenvolvimento do algoritmo de busca local(RNF);
l) utilizar a linguagem Java para o desenvolvimento do A* (RNF).

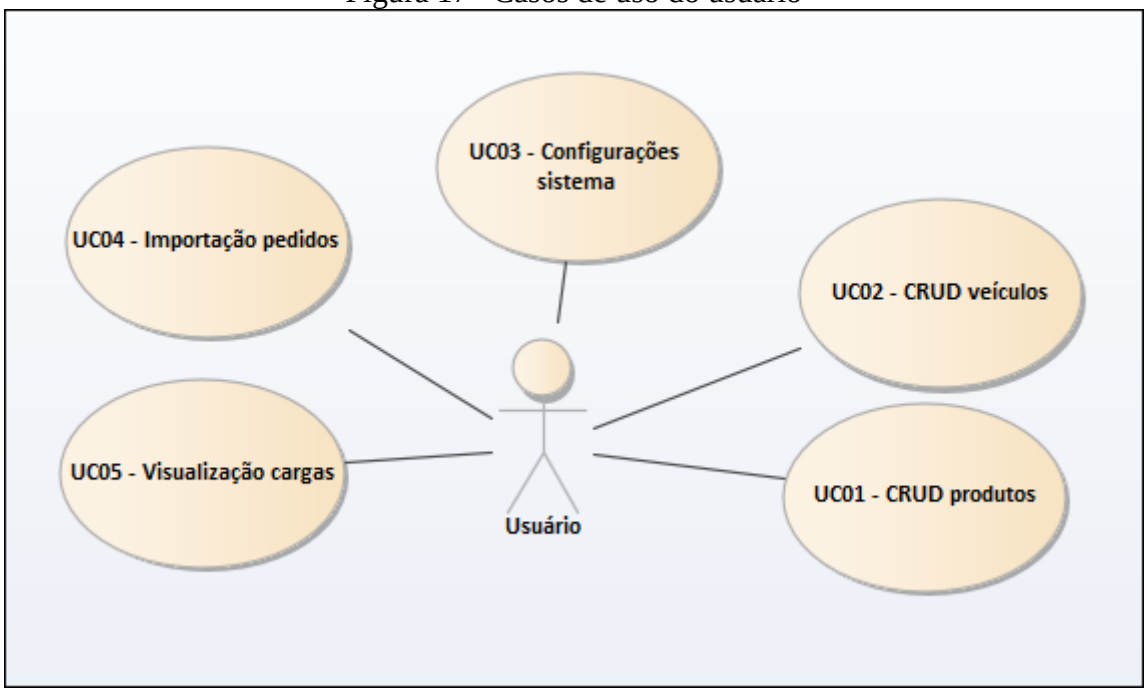
3.2 ESPECIFICAÇÃO

Para especificação do trabalho foi utilizada a ferramenta Enterprise Architect (EA) para a criação de diagramas de casos de uso, atividades, classes e pacotes.

3.2.1 Diagrama de casos de uso

A partir do levantamento dos requisitos do sistema, foram desenvolvidos dois diagramas de casos de uso, o primeiro, conforme ilustrado na Figura 17 representa as funcionalidades disponíveis para o ator `Usuário`, o segundo descreve as ações realizadas pelo ator `Worker` e pode ser visualizado na Figura 18.

Figura 17– Casos de uso do usuário

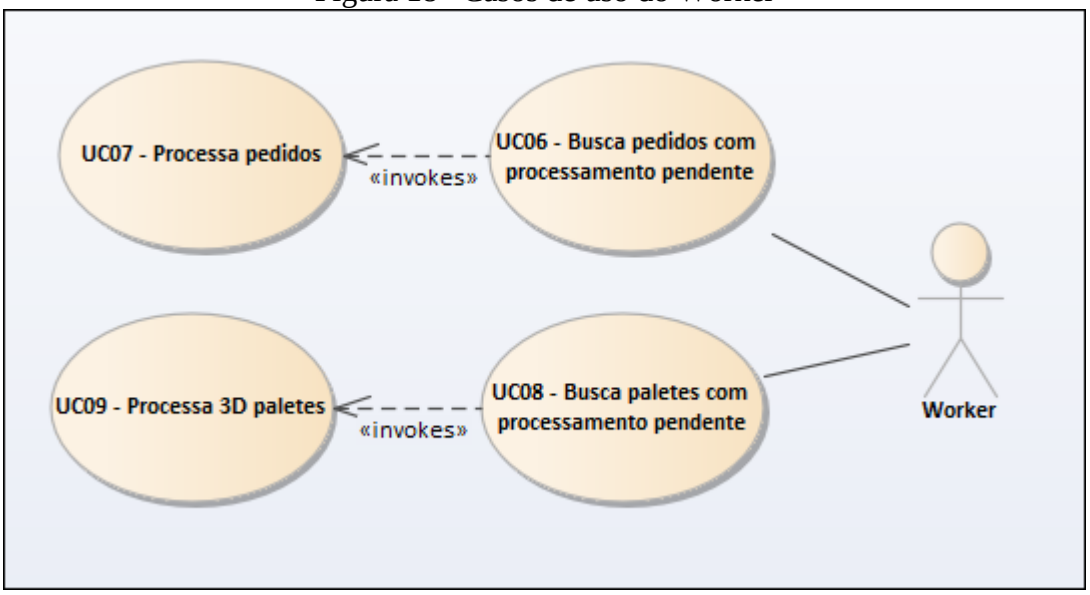


Fonte: elaborado pelo autor.

O primeiro caso de uso `UC01 - CRUD produtos` é disponibilizado a manutenção de todos os produtos cadastrados pelo usuário no sistema. O caso de uso `UC02 - CRUD veículo` é disponibilizado a manutenção de todos os veículos cadastrados pelo usuário no sistema.

O caso de uso `UC05 - Visualizar carga` é disponibilizado a visualização de todas as cargas processadas pelo sistema. O caso de uso `UC04 - Importar pedidos` permite que o usuário importe os pedidos através de um arquivo no formato *JavaScript Object Notation* (JSON) para que o sistema realize o processamento e gere uma nova.

Figura 18– Casos de uso do Worker



Fonte: elaborado pelo autor.

O Caso de uso `UC06 – Buscar pedidos com processamento pendente` representa a consulta realizada pelo sistema para buscar os pedidos importados. O Caso de uso `UC07 – Processar pedidos` é a ação realizada pelo sistema para realizar a paletização dos pedidos e gerar uma carga. O Caso de uso `UC08 – Buscar paletes com processamento pendente` representa a consulta realizada pelo sistema para buscar os paletes com processamento pendente gerados pelo `UC07`. O Caso de uso `UC09 – Processar 3D paletes` é a ação realizada pelo sistema para realizar o cálculo 3D do palete.

3.2.2 Diagrama de atividades

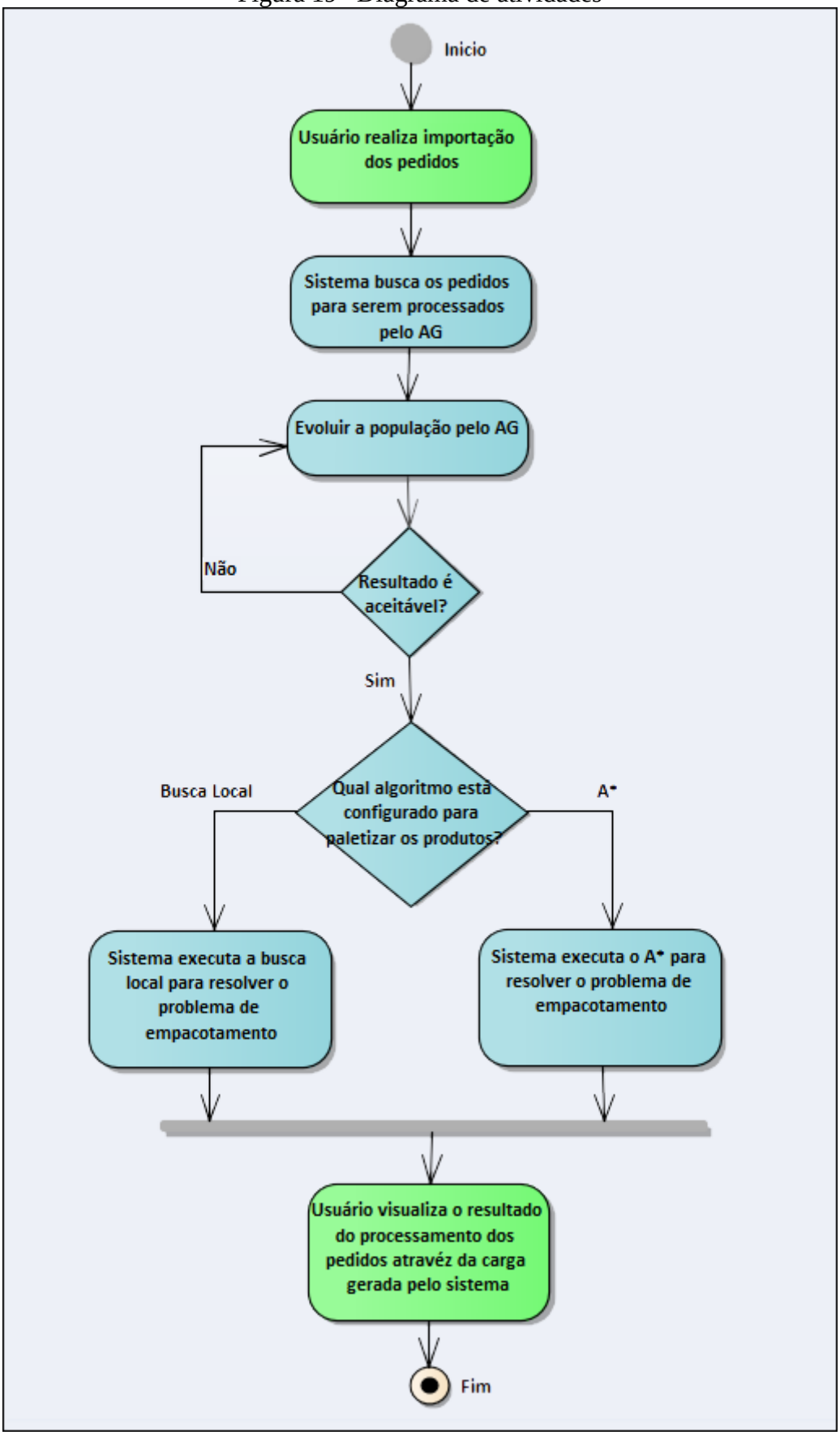
A Figura 19 apresenta o diagrama de atividades do sistema. Para facilitar o entendimento do processo, as atividades atribuídas ao usuário estão marcadas em verde, e as atividades que o sistema é responsável por realizar estão marcadas em azul.

O fluxo inicia quando o usuário realiza a importação dos pedidos no sistema, essa lista de pedidos contém a ordem de entrega, a placa do veículo que irá realizar o transporte e os produtos solicitados, no formato apresentado no Apêndice A. O sistema então busca os pedidos importados pelo usuário para iniciar o processamento, com a lista de pedidos importados pelo usuário o sistema identifica quais são as caixas que devem ser carregas no veículo solicitado. É então criado as populações iniciais pelo AG, o sistema busca a quantidade de baias (compartimentos onde são colocados os paletes) que o veículo possui e cria aleatoriamente paletes com as caixas dos produtos solicitados. O AG então evolui a população até que a disposição dos itens esteja de forma satisfatória, ou um limite de execuções seja atingido.

Ao término da execução do AG, é disponibilizado a lista dos produtos que devem ser colocados sobre os paletes para cada baia do caminhão. O sistema neste momento verifica nas configurações qual o algoritmo que está configurado para realizar o processamento 3D dos paletes, realizando o processamento do palete com o algoritmo configurado.

Ao final da execução é disponibilizado ao usuário a visualização 3D total da carga e dos paletes individualmente, a carga representa todos os paletes com seus respectivos produtos alocados que devem ser carregados no veículo solicitado pelo usuário.

Figura 19– Diagrama de atividades

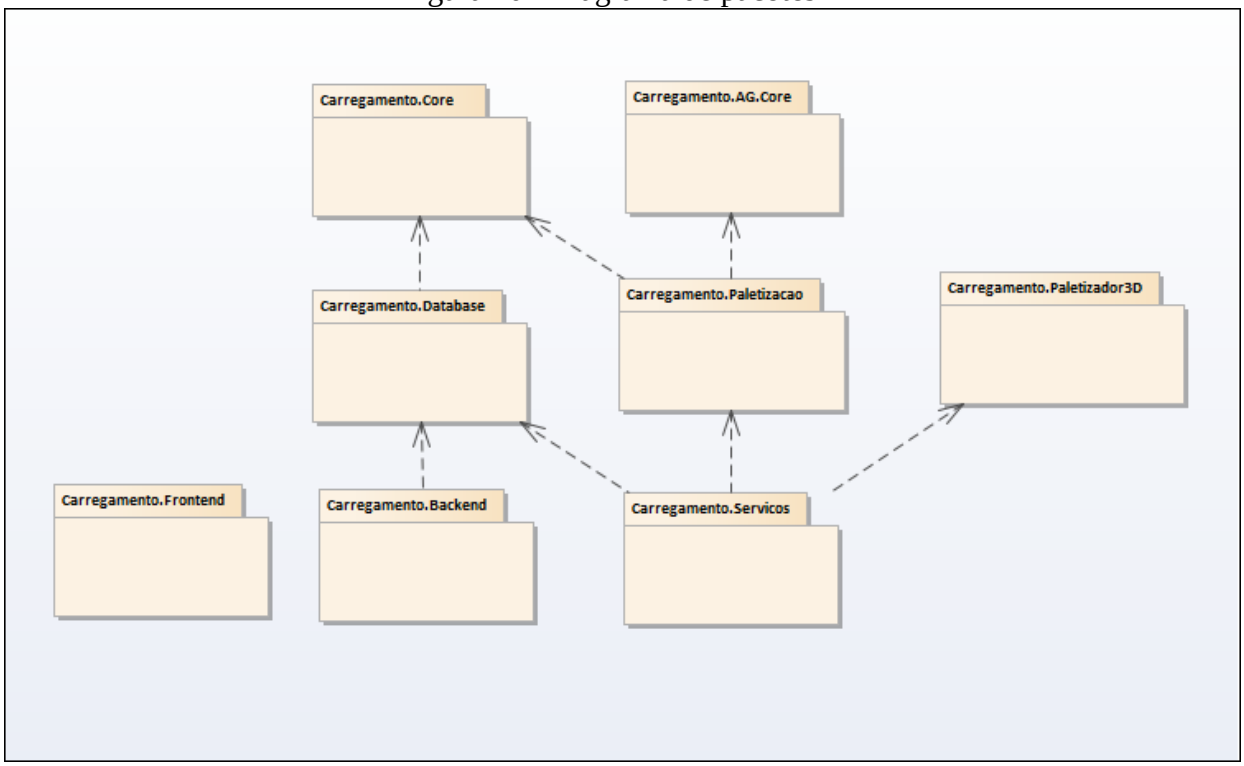


Fonte: elaborado pelo autor.

3.2.3 Diagrama de pacotes

Nessa seção será apresentada a arquitetura dos pacotes utilizados para o desenvolvimento deste trabalho. A Figura 20 apresenta o diagrama de pacotes do sistema.

Figura 20– Diagrama de pacotes



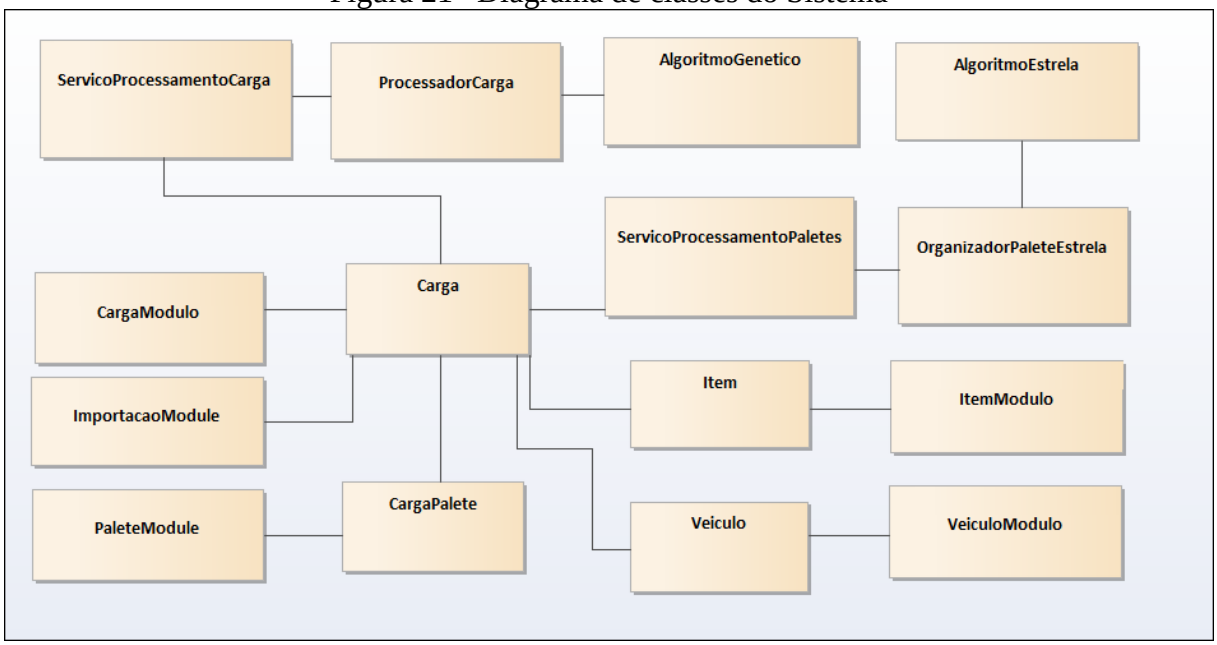
Fonte: elaborado pelo autor.

O pacote `Carregamento.Servicos` é responsável por iniciar o serviço de paletização e o serviço de cálculo 3D do palete. O pacote `Carregamento.Paletizador3D` é responsável pela execução do serviço que irá executar o cálculo do posicionamento 3D das caixas em um palete. O Pacote `Carregamento.Paletizacao` e `Carregamento.AG.Core` são responsáveis por realizar o processamento de uma carga realizando o processo de paletização. O pacote `Carregamento.Backend` é responsável por disponibilizar e atualizar todos os registros do sistema. O pacote `Carregamento.FrontEnd` é reponsável por disponibilizar todo o conteúdo estático da aplicação web do sistema. O pacote `Carregamento.Core` é responsável pelo modelo de dados da aplicação e classes comuns do sistema. Por fim, o pacote `Carregamento.Database` é responsável pela conexão com o banco de dados.

3.2.4 Diagrama de classes

Nessa seção é abordado o diagrama de classes do sistema. A Figura 21 apresenta o diagrama de classes sem a informação de métodos e atributos nas classes.

Figura 21– Diagrama de classes do Sistema



Fonte: elaborado pelo autor.

Nas classes `ItemModulo` e `VeiculoModulo` são realizadas as operações de cadastro e atualização dos itens e veículos no sistema. As classes `Item` e `Veiculo` representam os objetos que serão persistidos e manipulados no banco de dados. A classe `Item` representa o modelo de dados para o armazenamento dos itens, onde é informado o código, descrição, altura, largura e comprimento. A classe `Veiculo` representa o modelo de dados para armazenamento dos veículos, onde é informado a placa e a quantidade de baias que o veículo possui.

Para a importação de uma carga no sistema, a classe `ImportacaoModule` é responsável por receber o arquivo que contém os pedidos de uma carga a ser processada. A classe `Carga` representa o modelo de dados das cargas processadas pelo sistema, nela são informados os pedidos a serem processados e os paletes processados pelo sistema.

Após realizado a importação de uma `Carga` a classe `ServicoProcessamentoCarga` é responsável por consultar as cargas que precisam ser processadas. Estas cargas são passadas para a classe `ProcessadorCarga` que monta as informações necessárias para o processamento pelo algoritmo AG. A classe `AlgoritmoGenetico` é responsável pela execução das operações genéticas.

A classe `ServicoProcessamentoPaletes` é responsável por consultar os paletes que precisam ser processados, estes paletes são encaminhados para a classe `OrganizadorPaleteEstrela` que é responsável por preparar os dados necessários para a execução do algoritmo A*. A classe `AlgoritmoEstrela` é reponsável pela execução das técnicas do algoritmo A*.

A classe CargaModulo é reponsável por consultar as cargas processadas pelo sistema, e a classe PaleteModule é responsável por consultar os paletes processados pelo sistema.

3.3 IMPLEMENTAÇÃO

Nessa seção são mostradas as ferramentas e técnicas utilizadas para implementar o presente trabalho.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação LIP foi desenvolvida utilizando a linguagem de programação C# através da IDE Visual Studio na versão 2015 em conjunto com a ferramenta de gerenciamento de dependências Nuget e também a linguagem de programação Java, dentro do ambiente de desenvolvimento do Eclipse Neon na versão 4.6.3. O banco de dados utilizado foi o MongoDB em conjunto com o cliente MongoDB.Driver que o mesmo disponibiliza para aplicações desenvolvidas em C#.

Para apresentar do resultado do algoritmo dentro de um ambiente 3D foi utilizada a biblioteca Three.js, esta ferramenta foi utilizada pois facilita a criação de objetos tridimensionais através de uma utilização simplificada do WebGL.

O desenvolvimento do trabalho foi realizado em quatro estágios: O primeiro estágio foi a criação de um AG para o processamento das cargas. O segundo e o terceiro estágio foram desenvolvidos dois algoritmos capazes de calcular o posicionamento tridimensional dos produtos dentro do palete. O quarto estágio foi o desenvolvimento de uma aplicação web, que tem como objetivo disponibilizar ao usuário um meio para realizar as entradas no sistema, e visualizar os resultados obtidos pelo sistema. Nas seções seguintes são abordadas as etapas do desenvolvimento.

3.3.1.1 Primeiro estágio de desenvolvimento

O primeiro estágio é a criação de um AG para o processamento das cargas importadas utilizando a linguagem de programação C#. O AG foi desenvolvido de forma genérica, ou seja, pode ser estendido para solução de qualquer outro problema. O Quadro 1 apresenta um trecho do código onde é iniciada a execução do AG para solucionar o problema do carregamento.

Quadro 1 – Inicialização da execução do AG

```
01 var caixas = ObterCaixasAG(carga, produtos);
02 var configuracaoAlgoritmo = ObterConfiguracoesAG();
03 var geradorPopulacaoInicial = ObterGeradorPopulacaoInicial(
   veiculo, caixas);
04 var avaliador = ObterMetodoAvaliacao();
05 var metodoSelecao = ObterMetodoDeSelecao();

06 var algoritmo = new AlgoritmoGenetico<CargaAG>(
   configuracaoAlgoritmo, geradorPopulacaoInicial,
   avaliador, metodoSelecao);
07 algoritmo.AdicionarExecucaoExata(new OrdenadorProdutosPalete());

08 var resultado = algoritmo.Executar();
```

Fonte: elaborado pelo autor.

Na linha 1 do Quadro 1 é criado as caixas que serão processadas pelo AG. Nesta etapa é buscado as características dos produtos contidos na carga a ser processada (ordem entrega) junto com as configurações definidas para o item no sistema (volume, resistência). Na linha 2 é buscado as configurações do AG que contém o tamanho da população e a quantidade máxima de iterações que o AG irá realizar. Na linha 3 é criado o serviço responsável pela criação da população inicial do AG, esse serviço leva em consideração as características do veículo (quantidade de baias) e as caixas que serão processadas. Na linha 4 e 5 são inicializados os métodos de avaliação e seleção do AG respectivamente. Na linha 6 então é criado a instancia do AG passando por parâmetro todas as configurações necessárias. Na linha 7 é acrescentado um método de exato que será aplicado a todos os cromossomos a cada iteração do AG, este método é responsável por ordenar pela ordem de entrega os produtos dentro do palete calculado pelo AG. Por fim, na linha 8 é realizado a execução do AG e obtido seu resultado.

O AG evolui a população até que seja obtida uma solução desejável ou um número máximo de iterações seja alcançado. No Quadro 2 pode ser visualizado um trecho de código onde é evoluído a população até que um critério de parada seja atendido.

Quadro 2 – Inicialização da execução do AG

```
01 var populacaoEvoluir = GerarPopulacaoInicial();
02 var qtdExecucoes = _config.QuantidadeMaximaExecucoes;
03 for (var i = 0; i < qtdExecucoes; i++)
04 {
05     populacaoEvoluir = EvoluirPopulacao(populacaoEvoluir);
06
07     var melhor = populacaoEvoluir.ObterCromossomoPorIndice(0);
08     if (_avaliadorCromossomo.EhMeta(melhor.Aptidao))
09     {
10         break;
11     }
12 }
13 return populacaoEvoluir.ObterCromossomoPorIndice(0);
```

Fonte: elaborado pelo autor.

Na linha 1 do Quadro 2 é gerado a população inicial que será evoluída, conforme pode ser visto na linha 5. Na linha 7 é obtido o melhor cromossomo da iteração para ser checado na linha 8 se é uma solução desejada. Ao final das iterações é obtido a melhor solução alcançada e retornada como pode ser observado na linha 13.

Para gerar a população inicial, foi desenvolvido um algoritmo que randomicamente distribui as caixas a serem organizadas pelo AG em uma quantidade de paletes equivalente a quantidade de baias do veículo que será carregado, gerando então um cromossomo. O Quadro 3 apresenta um trecho de código onde é realizado a criação de um cromossomo para a população inicial.

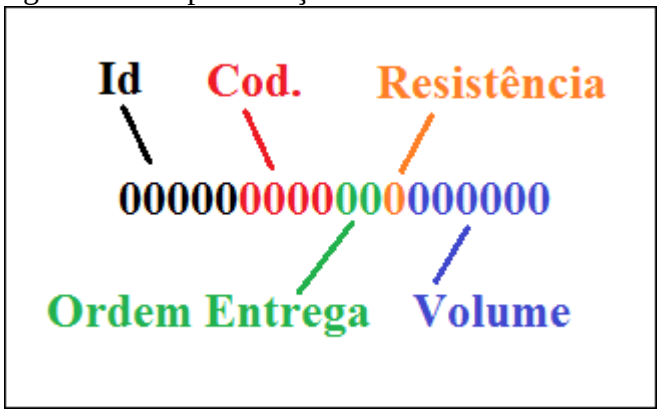
Quadro 3 – Criação da população inicial

```
01 var qtdPaletes = _quantidadeBaias;
02 var paletes = new List<List<string>>();
03 for (var i = 0; i < qtdPaletes; i++)
04 {
05     paletes.Add(new List<string>());
06 }
07 while(_caixas.Count > 0)
08 {
09     var indexCaixaSelecionada = Randon(0, _caixas.Count);
10     var caixaSelecionada = _caixas[indexCaixaSelecionada];
11     _caixas.RemoveAt(indexCaixaSelecionada);
12     var indexPaleteSelecionado = Randon(0, qtdPaletes);
13     paletes[indexPaleteSelecionado]
           .Add(caixaSelecionada.ToString());
14 }
15 return new CargaAG(paletes);
```

Fonte: elaborado pelo autor.

Na linha 2 do Quadro 3 é inicializado uma lista onde serão adicionados os paletes criados na linha 5. Na linha 7 é dado início a distribuição das caixas nos paletes de forma aleatória, para o AG a caixa é representada em forma de caracteres e suas especificações podem ser visualizadas na Figura 22.

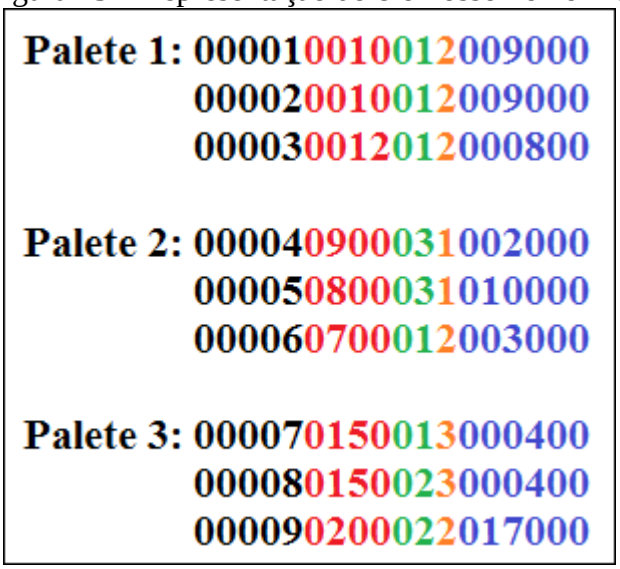
Figura 22 – Representação da caixa utilizada no AG



Fonte: elaborado pelo autor.

A representação do cromossomo é realizada através de uma lista de paletes que contém uma lista de caixas em forma de caracteres, a Figura 22 mostra como um cromossomo é representado dentro do AG.

Figura 23 – Representação do cromossomo no AG



Fonte: elaborado pelo autor.

Todo cromossomo criado é submetido a um cálculo de aptidão, etapa necessária para verificar quão bom é a solução gerada. A aptidão no AG desenvolvido considera duas restrições, a primeira representada pelo Quadro 4, verifica se a capacidade de cada palete da carga foi ultrapassada e a segunda representada pelo Quadro 5, verifica se as ordens de entrega das caixas foram respeitadas.

O Quadro 4 apresenta um trecho do código onde a restrição do volume do palete é considerada. Na linha 2 é realizado a iteração em todos os paletes da carga para verificar se o volume foi excedido, após isso na linha 5 é aberto um comando de laço de repetição para somar o volume total das caixas contidas no palete. Por fim, na linha 10 é verificado se a capacidade do palete foi ultrapassada e caso positivo, na linha 12 é acrescentado a aptidão o volume que foi ultrapassado da capacidade do palete.

Quadro 4 – Cálculo aptidão capacidade palete

```
01 var aptidao = 0;
02 foreach (var palete in cromossomo.Paletes)
03 {
04     var volumeDosProdutos = 0;
05     foreach (var caixa in palete)
06     {
07         var volumeCaixa = caixa.Substring(12, 6);
08         volumeDosProdutos += volumeCaixa;
09     }
10     if (volumeDosItens > volumePalete)
11     {
12         aptidao += (volumeDosProdutos - volumePalete);
13     }
14 }
15 return aptidao;
```

Fonte: elaborado pelo autor.

O Quadro 5 apresenta o cálculo da aptidão considerando a restrição por ordem de entrega, a aplicação dessa restrição faz com que as caixas a serem entregues por último, sejam as primeiras a serem alocadas no palete. Na linha 2 é realizado uma iteração sobre todos os paletes para verificar uma violação nas ordens de entregas, após isso na linha 5 é realizado uma repetição para verificar a ordem de entrega de cada caixa. Na linha 8 é verificado se a ordem de entrega da caixa atual é maior que a ordem de entrega da caixa anterior, e caso positivo na linha 9 é acrescentado um ponto na aptidão.

Quadro 5 – Cálculo aptidão ordem entrega

```
01 var aptidao = 0;
02 foreach (var caixas in cromossomo.Paletes)
03 {
04     var entregaAnterior = int.MaxValue;
05     foreach (var caixa in caixas)
06     {
07         var ordemEntrega = caixa.Substring(9,2);
08         if (ordemEntrega > entregaAnterior)
09             aptidao++;
10         entregaAnterior = ordemEntrega;
11     }
12 }
13 return aptidao;
```

Fonte: elaborado pelo autor.

O método de seleção escolhido para ser implementado foi a seleção por roleta. O Quadro 6 apresenta um trecho de código onde a seleção é realizada. Primeiramente é realizado a soma da aptidão de toda a população, como podemos observar na linha 1, logo em seguida é montado a roleta calculando o percentual que cada cromossomo irá ocupar, conforme a linha 2. Na linha 4 é gerado um número randômico que esteja dentro da roleta, então na linha 7 é iniciado um laço de repetição para buscar o índice do cromossomo sorteado. Por fim, na linha 12 é retornado o cromossomo sorteado.

Quadro 6 – Método de seleção por roleta

```
01 var somaAptidoes = cromossomos.Sum(x => x.Aptidao);
02 var total = cromossomos.Sum(x => somaAptidoes / x.Aptidao);
03
04 var randon = Randon(0, total);
05 var indice = -1;
06 var totalParcial = 0;
07 do
08 {
09       i++;
10       totalParcial += somaAptidoes/cromossomos[i].Aptidao;
11 } while (totalParcial >= randon);
12 return cromossomos[i];
```

Fonte: elaborado pelo autor.

A técnica de cruzamento escolhida para ser implementada foi o cruzamento de dois pontos, e foi realizado selecionando um palete aleatório de cada cromossomo e trocando os genes de um ponto de cruzamento randomicamente calculado. O Quadro 7 mostra o trecho de código onde o cruzamento é realizado. Primeiramente é escolhido um palete para realizar o cruzamento, como podemos visualizar na linha 3, logo após é gerado randomicamente o ponto de início e fim do cruzamento, como pode ser observado nas linhas 4 e 5 respectivamente. O conteúdo do cruzamento de cada palete é obtido conforme mostrado nas linhas 8 e 9 e logo em seguida é retirado dos cromossomos de destino, sendo então inseridos no ponto de cruzamento calculado anteriormente como pode ser visto nas linhas 20 e 21. Por fim, após o cruzamento ter sido realizado os novos cromossomos são retornados como apresentado na linha 22.

Quadro 7 – Cruzamento

```
01 var carga1 = cromossomo.Clone();
02 var carga2 = proximoCromossomo.Clone();
03 var indicePaleteCruzamento = Randon(0, cromossomo.Paletes.Count);
04 var inicioCruzamento = Randon(0, tamanhoPalete);
05 var fimCruzamento = Randon(inicioCruzamento, tamanhoPalete);
06 var paleteCruzamento1 = carga1.Paletes[indicePaleteCruzamento];
07 var paleteCruzamento2 = carga2.Paletes[indicePaleteCruzamento];
08 var partePalete1 = paleteCruzamento1.GetRange(inicioCruzamento,
   fimCruzamento);
09 var partePalete2 = paleteCruzamento2.GetRange(inicioCruzamento,
   fimCruzamento);
10 foreach (var caixa in partePalete1) {
11     foreach (var palete in carga2.Paletes) {
12         if (palete.Contains(caixa))
13             palete.Remove(caixa);
14 }}
15 foreach (var caixa in partePalete2) {
16     foreach (var palete in carga1.Paletes){
17         if (palete.Contains(caixa))
18             palete.Remove(caixa);
19 } }
20 paleteCruzamento1.InsertRange(inicioCruzamento, partePalete2);
21 paleteCruzamento2.InsertRange(inicioCruzamento, partePalete1);
22 return new List<CargaAG> { carga1, carga2 };
```

Fonte: elaborado pelo autor.

Após realizar o cruzamento, o sistema realiza a mutação dos cromossomos, o processo de mutação implementado para o AG consiste basicamente de trocar de lugar duas caixas aleatoriamente. O Quadro 8 demonstra passo-a-passo o processo de mutação. Após receber um cromossomo, este cromossomo é clonado para dar início ao processo de mutação como pode ser visualizado na linha 1. Nas linhas 2 e 3 são buscados dois paletes aleatórios para realizar a troca de uma caixa que eles possam conter. Os locais das caixas a terem suas posições trocadas é obtido também de forma randômica nas linhas 6 e 7. A troca é realizada nas linhas 10, 11, 12 e 1, primeiramente retira-se as caixas do seu local original e em seguida são adicionadas no local da troca.

Quadro 8 – Mutação

```
1  var cargaMutada = cromossomo.Clone();
2  var paleteMutacao1 = Rand(0, cargaMutada.Paletes.Count);
3  var paleteMutacao2 = Rand(0, cargaMutada.Paletes.Count);

4  var qtdProdPalete1 = cargaMutada.Paletes[paleteMutacao1].Count;
5  var qtdProdPalete2 = cargaMutada.Paletes[paleteMutacao2].Count;

6  var localMutacao1 = Rand(0, qtdProdPalete1);
7  var localMutacao2 = Rand(0, qtdProdPalete2);

8  var valorMutacao1 =
   cargaMutada.Paletes[paleteMutacao1][localMutacao1];
9  var valorMutacao2 =
   cargaMutada.Paletes[paleteMutacao2][localMutacao2];

10 cargaMutada.Paletes[paleteMutacao1].RemoveAt(localMutacao1);
11 cargaMutada.Paletes[paleteMutacao2].RemoveAt(localMutacao2);

12 cargaMutada.Paletes[paleteMutacao1].Insert(localMutacao1,
   valorMutacao2);
13 cargaMutada.Paletes[paleteMutacao2].Insert(localMutacao2,
   valorMutacao1);

14 return cargaMutada
```

Fonte: elaborado pelo autor.

3.3.1.2 Segundo estágio de desenvolvimento

No segundo estágio de desenvolvimento foi implementado um algoritmo em C# para realizar a organização tridimensional das caixas dentro do palete utilizando uma estratégia gulosa. Como entrada do algoritmo é dado um palete com os produtos ordenados pela ordem de montagem, previamente calculados pelo AG desenvolvido no primeiro estágio de desenvolvimento.

O Quadro 9 demonstra passo-a-passo o cálculo da posição das caixas dentro do palete. Após receber o palete e as caixas que devem ser organizadas, a linha 3 pega o vértice inicial do palete (neste caso x,y,z com valores 0,0,0 respectivamente). Então a linha 4 inicia um laço

de repetição até que todas as caixas tenham sido organizadas, a caixa é considerada organizada quando estiver em um vértice dentro do palete. A busca pelo vértice que consegue adicionar a caixa acontece na linha 8, que uma vez encontrado tem suas coordenadas atribuídas a caixa pela linha 12, após a caixa ser atribuída a um vértice, são buscados os novos vértices criados. Caso nenhum vértice possa alocar a caixa a linha 16 é responsável por parar o algoritmo, retornando então o resultado obtido até o momento.

Quadro 9 – Organização tridimensional das caixas com método guloso

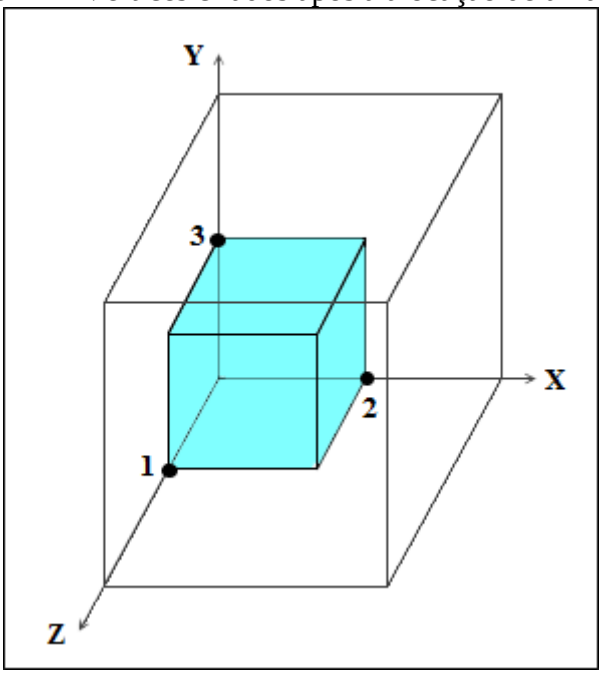
```
01 var caixasOrganizadas = new List<Caixa>();
02 var caixasOrganizar = caixas.ToList();
03 var vertices = BuscarVertices(palete);
04 while (caixasOrganizar.Any())
05 {
06     var caixaOrganizar = caixasOrganizar[0];
07     caixasOrganizar.RemoveAt(0);
08     var verticeAdicionar = vertices
           .FirstOrDefault(x =>
              x.ConsegueAdicionar(caixaOrganizar));

09     if (verticeAdicionar != null)
10     {
11         caixasOrganizadas.Add(caixaOrganizar);
12         verticeAdicionar.AdicionarCaixa(caixaOrganizar);
13     }
14     else
15     {
16         break;
17     }
18     var novosVertices = BuscarNovosVertices(verticeAdicionar);
19     vertices.Add(novosVertices);
20 }
21 return caixasOrganizadas;
```

Fonte: elaborado pelo autor.

Uma vez alocado uma caixa dentro do palete, são criados novos vértices para alocas as próximas caixas. A Figura 24 mostra os três novos vértices após a alocação da primeira caixa do palete. Ao realizar a busca os vértices são retornados na mesma sequência apresentada na imagem, de modo que as alocações das caixas sejam primeiramente na direção de Z, depois na direção de X e por último na direção de Y. Esta ordenação garante que a base do palete seja preenchida para que depois novas caixas possam ser alocadas na parte superior.

Figura 24 – Vértices criados após a alocação de uma caixa



Fonte: elaborado pelo autor.

3.3.1.3 Terceiro estágio de desenvolvimento

Neste estágio de desenvolvimento foi implementado outro algoritmo em Java para realizar a organização tridimensional das caixas dentro do palete, este, porém, foi implementado utilizando a técnica do A*. Diferente da busca gulosa, este algoritmo expande a busca para todos os vértices criados a partir de uma alocação, e o próximo estado a ser expandido é determinado por uma heurística.

No Quadro 10 é apresentado um trecho de código com a implementação do algoritmo A*. Após ter recebido um estado inicial, a linha 1 inicia uma lista de estados abertos ordenados pelo resultado da sua função heurística, logo em seguida é aberto um laço de repetição que percorre até que não tenham mais nós abertos para serem explorados ou um comando de parada seja acionado. Na linha 7 é obtido o melhor estado aberto e caso ele seja uma solução aceita a linha 9 retorna este estado como solução do problema. Na linha 11 são buscados os estados sucessores do estado atual e adicionados na lista de nós abertos, a linha 12 verifica se o nó atual tem sua heurística melhor do que a melhor solução encontrada até o momento. Caso isso seja verdade, a linha 13 atribui o nó atual como sendo a melhor solução encontrada.

Quadro 10 – Execução do algoritmo A*

```
01  Queue<Nodo> abertos =
        new PriorityQueue<Nodo>(100, getNodoComparatorF());
02  Nodo nInicial = new Nodo(inicial, null);
03  abertos.add(nInicial);
04  Nodo melhorSolucao = nInicial;
05
06  while (!parar && abertos.size() > 0) {
07      Nodo melhor = abertos.remove();
08      if (melhor.estado.ehMeta()) {
09          return melhor;
10      }
11      abertos.addAll( sucessores(melhor) );
12      if (melhor.f() < melhorSolucao.f()) {
13          melhorSolucao = melhor;
14      }
15  }
16  return melhorSolucao;
```

Fonte: elaborado pelo autor.

No Quadro 11 é apresentado como a criação dos estados sucessores são gerados a partir de um estado sendo explorado. No nó sendo pesquisado é gerado um novo sucessor para cada vértice disponível dentro do palete, isso pode ser visualizado na linha 2, neste vértice é realizado uma tentativa de alocação da próxima caixa a ser alocada, isso pode ser visualizado na linha 7 que verifica se a caixa não ultrapassou os limites do palete e na linha 9 que verifica se a caixa não entrou em colisão com nenhuma caixa previamente alocada. Caso a alocação tenha sido possível, o novo estado é adicionado na lista de sucessores na linha 15.

Quadro 11 – Busca dos novos sucessores

```
01  List<Estado> suc = new LinkedList<Estado>();
02  for(int i = 0; i < vertices.size(); i++) {
03      EstadoPalete novoEstado =
            new EstadoPalete(vert, cxPendentes, cxAdicionadas, x, y, z);
04      int[] vertice = novoEstado.vertices.remove(i);
05      int[] caixaAdicionar = novoEstado.caixasPendentes.remove(0);
06
07      if(!novoEstado
           .podeAdicionarNoPontoSolicitado(vertice, caixaAdicionar))
08          continue;
09      if(novoEstado
           .temProdutoNoPontoSolicitado(cxAdicionadas, vertice,
              cxAdicionar))
10          continue;
11
12      novoEstado.adicionar(vertice, cxAdicionar);
13      novoEstado.vertices
           .addAll(buscarVertice(vertice, cxAdicionar));
14      novoEstado.removerVerticesDuplicados();
15      suc.add(novoEstado);
16  }
17  return suc;
```

Fonte: elaborado pelo autor.

A lista de estados percorridos pelo algoritmo A* é ordenado por uma função heurística que determinada a qualidade dos estados, de modo que quanto melhor o valor retornado pela

função heurística, mais qualificado é o estado. Para o algoritmo proposto foram combinadas quatro heurísticas como podemos visualizar no Quadro 12.

Quadro 12 – Calculo da heurística

```
01 public int h() {
02    Double h = h1() * 15 + h2() * 3 + h3() * 15 + h4() * 0.25;
03    return h.getIntValue();
04 }
```

Fonte: elaborado pelo autor.

A heurística `h1` apresentada pelo Quadro 13 retorna a quantidade de caixas pendentes de serem alocadas no palete, esta heurística é importante pois garante a profundidade da busca que o algoritmo está realizando.

Quadro 13 – Código da heurística `h1`

```
01 public int h1() {
02     return this.caixasPendentes.size();
03 }
```

Fonte: elaborado pelo autor.

A heurística `h2` retorna a quantidade de vértices inutilizáveis, ou seja, que não é possível alocar nenhuma caixa no mesmo. No Quadro 14 é apresentado detalhadamente o funcionamento desta heurística. A linha 3 cria um laço de repetição para cada vértice do palete, logo em seguida a linha 5 passa todas as caixas com alocação pendente verificando pela linha 6 se ela pode ser adicionada no vértice que está sendo investigado. Caso nenhuma caixa possa ser alocada no vértice ele é considerado perdido pela linha 9. Por fim a linha 12 retorna a quantidade total dos vértices que foram perdidos.

Quadro 14 – Código da heurística `h2`

```
01 public int h2() {
02   int qtdVerticesPerdidos = 0;
03   for(int[] vertice : vertices){
04     boolean perdido = true;
05       for(int[] caixa : caixasPendentes){
06         if(podeAdicionarNoPontoSolicitado(vertice, caixa))
07           perdido = false;
08       }
09       if(perdido)
10         qtdVerticesPerdidos++;
11   }
12   return qtdVerticesPerdidos;
13 }
```

Fonte: elaborado pelo autor.

A heurística `h3` é responsável retornar a quantidade de caixas alocadas para cada andar do palete, de modo que quanto maior for a quantidade de caixas por andar menor é o resultado da heurística, esta heurística ajuda a determinar a qualidade do estado, uma vez que, quanto maior a quantidade de caixas em um único andar, maior será a chance de alocar um maior número de caixas.

O Quadro 15 demonstra passo-a-passo como este cálculo é realizado. Na linha 2 é iniciado uma lista para guardar os andares do palete formados pelas caixas, então na linha 3 é realizado uma iteração por todas as caixas adicionadas no palete e logo em seguida, na linha 5 é verificado se já existe um andar adicionado para a posição Y da caixa em questão, caso não tenha esta posição Y é adicionada como um novo andar existente no palete. Na linha 12 é então retornado o valor da função, tirando a média das caixas adicionadas por andares do palete e utilizando este valor como dividendo de 100, de modo que quanto menor a média de caixas por andar maior será o valor retornado pela função.

Quadro 15 – Código da heurística `h3`

```
01 public int h3(){
02   ArrayList<Integer> andares = new ArrayList<Integer>();
03   for(int[] caixa : caixasAdicionadas){
04     int posicaoYCaixa = caixa[6];
05     if(!andares.contains(posicaoYCaixa)){
06       andares.add(posicaoYCaixa);
07     }
08   }
09   if(caixasAdicionadas.size() == 0){
10     return 0;
11   }
12   return 100 / (caixasAdicionadas.size() / andares.size());
13 }
```

Fonte: elaborado pelo autor.

A heurística `h4` é responsável por garantir que a montagem do palete será efetuada da base para o topo, evitando que um item seja adicionado em um andar superior sem ter antes estressado as possibilidades de alocação nos andares inferiores. Basicamente, como podemos observar no Quadro 16, a linha 5 busca a última caixa adicionada no palete, e a linha 6 busca a penúltima caixa adicionada no palete. Na linha 8 é retornado a distância entre as duas caixas, deste modo o algoritmo está sujeito a adicionar as caixas perto das demais caixas já alocadas sem criar andares desnecessários.

Quadro 16 – Código da heurística `h4`

```
01 public int h4(){
02   int qtdAdicionadas = caixasAdicionadas.size();
03   if(qtdAdicionadas > 2 && vertices.size() > 0){
04     int[] vertice = vertices.get(0);
05     int[] ultimaCaixa = caixasAdicionadas.get(qtdAdicionadas-1);
06     int[] penultimaCaixa = caixasAdicionadas.get(qtdAdicionadas-2);
07     if(ultimaCaixa[6] > penultimaCaixa[6]){
08       return (posicaoUltimaCaixa) - (posicaoPesultiaCaixa);
09     }
10     return 0;
11   }
12   return 0;
13 }
```

Fonte: elaborado pelo autor.

3.3.1.4 Quarto estágio de desenvolvimento

Neste quarto e último estágio de desenvolvimento, será apresentado a maneira que foi construída a aplicação web do sistema LIP. A aplicação web foi construída utilizando HTML5 e o framework Bootstrap para o *desing* das telas. Para o desenvolvimento da visualização tridimensional foi utilizado a biblioteca javascript three.js, e para expor a API do servidor foi feito o uso do framework Nancy com a linguagem C#.

As requisições para buscar os dados da carga são realizadas via AJAX e consumidas por um *endpoint* Nancy. O Quadro 17 apresenta uma requisição do AJAX para o Nancy via método *GET*.

Quadro 17 – Requisição via AJAX para o servidor

```
01 $.ajax({
02   url: "http://localhost:5566/cargas/" + id + "/palete/M1",
03   type: 'GET'
04   success: function (data) {
05     PalletDrawer.draw(data);
06   },
07   error: function (ex) {
08     alert('Error: ' + ex);
09   }
10 });
```

Fonte: elaborado pelo autor.

Na linha 2 até a linha é configurada a chamada AJAX para o servidor com as informações do endereço e o tipo da requisição e será enviado. Os dados da requisição são retornados no formato JSON onde, posteriormente, será carregado um palete com as caixas nas coordenadas retornadas.

Para a criação do palete foi utilizado a biblioteca three.js. No Quadro 18 é demonstrado o trecho de código que realiza a criação tridimensional do palete.

Quadro 18 – Criação palete 3D com three.js

```
01 this.scene = new THREE.Scene();
02 for (var i = 0; i < qtdCaixas; i++) {
03   var caixa = palete.caixas[i];
04   var cor = Math.random() * 0xffffff;
05
06   var geometry = new THREE.BoxGeometry(largura, altura, comprimento);
07   var object = new THREE.Mesh(geometry, new
08         THREE.MeshLambertMaterial({ color:cor }));
09
10   object.position.x = caixa.x;
11   object.position.y = caixa.y;
12   object.position.z = caixa.z;
13   this.scene.add(object); }
```

Fonte: elaborado pelo autor.

Na linha 1 é criado a cena onde o palete será desenhado e logo após, na linha 3 é iniciado um laço de repetição para adicionar as caixas do palete na cena. Na linha 6 é criado

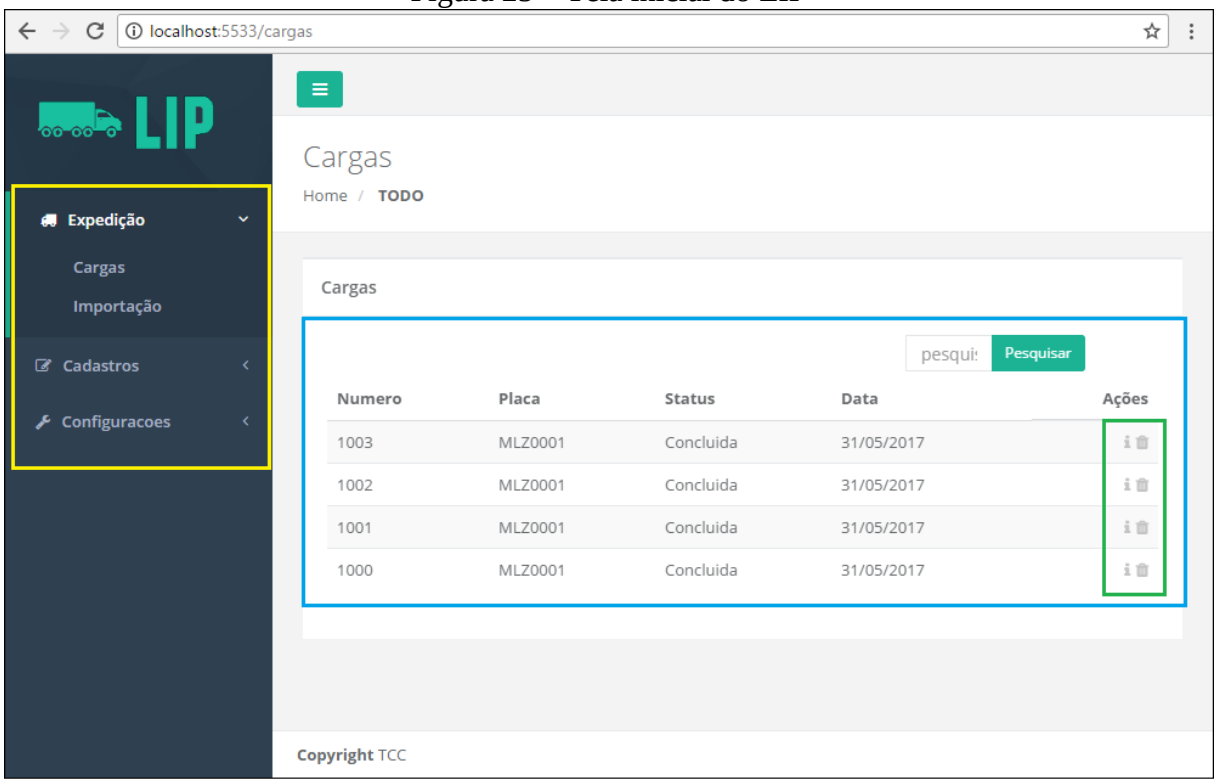
uma geometria do tipo caixa e na linha 7 é especificado o tipo da superfície que será aplicada. Por fim, da linha 10 até a linha 12 é especificada a coordenada do objeto e a linha 13 adiciona o objeto na cena.

3.3.2 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade da implementação, com as suas funcionalidades no sistema LIP. São demonstrados os resultados obtidos pelos algoritmos implementados, os cadastros e as configurações necessárias para o sistema.

A Figura 25 apresenta a tela inicial do sistema com as cargas processadas no dia (circulado em azul) e também o botão `Pesquisar` para buscar uma determinada carga. Ainda na tabela de cargas encontra-se as ações (circulado em verde) de `Excluir` e `Visualizar` uma carga. Na lateral esquerda (circulado em amarelo) é possível visualizar o menu de navegação para acessar os demais módulos do sistema.

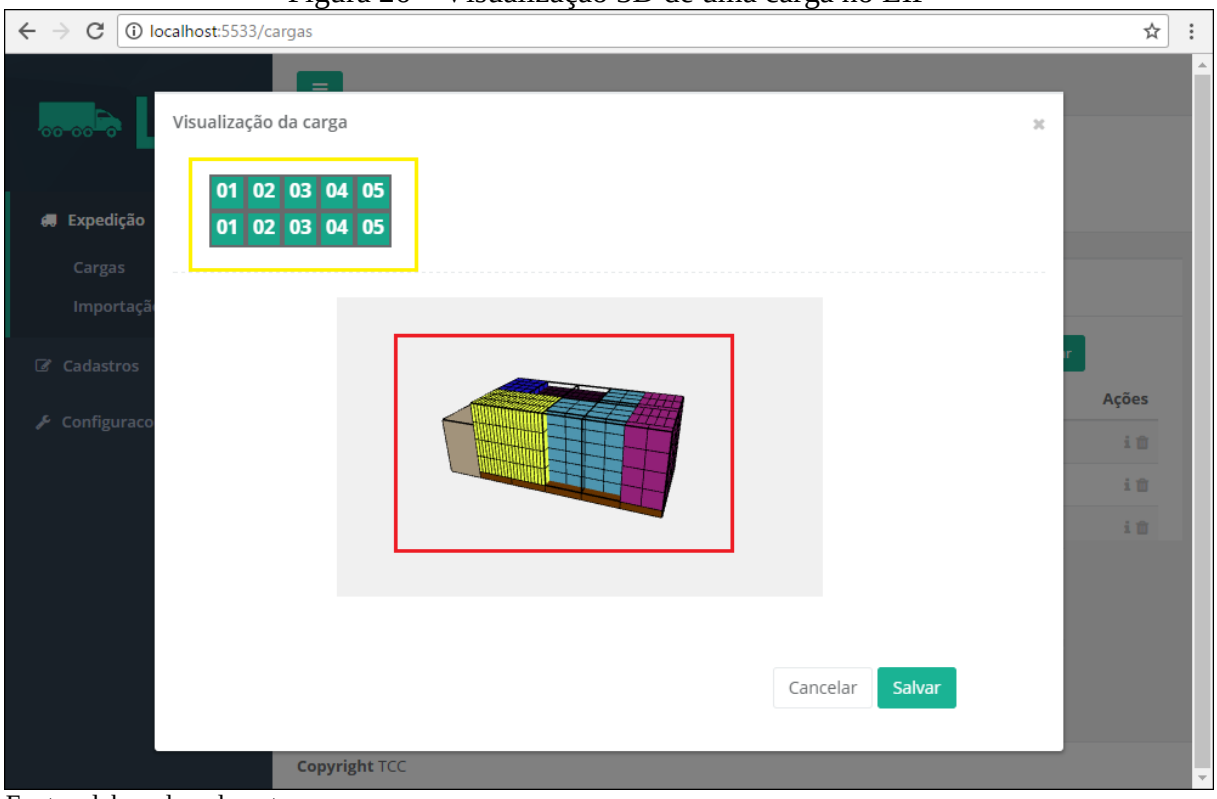
Figura 25 – Tela inicial do LIP



Fonte: elaborado pelo autor.

A Figura 26 apresenta a visualização de uma carga processada pelo sistema (circulado em vermelho). Cada palete pode ser visualizado individualmente pelo menu superior da tela (circulado em amarelo).

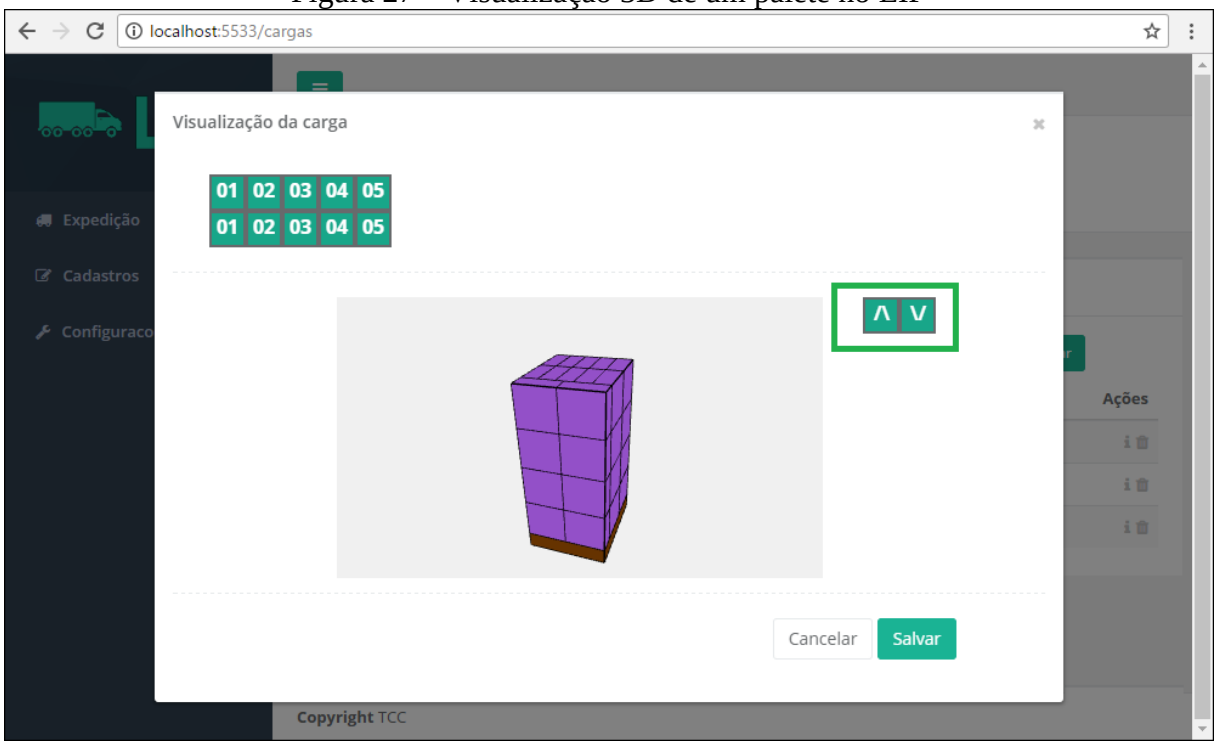
Figura 26 – Visualização 3D de uma carga no LIP



Fonte: elaborado pelo autor.

A Figura 27 mostra a visualização individual em um palete calculado pelo sistema e também as setas (circulado em verde) para visualizar a ordem de montagem do palete.

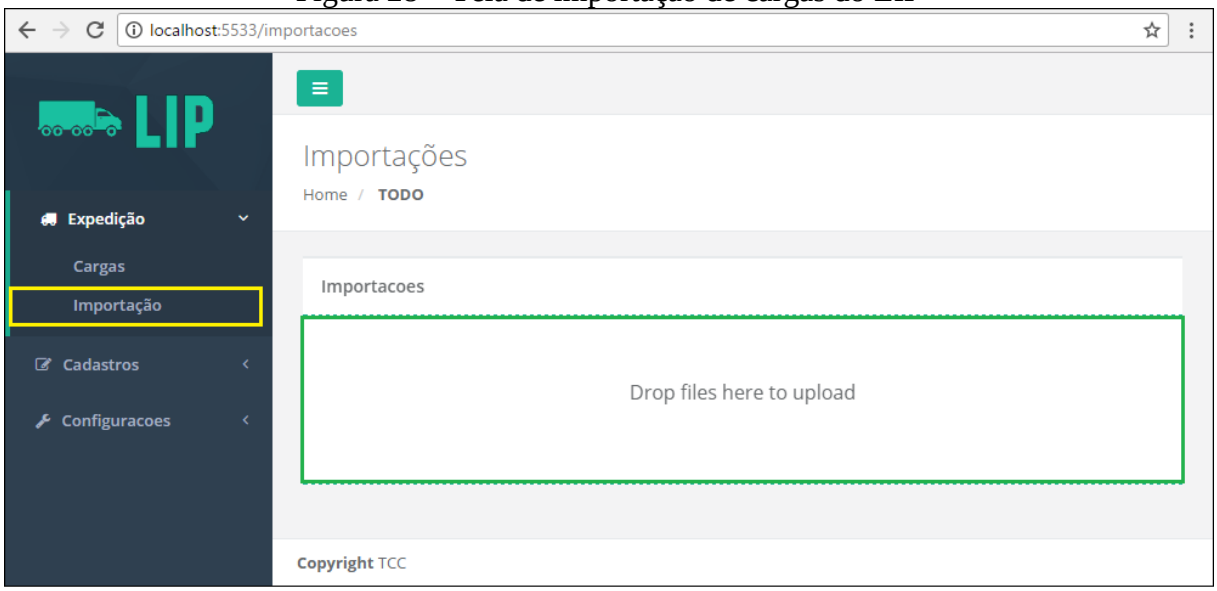
Figura 27 – Visualização 3D de um palete no LIP



Fonte: elaborado pelo autor.

A importação de uma nova carga para ser calculada pelo sistema é realizada pela tela de importação (Figura 28), que pode ser acessada pelo menu `Expedição > Importação` (circulado em amarelo), a importação da carga é realizada por um arquivo com o formato JSON que deve ser arrastado para a área de *upload* (circulado em verde).

Figura 28 – Tela de importação de cargas do LIP



Fonte: elaborado pelo autor.

No Quadro 19 é apresentado um exemplo de um arquivo de importação para o sistema. Na linha 2 é informado o número da carga e na linha 3 é informado a placa do veículo que a carga será carregada. Na linha 4 são informadas as entregas que estão contidas na carga, com a sua ordem e pedidos.

Quadro 19 – Exemplo de um arquivo de importação

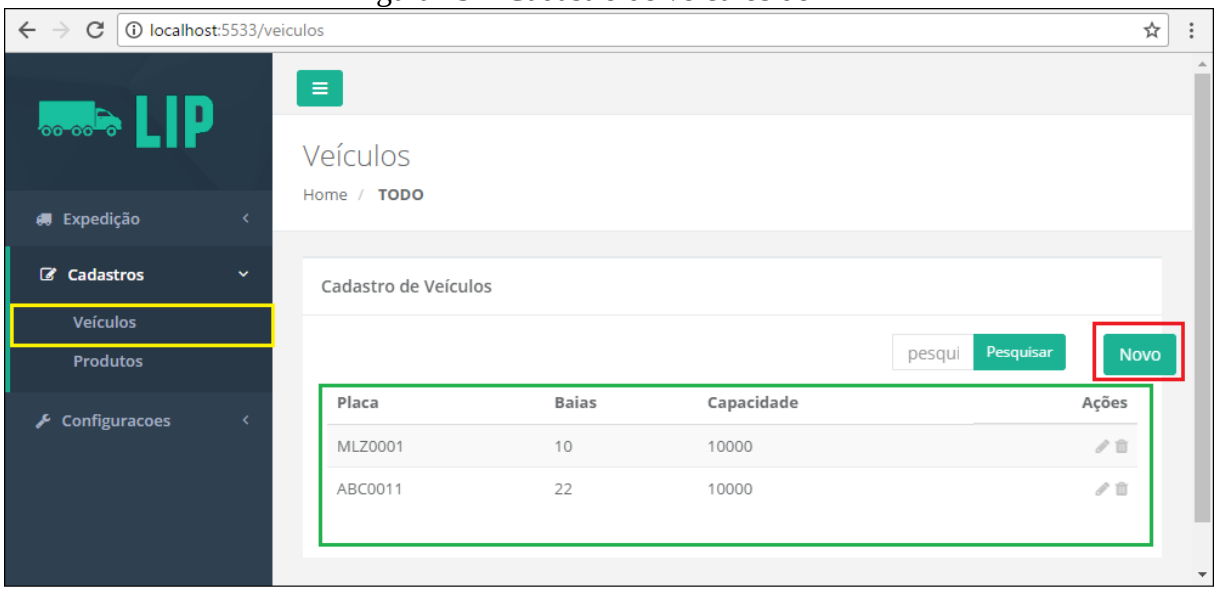
```
01 {
02   "Numero": 1002,
03   "Placa": "MLZ0001",
04   "Entregas": [{
05     "OrdemEntrega":1,
06     "Pedidos":[{
07       "CodigoProduto":"111",
08       "QuantidadeCaixas":96
09   }]
10   },{
11     "OrdemEntrega":2,
12     "Pedidos":[{
13       "CodigoProduto":"222",
14       "QuantidadeCaixas":126
15       }]
16   }]
17 }
```

Fonte: elaborado pelo autor.

Na Figura 29 é apresentado o cadastro de veículos, esta tela pode ser acessada pelo menu `Cadastros > Veículos` (circulado em amarelo). No centro da tela (circulado em

verde), pode ser visualizado os veículos cadastrados no sistema e na lateral direita (circulado em vermelho) pode ser visualizado o botão `Novo`, para criação de um novo veículo no sistema.

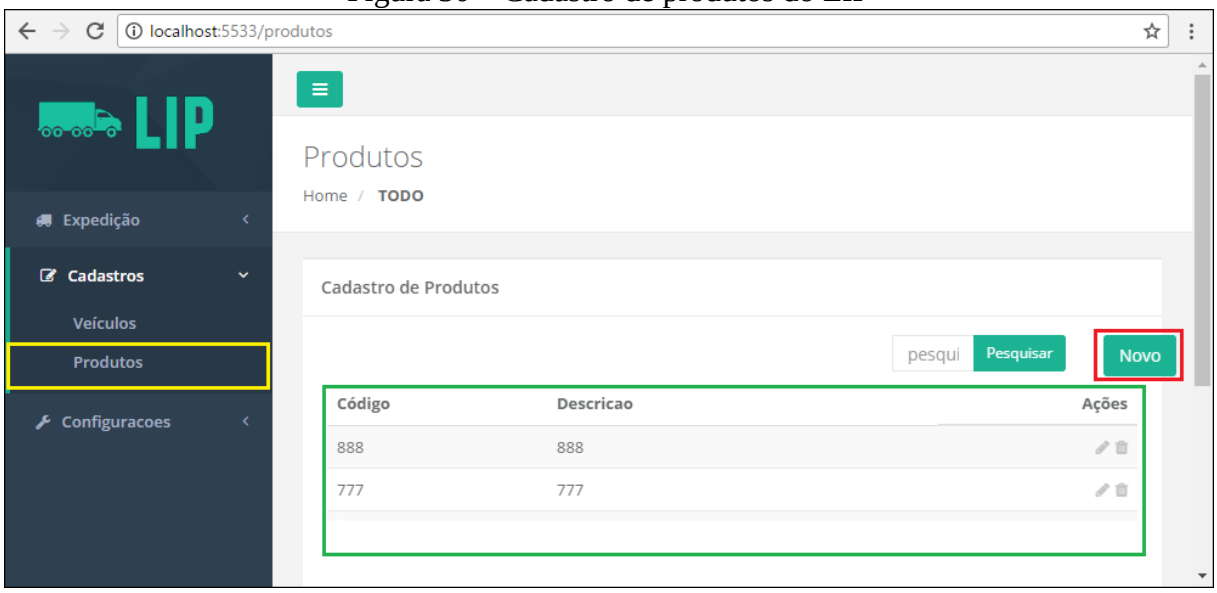
Figura 29 – Cadastro de veículos do LIP



Fonte: elaborado pelo autor.

Na Figura 30 é apresentado o cadastro de produtos, esta tela pode ser acessada pelo menu `Cadastros` > `Produtos` (circulado em amarelo). No centro da tela (circulado em verde), pode ser visualizado os produtos cadastrados no sistema e na lateral direita (circulado em vermelho) pode ser visualizado o botão `Novo`, para criação de um novo produto no sistema.

Figura 30 – Cadastro de produtos do LIP

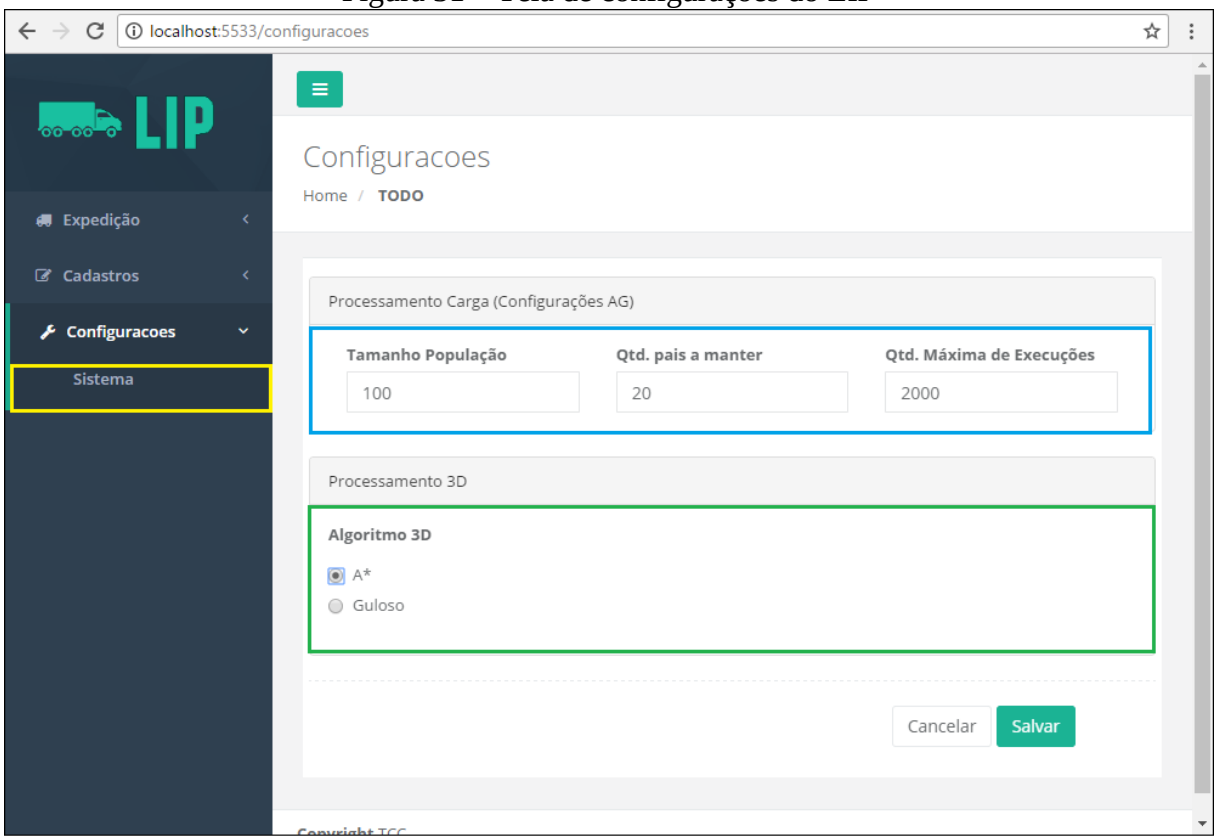


Fonte: elaborado pelo autor.

Na Figura 31 é apresentado o cadastro das configurações do sistema, esta tela pode ser acessada pelo menu `Configurações` > `Sistema` (circulado em amarelo). Nesta tela é possível

configurar os parâmetros utilizado pelo AG (`Tamanho da População`, `Quantidade pais a manter` e `Quantidade máxima de execuções`) e também o algoritmo que será utilizado para o cálculo 3D o palete (`A*` e `busca local`).

Figura 31 – Tela de configurações do LIP



Fonte: elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

Este trabalho permite encontrar a melhor organização de produtos em um veículo utilizando técnicas aproximadas no processamento dos pedidos. Com base nos testes realizados é possível identificar a viabilidade de utilizar essas técnicas em pequenos problemas.

Na seção 3.4.1 são apresentados os experimentos realizados, na seção 3.4.2 são apresentados os resultados do processamento de pedidos realizado pelo AG. Na seção 3.4.3 são apresentados os resultados do cálculo 3D realizado pelo algoritmo de busca local e pelo A*. Por fim, na seção 3.4.4 é feito uma comparação do sistema desenvolvido com os trabalhos correlatos.

3.4.1 Resultados e classificações dos dados nos testes realizados

Um grande número de experimentos foi realizado para avaliar os algoritmos propostos. As amostras foram executadas em um computador com processador Core i5 2.50 GHz com

8Gb de RAM e o sistema operacional Windows. Os testes foram submetidos ao AG, responsável por realizar o processamento dos pedidos, para posteriormente serem submetidos aos algoritmos de busca local e A* para realizar o cálculo 3D dos paletes. As seguintes classes de problemas foram consideradas:

a) classe 1: os produtos são altos e compridos;
b) classe 2: os produtos são largos e compridos;
c) classe 3: os produtos são altos e largos;
d) classe 4: os produtos têm grandes dimensões;
e) classe 5: os produtos têm pequenas dimensões;
f) classe 6: problemas do distribuidor obtidos de um cenário real;
g) classe 7: problemas do produtor de um cenário real.

Como pode ser observado na Tabela 1, para cada classe de 1 à 5 foram gerados aleatoriamente 300 cenários com a quantidade de entregas variando de 1 à 20 e o percentual de ocupação variando de 75 à 100%. Para a classe 6 foram obtidos 850 cenários reais de um centro de distribuição de bebidas referente a 1 semana de operação, nestes cenários além do percentual de ocupação e a quantidade de entregas, a quantidade de baias dos veículos variou de 4 à 22. Na classe 7 foram obtidos 10 cenários para o problema do produtor, cada cenário dessa classe é representado por 1 único pedido contendo apenas 1 produto.

Tabela 1 – Cenários de teste

Classe	Quantidade de cenários	Quantidade média de produtos	% médio de ocupação da carga
1	300	492	86
2	300	469	87
3	300	460	86
4	300	228	88
5	300	4216	86
6	850	536	54
7	10	835	100
	2360	1033	72

Fonte: elaborado pelo autor.

3.4.2 Resultado do processamento dos pedidos pelo AG

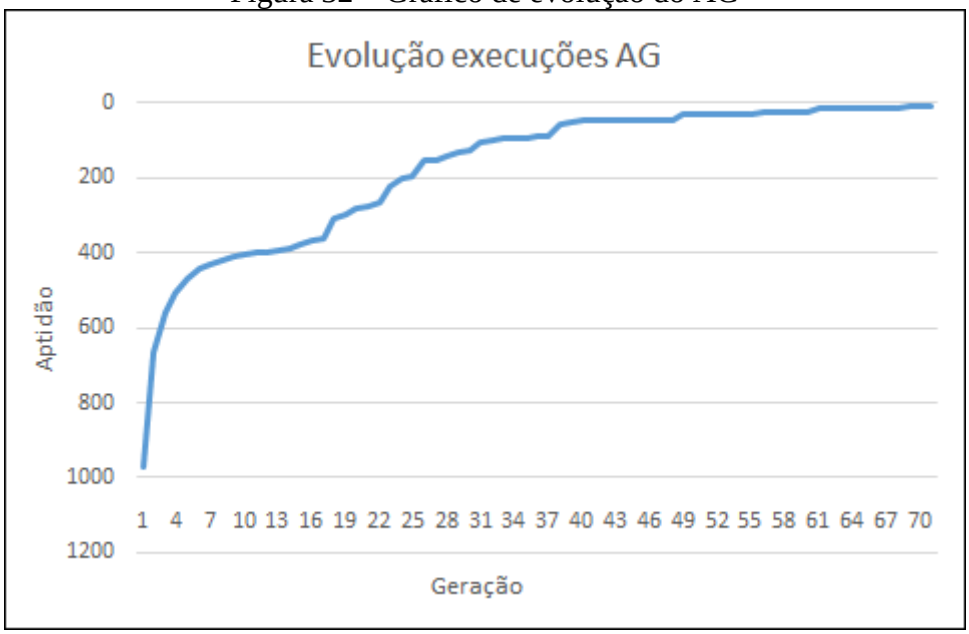
Para encontrar a melhor configuração de execução do AG, foram realizados testes preliminares considerando 36 combinações dos parâmetros (taxa de mutação, tamanho da população e quantidade de cromossomos pais a manter). A melhor configuração obtida considera uma taxa de mutação de 1% com uma população máxima de 300 cromossomos e mantendo 10 cromossomos pais a cada geração, o resultado dos demais cenários podem ser visualizados no Apêndice B deste trabalho. A quantidade de máxima de gerações foi limitada

a 200, ou caso o mesmo execute 15 gerações seguidas sem nenhuma evolução da aptidão ter sido alcançada.

O AG encontrou a solução (volume, estabilidade da carga e ordem de entrega) para 2266 dos 2360 cenários de teste. Isso representou 96% dos dados utilizados para os testes, sendo que 1842 destes cenários a solução foi encontrada em menos de 1 segundo.

A Figura 32 apresenta o gráfico da evolução da aptidão diante das gerações realizadas pelo AG, estes dados foram extraídos a partir da média da aptidão de todas as execuções realizadas para os cenários. O eixo vertical traz a aptidão e o eixo horizontal apresenta as gerações geradas pelo AG.

Figura 32 – Gráfico de evolução do AG



Fonte: elaborado pelo autor.

A Tabela 2 traz o resultado do processamento dos pedidos realizado por AG separado por sua classe de teste. Podemos observar os pedidos da classe 5 tiveram seu tempo de processamento muito superior aos demais cenários de testes, isto aconteceu pela quantidade elevada de produtos presente nos pedidos, que fez com que os operadores genéticos tivessem um tempo relativamente alto de execução.

Tabela 2 – Resultado processamento dos pedidos

Classe	Quantidade média de produtos	Tempo médio de processamento em milissegundos	% de processamentos com sucesso
1	492	817	100
2	469	586	100
3	460	710	100
4	228	283	100
5	4216	50174	97
6	536	5593	91
7	835	483	100
	1033	8378	98

Fonte: elaborado pelo autor.

O sistema conseguiu processar com 100% de sucesso 5 das 7 classes de teste. A classe 6 possuiu a menor taxa de sucesso, visto que, o algoritmo conseguiu encontrar a solução para 91% dos cenários. Analisando os pedidos foi possível verificar que 31 dos cenários obtidos continham problemas de cadastros. O Quadro 20 apresenta como exemplo um pedido não processado pelo AG.

Quadro 20 – Pedido não processado pelo AG

Número da carga: 270734

Placa do veículo: CVV6836

Quantidade de baias: 4

Volume solicitado: 15,5m³

Volume disponível: 9,6m³

Fonte: elaborado pelo autor.

É possível observar que o pedido `270734` solicitou o processamento de um volume de `15,5m³` para o veículo `CVV6836`, este veículo possui apenas `4` baias com um volume total de `9,6m³` disponível para carregamento. Seria necessário um veículo com no mínimo o dobro da capacidade para realizar o processamento do pedido solicitado.

3.4.3 Resultado do processamento dos paletes

A Tabela 3 traz uma comparação do algoritmo de busca local com o algoritmo A* desenvolvidos para solucionar o PLP. Para encontrar a melhor configuração de peso das heurísticas do A*, foram realizados teste preliminares utilizando 32 combinações das heurísticas. A melhor combinação obtida considera `h1` com peso 15, h2 com peso 3, h3 com peso 15 e h4 com peso 0,25. O resultado dos testes de todas as combinações pode ser visualizado no Apêndice C deste trabalho. Também foi necessário limitar o tempo de

execução do A* em 40 segundos por palete, tendo em vista que para os cenários mais complexos seria inviável encontrar uma solução em um tempo computacional aceitável. Todos os cenários foram submetidos aos algoritmos, para os cenários que o AG não encontrou uma solução, foram utilizados os resultados aproximados obtidos. Estes resultados estão separados pela sua classe, como pode ser visualizado na primeira coluna. A segunda coluna traz o percentual médio da ocupação dos cenários no veículo. A terceira e quarta coluna traz o percentual de ocupação que o algoritmo de busca local e A* conseguiram atingir respectivamente. A quinta e sexta coluna traz o tempo médio de processamento do algoritmo de busca local e A*.

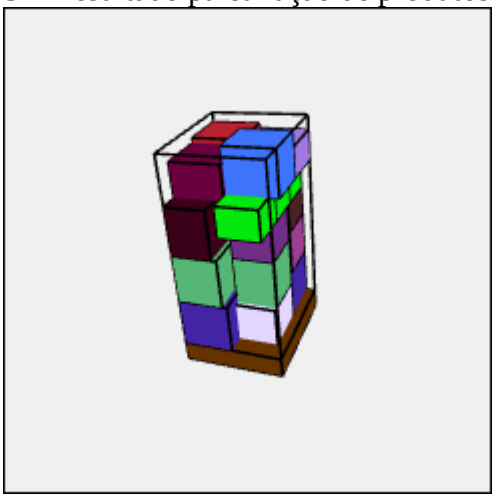
Tabela 3 – Comparação da busca local com o algoritmo A*

Classe	% médio de ocupação da carga	% médio de ocupação atingido (busca local)	% médio de ocupação atingido (A*)	Tempo médio de processamento em milissegundos (busca local)	Tempo médio de processamento em milissegundos (A*)
1	86	80	82	277	73608
2	87	76	76	239	182064
3	86	77	79	228	195498
4	88	73	75	109	184561
5	86	85	20	49256	252725
6	54	50	51	6176	34965
7	100	93	100	128	48394
	83	76	69	8059	138830

Fonte: elaborado pelo autor.

Houve problemas ao realizar o cálculo 3D dos paletes das classes 2, 3 e 4 em ambos os algoritmos, onde os pedidos não tiveram 100% dos seus produtos alocados. Este problema foi gerado principalmente por causa dos produtos serem largos e compridos, pois devido à grande dimensão dos produtos, ocorre uma grande perda de volume durante a alocação. Um exemplo dessa situação está na Figura 33, onde pode ser visualizado que o palete dispõe de um grande volume disponível, porém não há mais locais possíveis para alocação de um produto com grandes dimensões.

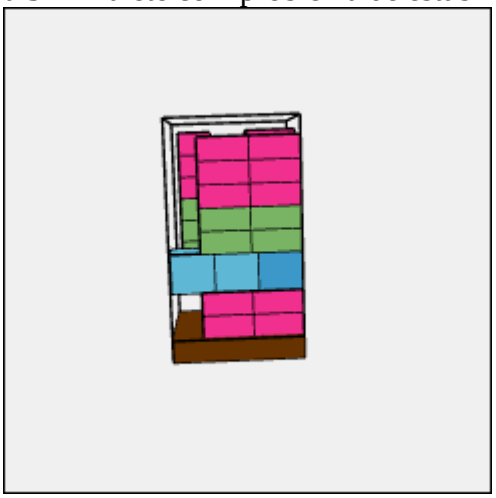
Figura 33 – Resultado paletização de produtos classe 4



Fonte: elaborado pelo autor.

Alguns paletes montados pelo algoritmo de busca local também apresentaram problemas de instabilidade dos produtos. Este problema foi gerado pois nenhuma heurística considerando estas condições foi utilizada. A Figura 34 apresenta um exemplo de um palete montado errado pelo algoritmo, é possível verificar que a caixa em azul possui grande parte de sua área suspensa.

Figura 34 – Palete com problema de estabilidade

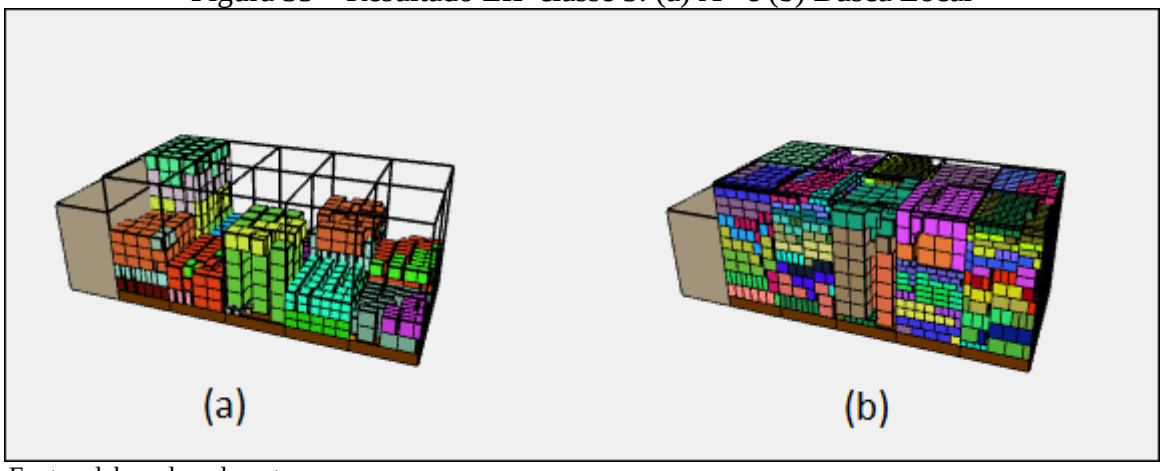


Fonte: elaborado pelo autor.

Para a classe 5, o algoritmo A* apresentou uma baixa performance. Este problema foi causado pelo grande número de produtos contidos nos pedidos dessa classe, gerando um grande número de estados possíveis, o que tornou inviável encontrar uma solução em um tempo computacional aceitável. Em contrapartida, o algoritmo de busca local apresentou bons resultados, muito deles próximos do ideal. Porém com um alto tempo de processamento, para estes cenários o algoritmo de busca local demorou em média 49 segundos para encontrar uma solução. A Figura 35 mostra a diferença do resultado obtido pela execução do algoritmo A* e

pela busca local para um mesmo exemplo da classe 5. A Figura 35a apresenta o resultado do algoritmo A* e a Figura 35b apresenta o resultado obtido pela busca local.

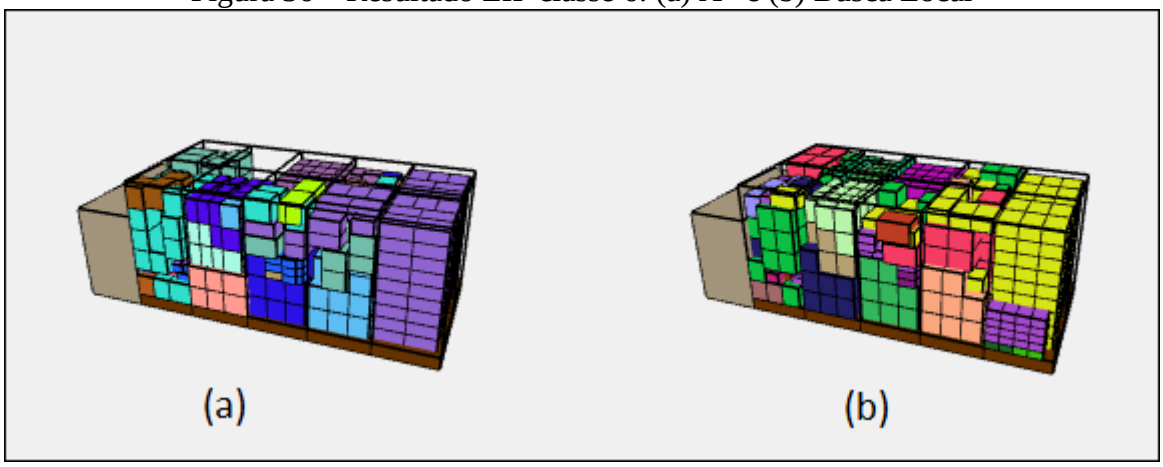
Figura 35 – Resultado LIP classe 5: (a) A* e (b) Busca Local



Fonte: elaborado pelo autor.

Para os cenários da classe 6 os dois algoritmos apresentaram bons resultados, sendo que o algoritmo A* encontrou a solução de 565 dos cenários e o algoritmo de busca local encontrou a solução de 489 destes cenários. Estes resultados tão expressivos se dão também pelo fato da maioria dos pedidos processados utilizarem em média apenas 50% do volume disponível no veículo. Na Figura 36a pode ser visualizado o resultado do algoritmo A* onde foi atingido 77% da ocupação do veículo e na imagem Figura 36b o resultado do algoritmo de busca local onde foi atingido 86% da ocupação do veículo.

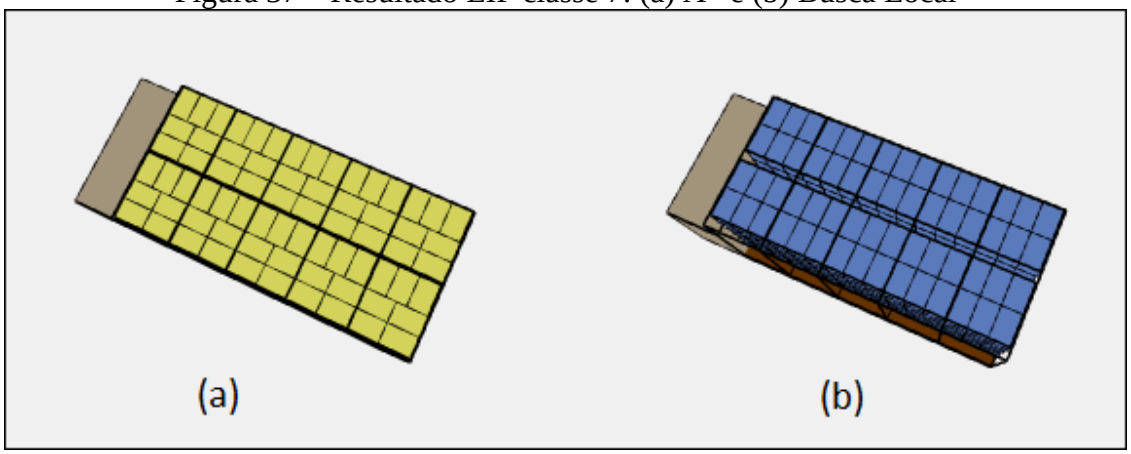
Figura 36 – Resultado LIP classe 6: (a) A* e (b) Busca Local



Fonte: elaborado pelo autor.

O algoritmo A* mostrou-se eficiente para solucionar o MPLP, representados pela classe 7, o algoritmo conseguiu resolver 100% dos cenários propostos. A Figura 37a mostra a o resultado obtido pela execução do algoritmo A* e a Figura 37b mostra o resultado obtido pela busca local na Figura 37b para a um mesmo exemplo da classe 7.

Figura 37 – Resultado LIP classe 7: (a) A* e (b) Busca Local



Fonte: elaborado pelo autor.

3.4.4 Comparação de trabalhos correlatos e o sistema desenvolvido

No Quadro 21 é feita uma comparação dos trabalhos correlatos apresentados na seção 2.4 com o sistema desenvolvido. Os trabalhos que possuírem as características descritas na primeira coluna estão marcados com "X".

Quadro 21 – Comparativo entre os trabalhos correlatos e o sistema desenvolvido

	Gonçalves e Resende (2012)	Cavalcanti Júnior (2009)	Silva e Soma (2002)	Sistema Desenvolvido (2017)
Técnica utilizada	AG	AG	AG	AG
Tamanhos variados	X		X	X
Estabilidade dos produtos			X	
Ordem de entrega				X
Técnica de cálculo	Volume	Área	Volume	Volume
Aplicação	Contâiner	Palete	Contâiner	Caminhão
Visualização do resultado		2D		3D

Fonte: elaborado pelo autor.

Conforme o Quadro 21 apresenta, é possível notar que todos os trabalhos utilizam uma técnica heurística para a solução do problema e permitem a utilização de caixas com tamanhos variados, exceto o trabalho de Cavalcanti Júnior (2009) que permite apenas caixas com tamanhos idênticos. O trabalho de Silva e Soma (2002) é o único que trava o problema de estabilidade dos produtos dentro do palete e nenhum dos trabalhos correlatos trata a ordem de entrega dos produtos. O cálculo por volume está presente em todos os trabalhos, exceto pelo trabalho de Cavalcanti Júnior (2009), que resolve o problema apenas de forma bidimensional. A aplicação em containers está presente na maioria das técnicas apresentadas, exceto por Cavalcanti Júnior (2009) onde o sistema é utilizado para paletes. E por fim, a visualização gráfica do resultado está presente apenas no trabalho de Cavalcanti Júnior

(2009), esta visualização ajuda no processo de montagem da solução encontrada para o problema.

4 CONCLUSÕES

Uma das principais missões da tecnologia é tornar a vida do homem mais fácil. As pessoas gastam muito tempo resolvendo problemas cotidianos, que podem ser delegados a um sistema. Este trabalho buscou oferecer uma forma de auxilio no processo de montagem de pedidos em um veículo, resolvendo problemas como volume, estabilidade da carga e ordem de entrega. Foi criado um sistema chamado LIP, que pode futuramente gerar um produto que auxilie as transportadoras no processo de paletização, reduzindo assim, custos envolvidos nesse processo, assim como no transporte e armazenagem de mercadorias.

A aplicação foi desenvolvida utilizando a linguagem C# no ambiente Visual Studio 2015 e a linguagem Java no ambiente Eclipse Neon 3.0. Todas as ferramentas se mostraram funcionais, não apresentando problemas durante o desenvolvimento do trabalho.

Uma das principais dificuldades encontradas neste trabalho foi o entendimento de alguns artigos científicos estudados. Algumas formulações matemáticas contidas nos artigos eram de difícil interpretação e fundamentais para a implementação das heurísticas presentes em alguns algoritmos.

Este trabalho pode ser o ponto de partida para outras aplicações com problemas relacionados, tais como: carregamento de containers, problemas de estabilidade da carga, entre outros problemas da área da logística.

A biblioteca three.js funcionou corretamente para a visualização dos paletes e também operar na ordem de montagem do mesmo. Entretanto, um ponto interessante a ser aprimorado, seria disponibilizar esta visualização em um dispositivo móvel, de modo que facilite a mobilidade do usuário durante o processo de montagem do palete.

A partir dos resultados dos experimentos realizados, pode-se concluir que a aplicação desenvolvida apesar das limitações, é uma opção interessante para resolver o MPLP. Para o DPLP os testes apontam que são necessárias algumas melhorias para se tornar mais completa e confiável. Em geral, o sistema possui boa usabilidade, porém para ser utilizado em um cenário real devem ser tratados os problemas de estabilidade dos produtos dentro do palete.

Em relação aos trabalhos correlatos, percebe-se uma preocupação em desenvolver soluções para os problemas de empacotamento, porém poucos apresentam uma forma de visualização eficiente do resultado para usuários finais. Para concluir, outro ponto interessante foi a visualização 3D do resultado obtido pelo sistema. Este tipo de visualização auxilia no processo de montagem dos pedidos para serem dispostos dentro do veículo, desta forma, facilitando a vida da pessoa responsável por realizar esta atividade.

4.1 EXTENSÕES

Algumas possíveis extensões para este trabalho são:

a) desenvolver uma nova heurística para garantir a estabilidade dos produtos dentro do palete;
b) desenvolver uma nova heurística no AG para tratar problemas de resistência dos produtos;
c) otimizar o algoritmo A* para resolução de problemas com itens pequenos;
d) adaptar a visualização do resultado para dispositivos móveis;
e) alterar o AG para tratar também o problema de empacotamento.

REFERÊNCIAS

ALEGRE, Alexander Rivera. **Método heurístico para escolha do sistema de picking de um operador logístico:** um Estudo de Caso. 2005. 97 f. Dissertação (Mestrado) - Curso de Engenharia Mecânica, Faculdade de Engenharia Mecânica, Campinas, 2005.

ALMEIDA, C. F. M.; KAGAN, N. Allocation of power quality meters by genetic algorithms and Fuzzy sets theory. Sba: **Controle & Automação Sociedade Brasileira de Automatica**, v. 21, n. 4, p. 363-378, 2010.

BALLOU, Ronald H.. **Logística empresarial.** São Paulo: Atlas, 1993.

BALLOU, Ronald H.. **Gerenciamento da cadeia de suprimentos:** logística empresarial. 5. ed. Porto Alegre: Bookman, 2006.

BISCHOFF, Eberhard E.; MARRIOTT, Michael D. A comparative evaluation of heuristics for container loading. **European Journal of Operational Research**, v. 44, n. 2, p. 267-276, 1990.

BORDINI, Camile Frazão. **Técnicas probabilísticas aplicadas em algoritmos de aproximação.** 2016. 94 f. Monografia (Especialização) - Curso de Pós-graduação em Informática, Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2016.

BOWERSOX, Donald J.; CLOSS, David J.. **Logística empresarial:** o processo de integração da cadeia de suprimentos. São Paulo: Atlas, 2001.

BRACHT, Evandro Cesar. **Algoritmos de aproximação para o problema de classificação métrica.** 2004. 100 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Estadual de Campinas, Campinas, 2004. Disponível em: <http://www.ic.unicamp.br/~evandro/tese.pdf>. Acesso em: 07 maio 2017.

BUENO, Fabrício. **Métodos heurísticos:** Teoria e Implementações. 2009. Disponível em: <https://wiki.ifsc.edu.br/mediawiki/images/b/b7/Tutorial_métodos_heurísticos.pdf>. Acesso em: 30 out. 2016.

CAVALCANTI JÚNIOR, George Moraes. **SIP – Sistema Inteligente de Carregamento de Paletes.** 2009. 56 f. TCC (Graduação) - Curso de Engenharia da Computação, Escola Politécnica de Pernambuco, Recife, 2009. Disponível em: <http://tcc.ecomp.poli.br/20091/TCC_George_Moraes_SIP.pdf>. Acesso em: 22 ago. 2016.

CECCILIO, Fabiana Oliveira; MORABITO, Reinaldo. Refinamentos na heurística de George e Robinson para o problema do carregamento de caixas dentro de contêineres. **Transportes**, v. 12, n. 1, 2004.

COELIS, Elenilce Lopes. **Logística empresarial.** 2008. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/507>. Acesso em: 27 set. 2016.

CORMEN, Thomas H. **Introduction to algorithms**. MIT press, 2009.

DIRENE, Alexandre Ibrahim. **Algoritmos de busca heurística.** 2016. Disponível em: <http://www.inf.ufpr.br/alexd/abh/abh1.pdf>. Acesso em: 30 ago. 2016.

ESTEVES, Yohans de Oliveira; ALVES, Cláudio da Silva. **Sistema de picking by voice na cadeia logística:** O caso da empresa Anglo do Brasil/RJ. 2013. Disponível em: <http://www.inovarse.org/filebrowser/download/15630>. Acesso em: 15 mar. 2017.

FINGLER, Henrique. **Otimização de colônias de formigas em cuda:** O problema da mochila e o problema quadrático de alocação. 2013. 67 f. TCC (Graduação) - Curso de Computação, Universidade Federal do Mato Grosso do Sul, Campo Grande, 2013. Disponível em: <https://sistemas.ufms.br/sigpos/portal/trabalhos/download/861/cursoId:29>. Acesso em: 08 maio 2017.

GONÇALVES, José Fernando; RESENDE, Mauricio GC. A parallel multi-population biased random-key genetic algorithm for a container loading problem. **Computers & Operations Research**, v. 39, n. 2, p. 179-190, 2012.

KANN, Viggo. **On the approximability of NP-complete optimization problems.** 1992. 168 f. Tese (Doutorado) - Curso de Computing Science, Department Of Numerical Analysis And Computing Science, Royal Institute Of Technology Stockholm, Stockholm, 1992.

LEITÃO, Rafael Pisciottano. **Atividade de picking, com estudo de caso da indústria de cigarros Souza Cruz S/A.** 2007. 64 f. TCC (Graduação) - Curso de Tecnólogo em Logística, Faculdade de Tecnologia da Baixada Santista, Santos, 2007.

LIMA, Chantele Cerqueira de et al. **Aplicação de IA no auxílio a tomada de decisão.** 2008. Disponível em: <http://gsigma.ufsc.br/~popov/aulas/icpg/20081/IA_na_Tomada_de_Decisao_Artigo.pdf>. Acesso em: 15 mar. 2017.

LINDEN, Ricardo. **Algoritmos Genéticos:** Uma importante ferramenta da Inteligência Computacional. Rio de Janeiro: Brasport, 2006. 348 p.

LUGER, George F. **Inteligência Artificial:** Estruturas e estratégias para a solução de problemas complexos. Bookman, 2004.

MACHADO, Raphaela Carvalho; ALVES, Helton do Nascimento. Um algoritmo genético para localização de faltas em redes aéreas radiais de distribuição de energia elétrica. In: CONGRESSO NORTE-NORDESTE DE PESQUISA E INOVAÇÃO, 5, 2010, Maceió. **Anais eletrônicos.** Maceió: IFAL, 2010. Disponível em: < http://congressos.ifal.edu.br/index.php/connepi/CONNEPI2010/paper/viewFile/1924/564>. Acesso em: 23 abr. 2017.

MARQUES, Robert. **História da Logística.** 2008. Disponível em: <http://www.administradores.com.br/artigos/marketing/historia-da-logistica/24829/>. Acesso em: 10 set. 2016.

MENDONÇA, Juliana Karim de. **Inteligência nos Negócios**: Logística faz a diferença. 2013. Disponível em: <http://docplayer.com.br/107848-Inteligencia-nos-negocios-logistica-faz-a-diferenca.html>. Acesso em: 23 abr. 2017.

MORALES, Silvia Regina; MORABITO, Reinaldo. A simple and effective heuristic to solve the manufacturing pallet loading problem. **Gestão & Produção**, v. 4, n. 1, p. 52-76, 1997.

MOURA, Reinaldo A.. **Manual de logística:** Armazenagem e distribuição física. 2. ed. São Paulo: Imam, 1998. 343 p.

OLIVEIRA, José Fernando; WÄSCHER, Gerhard. Cutting and packing. **European journal of operational research**, v. 183, n. 3, p. 1106-1008, 2007.

ZIVIANI, Nivio. **Projeto de Algoritmos.** Minas Gerais: Cengage Learning, 2006.

PEDRUZZI, Suzane et al. A mathematical model to optimize the volumetric capacity of trucks utilized in the transport of food products. **Gestão & Produção**, v. 23, n. 2, p. 350-364, 2016.

POZO, Aurora et al. **Computação Evolutiva.** 2002. Disponível em: <http://www.inf.ufpr.br/aurora/tutoriais/Ceapostila.pdf>. Acesso em: 30 out. 2016.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência artificial.** 3. ed. New Jersey: Pearson Education, 2013

SILVA, José Lassance de Castro; SOMA, Nei Yoshihiro. **Um algoritmo genético aplicado ao problema de empacotamento de bins tridimensionais.** 2002. Disponível em: <http://www.repositorio.ufc.br/bitstream/riufc/13832/1/2002_eve_jlcsilva.pdf>. Acesso em: 05 mar. 2017.

SILVEIRA, Jefferson Luiz Moisés da. **Algoritmos de Aproximação para Problemas de Empacotamento em Faixa com Restrições de Descarregamento.** 2011. 73 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Estadual de Campinas, Campinas, 2011.

WILLIAMSON, David P.; SHMOYS, David B. **The design of approximation algorithms**. Cambridge University Press, 2011.

WÄSCHER, Gerhard et al. An improved typology of cutting and packing problems. **European journal of operational research**, v. 183, n. 3, p. 1109-1130, 2007.

XAVIER, Eduardo Candido. **Algoritmos de aproximação para problemas de escalonamento em máquinas.** 2003. 105 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Estadual de Campinas, Campinas, 2003. Disponível em: <http://www.academia.edu/21766405/Algoritmos_de_Aproximac_a_o_para_Problemas_de_Escalonamento_em_Ma_quinas>. Acesso em: 07 maio 2017.

XAVIER, Carlos André de Araújo Frias. **Sistema de apoio à decisão para o problema de paletização no carregamento de aviões.** 2012. 63 f. Dissertação (Mestrado) - Curso de Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, Porto, 2012. Disponível em: <https://repositorio-aberto.up.pt/bitstream/10216/68378/1/000154301.pdf>. Acesso em: 15 mar. 2017.

ZAMBIASI, Saulo Popov. **Métodos de busca.** 2010. Disponível em: <http://www.gsigma.ufsc.br/~popov/aulas/ia/modulo3/>. Acesso em: 31 ago. 2016.

ZINI, Érico de Oliveira Costa. **Algoritmo genético especializado na resolução de problemas com variáveis contínuas e altamente restritos.** 2009. 151 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Estadual Paulista, Ilha Solteira - SP, 2009.

APÊNDICE A – Modelo de importação de pedidos para o sistema

Neste apêndice é apresentado no Quadro 22 um exemplo de um arquivo de importação contendo três pedidos.

Quadro 22 – Exemplo de pedido para importação no sistema

```
      {
          "Numero": "250000000003",
          "Placa": "MLZ0001",
          "Entregas": [{
                  "OrdemEntrega": 1,
                  "Pedidos": [
                   {"CodigoProduto": 55019, "QuantidadeCaixas": 14},
                   {"CodigoProduto": 55006, "QuantidadeCaixas": 11},
                   {"CodigoProduto": 55025, "QuantidadeCaixas": 21}
      ]
              }, {
                  "OrdemEntrega": 2,
                  "Pedidos": [
                   {"CodigoProduto": 55017, "QuantidadeCaixas": 30},
                   {"CodigoProduto": 55025, "QuantidadeCaixas": 16},
                   {"CodigoProduto": 55010, "QuantidadeCaixas": 15}
      ]
              }, {
                  "OrdemEntrega": 3,
                  "Pedidos": [
                   {"CodigoProduto": 55025, "QuantidadeCaixas": 70},
                   {"CodigoProduto": 55042, "QuantidadeCaixas": 50}
      ]
              }
          }]
}
```

Fonte: elaborado pelo autor.

APÊNDICE B – Resultado dos testes para configuração do AG

Neste apêndice é apresentado na Tabela 4 o resultado dos testes preliminares para encontrar a melhor configuração do AG. A coluna 1 apresenta a taxa de mutação utilizada. Nas 2 e 3 colunas são mostrados a quantidade de cromossomos pais a manter a cada geração e o tamanho da população respectivamente. Por fim, na coluna 4 é apresentado a média de gerações necessárias para o AG convergir para uma solução.

Tabela 4 – Resultados dos testes preliminares para configuração do AG

% de mutação	Quantidade de pais a manter	Tamanho da população	Média da quantidade de gerações necessárias
1	10	100	24
1	10	200	19
1	**10**	**300**	**14**
1	20	100	34
1	20	200	21
1	20	300	18
1	30	100	35
1	30	200	23
1	30	300	19
2	10	100	30
2	10	200	22
2	10	300	15
2	20	100	35
2	20	200	23
2	20	300	22
2	30	100	41
2	30	200	29
2	30	300	26
3	10	100	22
3	10	200	26
3	10	300	26
3	20	100	40
3	20	200	21
3	20	300	27
3	30	100	35
3	30	200	28
3	30	300	31
4	10	100	34
4	10	200	33
4	10	300	26
4	20	100	33
4	20	200	26
4	20	300	28
4	30	100	33
4	30	200	35
4	30	300	25

Fonte: elaborado pelo autor.

APÊNDICE C – Resultado dos testes para configuração do A*

Neste apêndice é apresentado na Tabela 5 o resultado dos testes preliminares para encontrar a melhor configuração do A*. Nas colunas de 1 a 4 são apresentados os pesos utilizados para as heurísticas `h1`, `h2`, `h3` e `h4` respectivamente. Por fim, na coluna 5 é apresentado a média de nós visitados para encontrar a solução dos testes realizados.

Tabela 5 – Resultados dos testes preliminares para configuração do A*

H1	H2	H3	H4	Quantidade média de nós visitados
15.0	1.0	15.0	0.0	152605
15.0	1.0	15.0	0.25	291
15.0	1.0	15.0	0.5	168988
15.0	1.0	15.0	0.75	187808
15.0	1.0	20.0	0.0	173593
15.0	1.0	20.0	0.25	289
15.0	1.0	20.0	0.5	178449
15.0	1.0	20.0	0.75	186316
15.0	3.0	15.0	0.0	163869
15.0	**3.0**	**15.0**	**0.25**	**88**
15.0	3.0	15.0	0.5	170228
15.0	3.0	15.0	0.75	194749
15.0	3.0	20.0	0.0	165545
15.0	3.0	20.0	0.25	291
15.0	3.0	20.0	0.5	167470
15.0	3.0	20.0	0.75	178028
20.0	1.0	15.0	0.0	170679
20.0	1.0	15.0	0.25	170408
20.0	1.0	15.0	0.5	190909
20.0	1.0	15.0	0.75	184002
20.0	1.0	20.0	0.0	182224
20.0	1.0	20.0	0.25	181763
20.0	1.0	20.0	0.5	167161
20.0	1.0	20.0	0.75	194831
20.0	3.0	15.0	0.0	174669
20.0	3.0	15.0	0.25	184885
20.0	3.0	15.0	0.5	178620
20.0	3.0	15.0	0.75	213003
20.0	3.0	20.0	0.0	178222
20.0	3.0	20.0	0.25	170373
20.0	3.0	20.0	0.5	159455
20.0	3.0	20.0	0.75	182646

Fonte: elaborado pelo autor.