O presente trabalho aborda o problema de armazenagem industrial através de algoritmos de inteligência artificial conhecidas como técnicas de busca. É descrita a dinâmica do sistema ASRS (*Automated Storage and Retrieval System*) referentes à inserção, organização e retiradas de peças levando em consideração heurísticas relacionadas ao custo de movimento e à organização de peças no armazém. Os resultados pertinentes aos algoritmos das técnicas de buscas aplicadas no sistema ASRS oferecem dados que justificam o uso de inteligência artificial no sistema de armazenagem industrial.

Orientador: Douglas Wildgrube Bertol

JOINVILLE, 2019

2019

RHAVI GONÇALVES DE BORBA | AUTOMAÇÃO DE ARMAZÉM INDUSTRIAL SOB A PERSPECTIVA DA INDÚSTRIA 4.0

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
PROGRAMA DE PÓS GRADUAÇÃO PROFISSIONAL EM ENGENHARIA ELÉTRICA

DISSERTAÇÃO DE MESTRADO

AUTOMAÇÃO DE ARMAZÉM INDUSTRIAL SOB A PERSPECTIVA DA INDÚSTRIA 4.0

RHAVI GONÇALVES DE BORBA

JOINVILLE, 2019

RHAVI GONÇALVES DE BORBA

AUTOMAÇÃO DE ARMAZÉM INDUSTRIAL SOB A PERSPECTIVA DA INDÚSTRIA 4.0

Dissertação apresentada ao Curso de Pós-Graduação *strictu sensu* em Engenharia Elétrica, da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Douglas Wildgrube Bertol

JOINVILLE – SC
2019

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Setorial do CCT/UDESC,

com os dados fornecidos pelo(a) autor(a)

Borba, Rhavi
AUTOMAÇÃO DE ARMAZÉM INDUSTRIAL SOB A PERSPECTIVA DA INDÚSTRIA 4.0 / Rhavi Borba. -- 2019.
106 p.

Orientador: Douglas Wildgrube Bertol
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação Profissional em Engenharia Elétrica, Joinville, 2019.

1. Automação. 2. Indústria 4.0. 3. Armazém automático. 4. Inteligência artificial. 5. Automação da manufatura. I. Wildgrube Bertol, Douglas. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação Profissional em Engenharia Elétrica. III. Título.

Automação de Armazém Industrial na Perspectiva da Indústria 4.0

por

Rhavi Gonçalves de Borba

Esta dissertação foi julgada adequada para obtenção do título de

MESTRE EM ENGENHARIA ELÉTRICA

Área de concentração em "Automação de Sistemas"
e aprovada em sua forma final pelo

CURSO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA
DO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DO ESTADO DE SANTA CATARINA.

Banca Examinadora:

Prof. Dr. Douglas Wildgrube Bertol
CCT/UDESC
(Orientador/Presidente)

Dr. Leandro Israel Pinto
IOTBRAS LTDA

Prof. Dr. Ademir Nied
CCT/UDESC

Joinville, SC, 18 de dezembro de 2019.

AGRADECIMENTOS

Agradeço a todos que contribuíram para a realização deste trabalho e me apoiaram na minha trajetória acadêmica e profissional.

Em especial, gostaria de agradecer aos meus pais pelo amor e incentivo ao estudo desde que iniciei minha graduação na UDESC.

À minha esposa Debora Luiza Motta pelo amor e carinho.

Gostaria de agradecer também ao professor orientador Dr. Douglas Wildgrube Bertol pela paciência e confiança no meu trabalho.

À ciência logosófica que me possibilitou conhecer a mim mesmo e me nutriu de elementos suficientes para realizar decisões importantes na minha vida.

À instituição de ensino SENAI na qual realizo minhas atividades profissionais e que me inspirou pelo labor do ensino técnico e profissionalizante e gerou a minha vontade de crescer como docente.

"Quem quiser chegar a ser o que não é, deverá principiar por deixar de ser o que é"

(Carlos Bernardo González Pecotche)

RESUMO

O presente trabalho aborda o problema de armazenagem industrial com o uso de algoritmos de inteligência artificial conhecidas como técnicas de busca. O trabalho descreve a dinâmica do sistema ASRS (*Automated Storage and Retrieval System*) apresentando-o na forma de matriz. São apresentadas dinâmicas referentes à inserção, organização e retiradas de peças levando em consideração parâmetros e heurísticas relacionadas ao custo de movimento e à organização de peças no armazém. Os resultados pertinentes aos algoritmos das técnicas de buscas aplicadas no sistema ASRS oferecem dados que justificam o uso de inteligência artificial no sistema de armazenagem industrial e permitem ao armazém realizar a operação de organização, pouco utilizada nos sistemas tradicionais de armazenagem. O uso de IA nos ASRSs torna-os mais eficientes em relação ao deslocamento de peças.

Palavras-chave: Indústria 4.0. Armazém automático. Inteligência artificial. Otimização de produção. Automação da manufatura.

ABSTRACT

This paper addresses the problem of industrial storage using artificial intelligence algorithms known as search techniques. The work describes the dynamics of the Automated Storage and Retrieval System (ASRS) presenting it in matrix form. Dynamics concerning the insertion, organization, and removal of parts are presented taking into account parameters and heuristics related to the cost of movement and the organization of parts in the warehouse. The results pertinent to the search techniques algorithms applied in the ASRS system provide data that justify the use of artificial intelligence in the industrial storage system and allow the warehouse to perform the organization operation, little used in traditional storage systems. The use of AI in ASRSs makes them more efficient in moving parts.

Keywords: Industry 4.0. Automatic warehouse. Artificial intelligence. Production optimization. Manufacturing Automation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABDI	Agência Brasileira de Desenvolvimento Industrial
AGV	*Automated Guided Vehicles*
ASRS	*Automated Storage and Retrieval System*
BNDES	Banco Nacional de Desenvolvimento Econômico e Social
IA	Inteligência Artificial
IOT	*Internet of Things*
ITA	Instituto Tecnológico de Aeronáutica
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNI	Confederação Nacional da Indústria
CNPQ	Conselho Nacional de Desenvolvimento Científico e Tecnológico
CPS	*Cyber Physical System*
GTI4.0	Grupo de Trabalho da Indústria 4.0
FMS	*Flexible Manufacturing System*
MEC	Ministério da Educação
PCP	Planejamento e Controle da Produção
RFID	*Radio-Frequency Identification*
SAP	*Systeme, Anwendungen und Produkte in der Datenverarbeitung*
SENAI	Serviço Nacional de Aprendizagem Industrial

LISTA DE SÍMBOLOS

$f(n)$	Função de avaliação
$h(n)$	Função heurística
$\mathrm{A} \times \mathrm{B}$	Número de linhas e colunas em uma matriz
B_o	Total de possibilidades geradas pela organização do armazém
B_{r_0}	Número de casas vazias no armazém
B_{r_i}	Número de endereços que contém estados repetidos da peça do tipo i
C_x	Custo na etapa x
C_t	Custo total
C_{in}	Custo de inserção de peça no armazém
C_{ind}	Indicador de custo do armazém
C_{out}	Custo de retirada de peças no armazém
C_{org}	Custo de organização do armazém
C_{ref}	Índice de referência de custo
G	Indicador proporcional entre custo e organização
N_c	Número de peças de saída em seu endereço prioritário
N_v	Número de vizinhos
N_p	Número de peças
N_{pe}	Número de peças de entrada
N_{ps}	Número de peças de saída
P_c	Peso resultante entre as colunas das laterias do armazém
P_{d_i}	Peso da peça de índice i à direita do armazém
P_{e_i}	Peso da peça de índice i à esquerda do armazém.
P_l	Peso resultante entre as linhas superior e inferior do armazém
P_{a_i}	Peso da peça i na parte acima do centro de massa armário
P_{b_i}	Peso da peça i abaixo do centro de massa do armário
P_r	Peso resultante entre P_c e P_l
p_s	Vetor de peças de saída
$P_{r_{ind}}$	Indicador de Organização do armazém
$P_{r_{ref}}$	Índice de referência de organização do armazém
T_m	Número de endereços do armazém

SUMÁRIO

1 INTRODUÇÃO

1.1 JUSTIFICATIVA

A indústria mundial passou por inúmeras mudanças nas últimas décadas desde o início da primeira revolução industrial. Os equipamentos produzidos em série foram tendência durante muitos anos e os avanços nos processos produtivos foram melhorando a qualidade, segurança e trazendo competitividade no custo dos produtos.

Na história da evolução da indústria, houve três principais marcos até o momento (ABDI, 2018):

- 1ª Revolução Industrial (1780) - marcada pelas máquinas à vapor que movimentavam as máquinas têxteis;
- 2ª Revolução Industrial (1870) - marcada pelas máquinas elétricas e produtos produzidos em série;
- 3ª Revolução Industrial (1969) - marcada pela automação dos processos, substituindo o homem em tarefas repetitivas e inserindo computadores para o controle e obtenção de dados da produção.

Segundo Silveira (2009), a indústria atual ainda aplica no seu processo produtivo um modelo melhorado da 3ª revolução industrial. Entretanto, com a evolução tecnológica de máquinas e equipamentos industriais, promovida pelas necessidades da indústria, o cenário industrial global é marcado por tecnologia de ponta, onde a eletrônica e a tecnologia da informação estão presentes em muitas empresas do Brasil e do mundo, seja qual for o segmento da indústria e o porte das empresas.

Silveira (2019) ainda complementa que, com o avanço da tecnologia de semicondutores, a indústria ganhou um grande número de soluções voltadas ao controle dos processos industriais, como os controladores lógico programáveis, sensores e transdutores industriais, atuadores, robôs, redes industriais e sistemas supervisórios, que proporcionam a automação e controle de uma planta industrial de modo rápido, eficiente, seguro e compacto. Além disso, a internet se demonstrou promissora nos últimos anos promovendo o compartilhamento de informações em velocidade recorde, proporcionando o acesso à tecnologia, contatos de negócios, fornecedores, compra e venda de materiais, em proporções nunca vistas.

Segundo Mello (2018), o investimento das indústrias brasileiras em tecnologia tem crescido de acordo com o crescimento econômico do país. Mesmo que o cenário brasileiro não seja animador para a criação e instalação de empresas em território nacional, a indústria brasileira ainda se mantém forte e confiante em investimento em tecnologia, pois é na tecnologia que a indústria encontra soluções que promovam uma melhoria na balança qualidade/custo de seus produtos e serviços.

O mundo industrial está se direcionando para a chamada 4ª Revolução industrial e, segundo a ABDI (2018), o Brasil, mesmo com suas dificuldades econômicas, também está se preparando para viver estas mudanças que farão parte de um novo mundo tecnológico.

Uma das principais diferenças entre as diferentes formas de manufatura industrial, segundo Bartodziej (2017), será a incorporação da inteligência artificial realizando tomadas de decisão dentro do contexto produtivo. Desta forma, a tecnologia não vem apenas ajudar o homem nas tarefas repetitivas, mas também o auxilia nas atividades cognitivas. Neste novo contexto de trabalho, os robôs poderão trabalhar em conjunto com os humanos aprendendo novas tarefas todos os dias apenas observando os padrões desempenhados, além de otimizar o seu trabalho no decorrer do tempo.

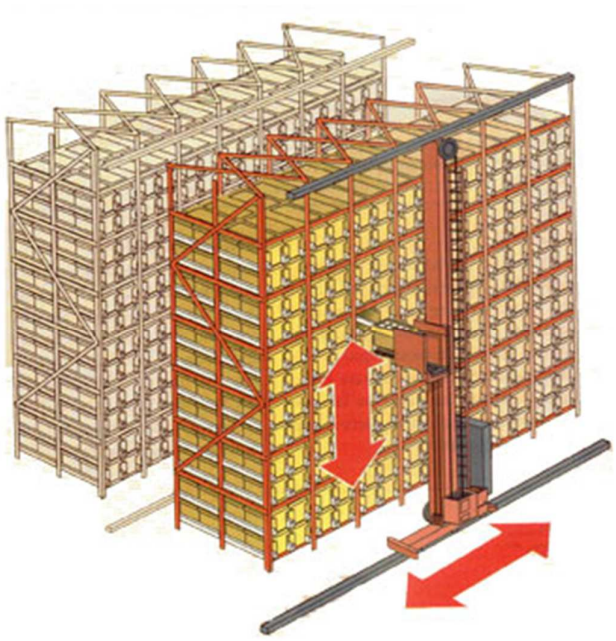
Na automação atual, os robôs são programados para tarefas específicas dentro de um espaço finito de possibilidades. Na indústria 4.0, a inteligência artificial poderá se adaptar às mudanças requeridas no processo produtivo modificando as atividades de todo o conjunto de máquinas, robôs e sistemas cyber físicos em tempo ótimo.

> Para atingir esse objetivo no I 4.0, os aplicativos de inteligência artificial fornecem gerenciamento de metas, planejamento e controle de comportamento. A ideia é que o sistema modifique automaticamente as metas para atender às condições operacionais variáveis e depois ajuste de forma autônoma o comportamento para acomodar as metas alteradas. (TRAPPEY, 2016, p. 7361)

Segundo Bartodziej (2017), a principal diferença entre a indústria atual e a do futuro é que, na atual, a máquina determina o processo produtivo, enquanto na indústria do futuro, a peça dentro do processo produtivo irá determinar por quais máquinas ela deverá passar e o que ela deverá ser futuramente. Isso é possível, pois não só as máquinas estão conectadas entre si, mas também as peças que estão sendo produzidas. Este tipo de demanda surgirá no futuro quando as empresas criarem processos que permitam a personalização de seus produtos.

Segundo Lee (2006), em 1950 a indústria ganhou um novo componente, o sistema de armazenamento e recuperação automatizado, conhecidos como ASRS (do inglês: *Automated Storage and Retrieval System*). Este componente auxilia o armazenamento e recuperação de peças de forma automática, produzindo um ganho de cerca de 70% em relação ao trabalho realizado de maneira manual. A Figura 1, a seguir, mostra um exemplo de sistema ASRS.

Figura 1 – ASRS



Fonte: Adaptado de Unarcorack (2018)

Segundo KHASASI et. al (2015), o ASRS pode ser configurado utilizando sistemas supervisórios que contam com o uso de operadores e/ou configurado a trabalhar de forma automática, contando com sensores industriais para identificação de peças com *Radio-Frequency Identification* (RFID) (YU, Xi; XU, 2012).

Muitas técnicas de otimização de armazéns já têm sido realizadas, no entanto, este trabalho busca estudar a vantagem na utilização de roteamentos de caminhos que gerem o menor custo de movimentação de peças e, também, validar a premissa de que tarefas de auto-organização do armazém contribuem para a melhora da execução de processos. Este estudo traz uma análise da validade de aplicação desta técnica durante a inserção ou retirada de peças.

1.2 METODOLOGIA

Este trabalho modela um armazém industrial utilizando a linguagem de programação em Python. O algoritmo de Terry Winograd (Winston, 1992) é utilizado para auxiliar no deslocamento de peças do armazém. São inseridos parâmetros específicos que auxiliam a dinâmica e análise dos resultados do armazém como endereços de cada uma das posições do armazém, tipo de peças, peso de peças, estado de distribuição das peças, posição da garra, etc. Depois de inseridos os parâmetros é necessário detalhar a dinâmica do armazém como a inserção de peças, organização do armazém e retirada de peças. Com o modelo da planta digital do armazém é possível aplicar métodos de IA para alterar a dinâmica do armazém e realizar a busca por melhores soluções.

São realizadas várias simulações aplicando as técnicas de busca gulosa com diferentes heurísticas e parâmetros. Os resultados destas simulações são comparados com um armazém simples sem o uso de nenhuma abordagem de IA. Desta forma é possível justificar o uso da IA em sistemas industriais como o ASRS.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Analisar os principais parâmetros e dinâmicas de um armazém industrial automatizado, comparando seu desempenho quando submetido a diferentes estratégias de busca de inteligência artificial.

1.3.2 Objetivos específicos

- Estudar a dinâmica de movimento de peças em um sistema ASRS;
- Propor heurísticas em um ASRS que auxiliem na implementação de técnicas de inteligência artificial;
- Estudar algoritmos de IA que auxiliem na resolução do problema de organização do armazém industrial;
- Implementar algoritmos de IA em um armazém virtual utilizando a linguagem de programação em Python.
- Comparar resultados da implementação de diferentes algoritmos de IA com um armazém industrial comum.

1.3.3 Organização do texto

Este trabalho está organizado em cinco partes. No segundo capítulo estão descritos os principais conhecimentos estudados, principalmente em relação à indústria 4.0 e inteligência artificial. No terceiro capítulo está detalhada a dinâmica da estrutura de armazenagem e recuperação de peças de acordo com um estudo de caso. No quarto capítulo estão as soluções e resultados obtidos com a aplicação de técnicas de inteligência artificial. Por fim, no capítulo de conclusão estão pontuadas as principais conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas próximas seções são apresentados os principais temas e informações que fundamentam o conteúdo deste trabalho de dissertação.

2.1 INDÚSTRIA 4.0

A Indústria 4.0, ou 4ª revolução industrial, é um termo criado na Alemanha e que também é conhecida como Manufatura Avançada. A Indústria 4.0 é um conceito de evolução da indústria constituído por vários pilares tecnológicos que possibilitam ter maior integração, controle, eficiência, produtividade, diversidade e adaptabilidade nos processos de manufatura. Segundo Bartodziej (2017), esta revolução tem como destaque os seguintes elementos:

1. Robôs autônomos;
2. Simulação;
3. Sistema de integração horizontal e vertical;
4. Internet das coisas aplicadas na indústria;
5. Cyber-segurança;
6. Computação em nuvem;
7. Manufatura aditiva;
8. Realidade aumentada;
9. Big data e análises estatísticas;
10. Inteligência artificial.

Segundo Venturelli (2018), a tecnologia da automação está evoluindo cada vez mais devido ao processo de digitalização da indústria. Além disso, Venturelli (2018) ainda destaca que a automação atual ou automação 3.0 pode ser representada por estrutura de camadas que se comunicam de maneira vertical entre suas diversas interfaces, com pouca flexibilidade e alta latência. Em um modelo da automação aplicada aos conceitos da Indústria 4.0, a maioria das tecnologias da automação permanecem como pilares da automação, mas a comunicação neste sistema ocorre em todas as cadeias do processo de forma vertical e horizontal, permitindo a troca de informações em todos os setores e em tempo real (Venturelli, 2018).

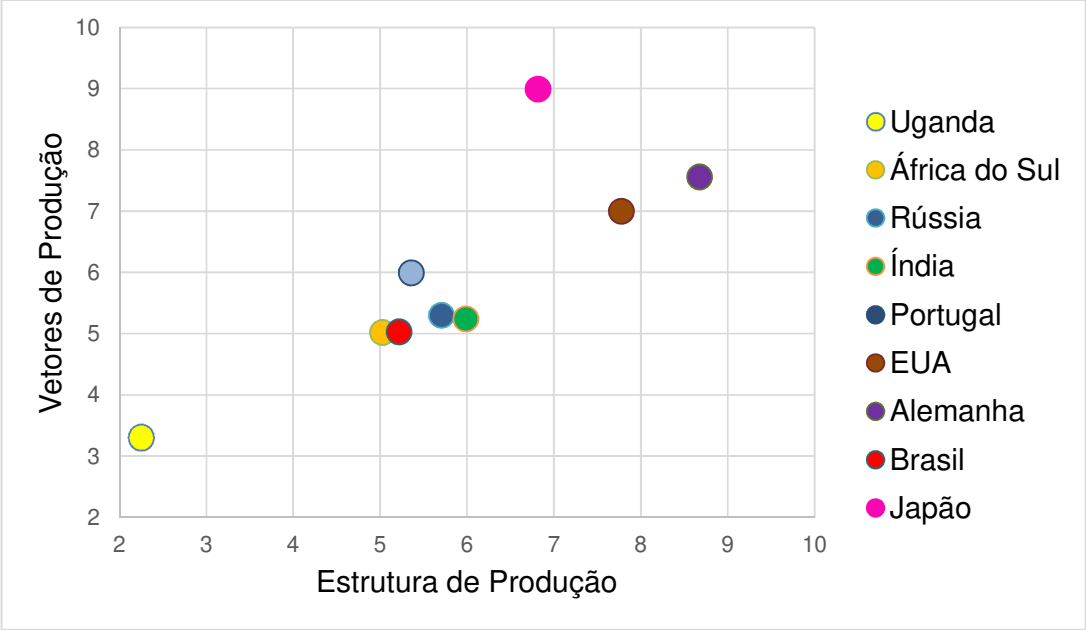
2.1.1 Brasil e a Indústria 4.0

Será que o Brasil está apto para integrar estas tecnologias para este novo cenário? Será que a indústria brasileira já domina a tecnologia da automação para dar

o passo para a 4ª revolução? Quais serão os desafios do Brasil nesta adaptação? O que já está sendo feito para que num futuro próximo a indústria brasileira possa ser competitiva no cenário internacional?

Na corrida da Indústria 4.0, o Brasil está entre os países nascentes para o tema, ou seja, ainda menos preparados e com um legado industrial a ser melhorado. Em contrapartida, alguns países se destacam por seu alto potencial para o tema, seu legado industrial classificando-os como verdadeiros líderes da corrida industrial da manufatura avançada. A figura 2 detalha a posição de alguns países em relação ao tema Indústria 4.0.

Figura 2 - Relação de alguns países com as perspectivas da indústria 4.0



Fonte: Adaptado de ADBI (2018)

Na Figura 2, os países que tem mais pontos à direita tem uma estrutura de produção larga e complexa, ou seja, possuem um legado industrial. Os países que possuem mais pontos para cima possuem mais potencial para a indústria 4.0, ou seja, estão mais preparados. Nestes índices, o Brasil está com 5,03 pontos de vetor de produção e 5,23 pontos de estrutura de produção, enquanto a Alemanha (país líder e pioneiro) está com 7,56 pontos de vetor de produção e 8,68 pontos de estrutura de produção.

Segundo a ADBI (2018), a Indústria 4.0 no cenário brasileiro poderá impactar positivamente na produtividade e, consequentemente, na redução de custos das empresas. Estima-se que a redução de custos da Indústria pode chegar a 73 bilhões

de reais por ano. Esta redução envolve principalmente os pilares de manutenção, custos com energia elétrica e eficiência no chão de fábrica.

Para que o Brasil esteja preparado para essa grande mudança, foi instituído em 2017 um grupo de trabalho para a Indústria 4.0 (o GTI4.0) com o objetivo de elaborar uma agenda nacional para o desenvolvimento e estudo deste novo cenário no Brasil. Este grupo é constituído por órgãos do governo, empresas, sociedade, entre outros. Dentre os mais de 50 participantes do grupo, podem ser destacados SENAI, CNI, ITA, BNDES, CAPES, CNPQ, IOT, MEC e SAP.

Segundo a ABDI (2018), a agenda da Indústria 4.0, que vai de 2017 a 2019, tem os quatro seguintes objetivos:

- Fomentar iniciativas que facilitem e habilitem o investimento privado, em vista da nova realidade fiscal do país;
- Propor agenda centrada no industrial/empresário, conectando instrumentos de apoio existentes, permitindo uma maior racionalização e uso efetivo, facilitando o acesso dos demandantes, levando o maior volume possível de recursos para “ponta”;
- Testar, avaliar, debater e construir consensos por meio da validação de projetos-piloto, medidas experimentais, operando com neutralidade tecnológica;
- Equilibrar medidas de apoio para pequenas e médias empresas com grandes companhias.

Os investimentos para essa mudança deverão ser distribuídos entre entidades públicas e privadas desde as etapas de conscientização da mudança até a etapa final de conexão das indústrias brasileiras com as indústrias internacionais em uma grande rede que se caracteriza a Indústria 4.0.

2.2 INTELIGÊNCIA ARTIFICIAL

As próximas seções tratam sobre os conceitos da inteligência artificial e sua relação com a Automação, apresentando algumas análises e identificando padrões de funcionamento.

2.2.1 O que é a IA?

Antes de expor o conceito da inteligência artificial, vale também refletir sobre o conceito de inteligência. Segundo o dicionário Michaelis (2018) da língua portuguesa,

em suma e que é pertinente neste trabalho, a inteligência representa um conjunto de capacidades para resolver situações novas com rapidez e êxito, adaptando-se a elas por meio do conhecimento adquirido.

Todas as funções realizadas pelos seres humanos dependentes de sua inteligência são realizadas, principalmente, quando se deparam com algum problema, algum tipo de decisão ou quando devem interferir em um sistema cuja participação pode afetar no seu resultado. Estas situações em que a inteligência humana age são precisamente as que as máquinas podem trabalhar, ou seja, onde elas serão capazes de analisar e tomar decisões com uma cognição baseada em algoritmos específicos que imitam a inteligência dos seres humanos.

Quando se trata da inteligência humana, nos estudos da IA, é comum utilizar o termo inteligência natural, ou seja, a inteligência natural dos seres vivos, aquela que não é artificial (ARTERO, 2009, p.19).

Ainda segundo Michaelis (2018), a IA pode ser entendida como: “projeto e desenvolvimento de programas de computador que simulam o pensamento humano, capaz de desenvolver um comportamento inteligente”. Para o professor Artero (2009), a inteligência artificial pode ser dada quando um computador ou máquina apresenta traços inteligentes. Mas o conceito que entende-se ser o que melhor se aloca neste trabalho é dado por Rich e Kinight (1990): “O estudo de como fazer computadores fazer coisas em que, no momento, as pessoas são melhores”.

2.3 REPRESENTAÇÃO DO CONHECIMENTO

Todo processo de tomada de decisão precisa de algum tipo de informação ou conhecimento para processar suas ações. Em termos computacionais, há necessidade de representação e organização de todo conhecimento utilizado pelos algoritmos ditos inteligentes.

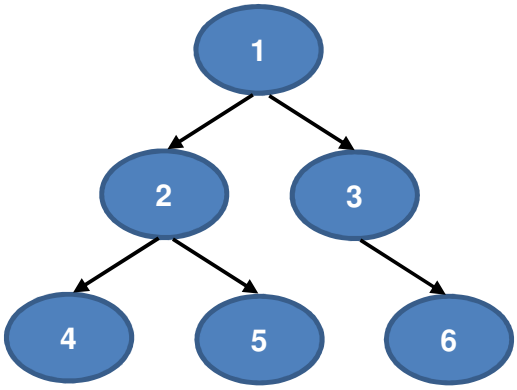
Ainda, para auxílio desses algoritmos a encontrar melhores soluções dentro de sistemas especialistas, existe um conjunto de estratégias que são chamadas de heurísticas. Daqui em diante este termo será utilizado quando as técnicas de busca por soluções dentro de um algoritmo forem tratadas.

2.3.1 Representação por árvores de decisão

Uma das inúmeras formas de tratar problemas e suas resoluções é a utilização de uma árvore de decisão. Estas árvores armazenam todos os estados do problema,

indicando quais serão os estados posteriores a partir de um estado inicial. Em outras palavras, a árvore de decisão contém todos os movimentos para que um problema se modifique de uma posição inicial até um determinado objetivo. (ARTERO, 2009; WINSTON, 1992). Uma árvore de decisão pode ser representada como na Figura 13.

Figura 3 – Representação de uma árvore de decisão



Fonte: Do autor (2019)

Os círculos com os números são os chamados vértices, estados ou nós. Portanto, pode-se perceber que o vértice 1 (nó 1 ou estado 1) representa o estado inicial desta árvore de decisão. Do mesmo modo, 2 e 3 são os sucessores de 1. As linhas entre os vértices são conhecidas como as arestas ou arcos (ARTERO, 2009; NOVIG, 2013).

Os vértices que dão origem a outros vértices podem ser chamados de pais. Os vértices sucessores dos vértices pai podem ser chamdos de filhos ou até mesmo vizinhos. Os vértices que não possuem filhos/vizinhos são conhecidos como vértices folha (NOVIG, 2013).

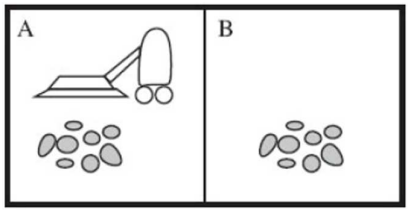
Dentro deste conceito de árvore de decisão, considerando em um instante que o estado inicial do problema é o vértice 1 e o objetivo final desta estrutura está localizada no vértice 6, há inúmeras técnicas para busca desta solução dentro da árvore e cada uma delas apresenta vantagens e desvantagens.

Dependendo do algoritmo de busca utilizado mais ou menos vértices podem ser visitados até se encontrar a solução. Se o tempo é um fator importante na busca por uma solução, então a escolha do algoritmo de busca é uma decisão pertinente de projeto.

Exemplificando a necessidade e utilização da estrutura de árvores de decisão dentro de um contexto prático envolvendo a IA, Novig (2013) apresentou um

problema, que consiste em um aspirador de pó automático que busca por cômodos sujos, limpando-os quando percebe que há presença de sujeira. Depois que a sujeira é eliminada ele parte para outro cômodo. Este exemplo teve o nome de “O mundo do aspirador de pó com dois locais” e pode ser visualizado na Figura 4.

Figura 4 – Problema do aspirador de pó automático

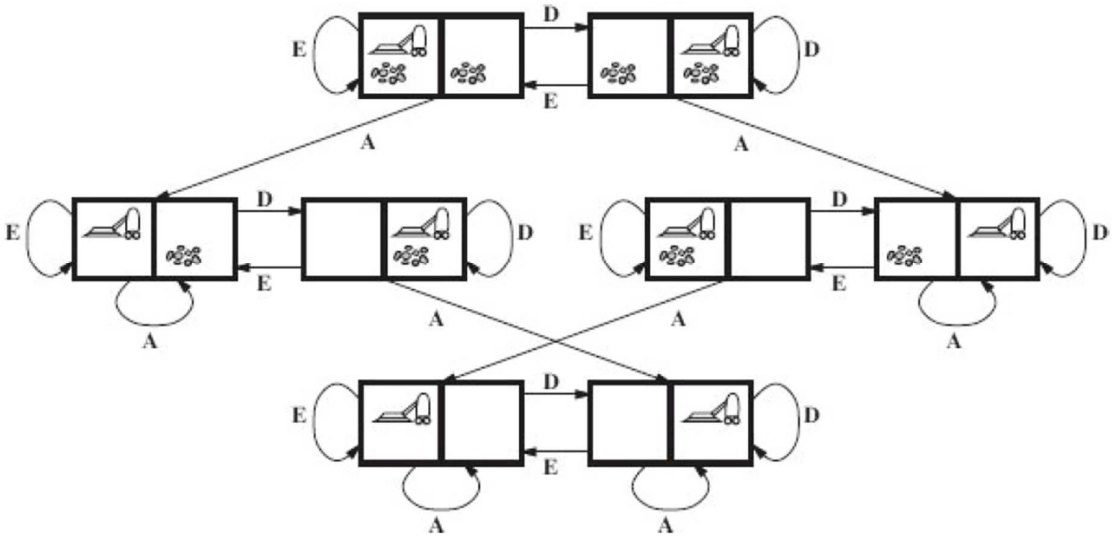


Fonte: Novig (2013)

O robô é capaz de realizar apenas 3 ações: (D) desloca-se para a direita; (E) deslocar-se para a esquerda; e (A) aspira o cômodo atual.

Desta forma, o objetivo final deste problema é que ambos os cômodos A e B estejam limpos. A árvore de decisão resultante pode ser vista na Figura 5.

Figura 5 – Árvore de decisão do problema do aspirador de pó automático



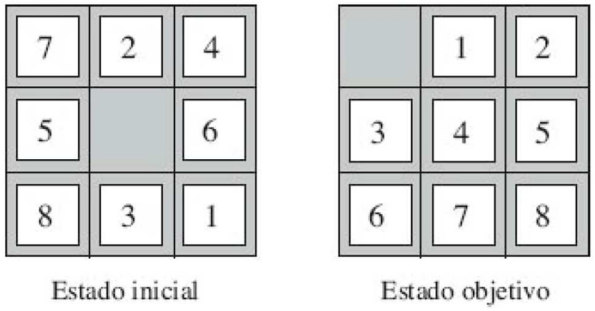
Fonte: Novig (2013)

Percebe-se na Figura 5 que todas as possibilidades do mundo do aspirador de pó foram representadas na árvore de decisão permitindo que as soluções finais possam ter sido encontradas.

As técnicas de inteligência artificial são bem adaptadas a problemas com espaço limitado de soluções. Como jogos, por exemplo, solucionando *puzzles* como quebra-cabeças, xadrez, jogo da velha, sudoku, entre vários outros. Um exemplo de jogo capaz de ser resolvido por IA é o dos oito quadrados, que é semelhante a um quebra-cabeças.

O jogo é composto por um tabuleiro contendo 9 espaços e 8 peças. Cada peça contém um número diferente. As peças contêm os números de 1 a 8 e são espalhadas de maneira aleatoriamente. O objetivo é movimentar as peças a fim de reorganizar o tabuleiro de forma que os números fiquem em sequência crescente e o espaço vazio no lugar que seria o número zero, assim como pode ser visto na Figura 6 (NOVIG, 2013.; ARTERO, 2009).

Figura 6 – Problema dos oito quadrados

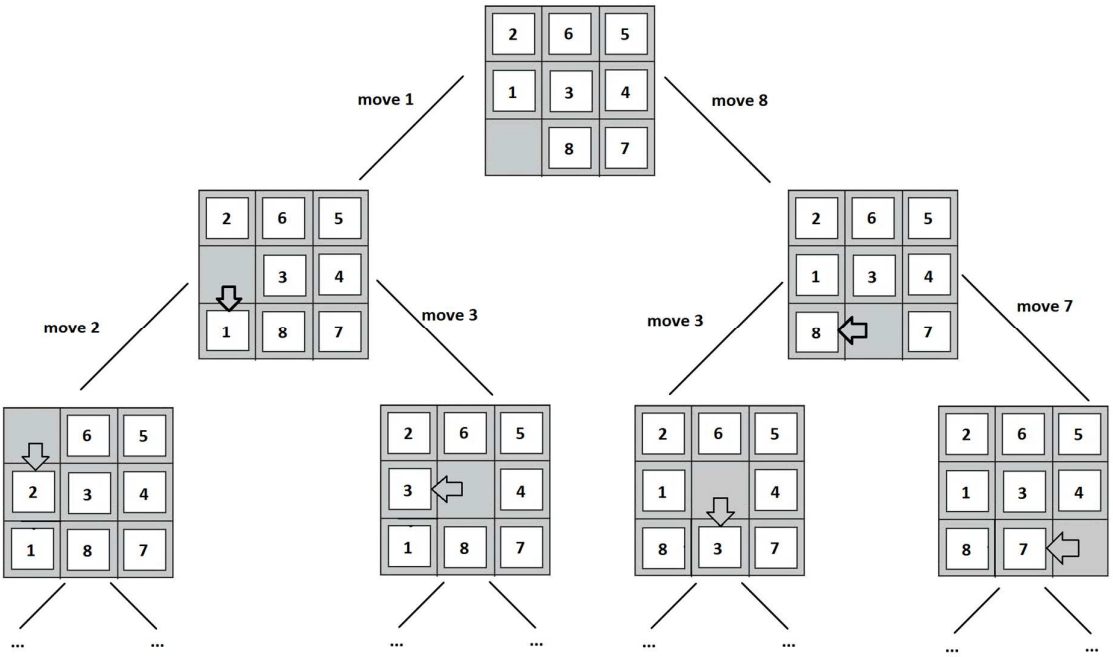


Fonte: Novig (2013)

Para este exemplo não será demonstrada árvore completa, pois o número de estados é muito grade (181400 estados) e não seria possível representá-lo em uma única imagem. Por curiosidade, um tabuleiro 4x4 com 15 peças alcança mais de 1,3 trilhões de estados (NOVIG, 2013; ARTERO, 2009).

Observando os primeiros movimentos do problema 3x3 (Figura 7), a cada movimento, ou a cada estado, são originados dois possíveis novos estados. Desta forma, é possível observar que a cada nova jogada o objetivo fica mais próximo de ser encontrado.

Figura 7 - Árvore de decisão do jogo dos oito quadrados



Fonte: Novig (2013)

Tanto no problema do aspirador de pó quanto no jogo dos 8 quadrados, permanece uma dúvida: uma ação dentro do problema pode ser melhor que outra? A resposta é sim.

A utilização da árvore de decisão e sua estrutura para representação de conhecimento, facilita encontrar caminhos mais rápidos, ou seja, com menor custo até o objetivo final. Existem várias formas de encontrar soluções utilizando a árvore de decisão em inteligência artificial. Estas soluções podem ser ruins (alto custo computacional envolvido), boas (relação aceitável entre custo computacional e resultado) e a solução ótima (melhor solução possível).

Uma das formas mais importantes na resolução de problemas na inteligência artificial é utilizando técnicas de busca. Basicamente, as técnicas de busca se baseiam na estrutura do problema para chegar a uma solução. Dependendo da técnica escolhida, uma obterá uma solução melhor que outra.

2.4 TÉCNICAS DE BUSCA

A partir do problema descrito por árvores de decisão, as possibilidades deste problema estarão descritas na forma de estados e dependendo da ação tomada, novos estados resultantes aparecerão até que a solução ou objetivo seja alcançado.

> Esse processo de procurar por tal sequência de ações que alcançam o objetivo é chamado de busca. Um algoritmo de busca recebe um problema como entrada e devolve uma solução sob a forma de uma sequência de ações. Depois que uma solução é encontrada, as ações recomendadas podem ser executadas. Isso se chama fase de execução. (NOVIG, 2013, p. 99)

Quando se está utilizando técnicas de busca, é necessário definir um problema por 5 componentes indispensáveis para sua resolução:

1. Estado inicial;
2. Descrição de possíveis ações;
3. Modelo de transição;
4. Teste de objetivo;
5. Função custo.

É importante ter em conta os itens apresentados acima para que o problema seja bem definido e que dê condições ao algoritmo de busca encontrar uma solução e, posteriormente, definir se a solução foi ruim, boa ou ótima (NOVIG, 2013).

Ao aplicar uma técnica de busca são necessários cuidados em alguns aspectos a fim de não deixar a busca muito ineficiente. Este cuidado é realizado através de um mecanismo de controle para que todos os nós possam ser visitados, mas que não sejam visitados nós de forma repetida (ARTERO, 2009).

Os passos básicos para a realização destes mecanismos são:

1. Escolhe-se o estado inicial;
2. Testa-se se é estado objetivo;
3. Se não for, expande-se os estados filhos (geram-se sucessores) seguindo uma técnica de busca específica;
4. Escolhe-se um próximo estado e realiza-se os passos anteriores até encontrar a solução (estado objetivo) ou não existirem mais estados a serem visitados.

Existem na literatura algumas técnicas que buscam encontrar a solução do problema de acordo com estes passos básicos. Dentre as principais técnicas de busca estão: Busca em largura; Busca em profundidade; Busca por custo; Busca heurística; Busca A*; *Hill Climbing*; *Simulated Annealing*; e Busca Tabu (NOVIG, 2013).

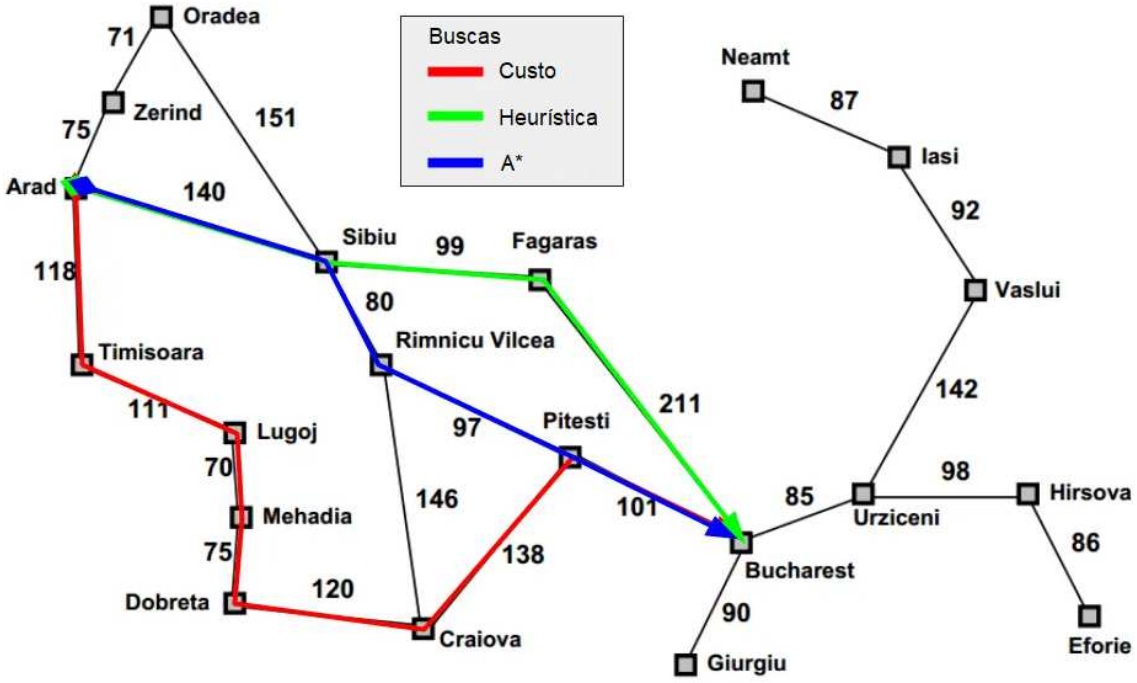
Apenas algumas destas técnicas serão detalhadas no decorrer deste trabalho, de modo a ajudar na compreensão do motivo de utilização dessas técnicas de IA na execução do objetivo deste trabalho.

Antes de abordá-las há necessidade de esclarecer como é avaliado o desempenho delas. Assunto dá próxima subseção.

2.4.1 Desempenho em técnicas de busca

Para compreender a lógica da busca através de uma árvore de decisão, Novig (2013) traz um problema de deslocamento de uma cidade a outra, na Romênia, de Arad até Bucareste (ver Figura 8). Neste exemplo, há diversos possíveis caminhos que levam até Bucareste que, dependendo da estratégia (técnica) adotada e, por sua vez, dependendo do seu objetivo, haverá diferentes caminhos até o objetivo final.

Figura 8 - Mapa rodoviário simplificado de parte da Romênia.



Fonte: Novig (2013)

É possível enxergar na Figura 8 três caminhos realçados que levam a Bucareste, obtidos cada um por algoritmos de busca diferentes. Perceba que para cada solução há diferentes custos de viagem. Dependendo da técnica de busca, a expansão e escolha dos próximos estados na árvore de decisão irá ser diferente e, automaticamente, irá gerar soluções diferentes. Mas como definir se a estratégia de busca foi realmente bem realizada? Como medir eficiência do algoritmo escolhido?

Segundo Novig (2013) e Artero (2009), o desempenho dos algoritmos regidos por uma estratégia de busca pode ser avaliado de acordo com:

- **Completeza**: O algoritmo encontrará alguma solução caso ela exista?
- **Otimização**: O algoritmo encontrará a solução ótima?
- **Complexidade de tempo**: Quanto tempo levará para encontrar a solução?
- **Complexidade de espaço**: Qual a memória necessária para realizar todas as ações e armazenar as informações até que a solução seja encontrada?

Nas próximas subseções são mostrados com mais detalhes algumas técnicas de busca que são pertinentes ao trabalho.

2.4.2 Busca em largura

A busca em largura, ou busca em amplitude, é uma das formas de busca mais simples presente nas técnicas pertinentes às soluções de IA. Esta estratégia não conta com nenhum tipo de heurística, ou seja, não possui nenhum tipo de informação sobre o problema para realizar a mudança de estados a partir do estado inicial, isto quer dizer que esta estratégia é conhecida como busca sem informação (ARTERO, 2009; NOVIG 2013).

Ela prioriza os níveis de uma árvore de decisão, isso quer dizer que antes do algoritmo expandir novos estados, todos os estados de um determinado nível são visitados a fim de verificar o estado objetivo (ARTERO, 2009; NOVIG, 2013).

2.4.3 Busca em profundidade

A busca em profundidade, assim como a busca em largura também é um tipo de busca sem informação. A busca em profundidade visa verificar os estados indo em direção ao nó folha, ou seja, um estado que não possui sucessores.

Segundo Artero (2009) e Novig (2013), a busca em profundidade possui uma lógica na ordem de visitação baseado em pilha, ou seja, o último estado ao entrar é o primeiro a sair. Desta forma, este algoritmo sempre busca expandir o nó mais profundo da árvore de decisão.

A busca em profundidade pode ser estendida utilizando duas estratégias adicionais que são: busca em profundidade limitada e a busca em profundidade iterativa. Através destas duas abordagens, a estratégia de busca em profundidade se torna viável dentro de sua aplicação e assim é possível determinar seus algoritmos de maneira mais eficiente.

Busca em profundidade limitada

A busca em profundidade limitada é uma estratégia interessante quando se tem muitos ramos longos ou infinitos, ou seja, o nó folha de uma árvore está muito distante ou não existe. Neste caso, é interessante criar um parâmetro de profundidade máxima em que os nós são expandidos. Além disso, pode-se dizer que a busca em profundidade convencional é uma busca em profundidade limitada com limite infinito (NOVIG, 2013).

Talvez limitar a busca em determinados níveis não seja tão interessante, pois é possível que a solução esteja além do limite escolhido. Por isso, uma forma complementar de realizar a busca em profundidade seja utilizar uma abordagem iterativa.

Busca de aprofundamento iterativo

A busca de aprofundamento iterativo é uma melhoria do caso anterior. Ela trabalha com limites, entretanto, estes limites podem ser modificados se por acaso a solução não for encontrada. Para que o algoritmo possa encontrar uma solução dentro de um determinado limite de profundidade, é interessante que ele possa, por conta própria, impor diferentes limites de profundidade.

Um bom exemplo disso seria um homem cavando um buraco na areia em busca de água. Primeiro ele cava 1 metro de profundidade, vê que não encontra a água, decide escavar pelas laterais e novamente não encontra a água. Depois decide cavar mais um metro, vê que não encontra a água, decide escavar pelas laterais e novamente não encontra a água. Depois decide cavar mais um metro e assim por diante sempre aumentando o nível de profundidade a cada tentativa. Em cada iteração a profundidade é limitada e na próxima iteração este limite é incrementado.

Segundo Novig (2013, p.125), “o aprofundamento iterativo é o método de busca sem informação preferido quando o espaço de busca é grande e a profundidade da solução não é conhecida”. Isto é, esta técnica de busca possui suas vantagens quando não se tem informações sobre a solução dentro do contexto do problema.

Quando o problema apresenta características próprias e especiais talvez seja difícil escolher a melhor estratégia de busca sem informação, no entanto, quanto mais informações sobre o contexto do problema, mais os algoritmos podem ser adaptados e melhorados a fim de encontrar a solução de forma mais rápida.

As próximas subseções tratam de estratégias de busca que contém informações adicionais e que podem deixar a aplicação do algoritmo mais eficiente.

2.4.4 Busca gulosa

A busca gulosa é uma estratégia de busca que utiliza informação do contexto do problema, isso quer dizer que esta é uma estratégia ou técnica de busca com informação. A busca com informação também é conhecida como busca heurística. (ARTERO, 2009; NOVIG, 2013).

A informação pode possibilitar ao algoritmo expandir nós ou caminhos mais promissores, tais como, o caminho mais curto, o menor tempo de deslocamento, o trajeto mais barato etc. Elas podem ser incorporadas a fim de capacitar o algoritmo a avaliar e chegar em resultados bons em um tempo computacional menor.

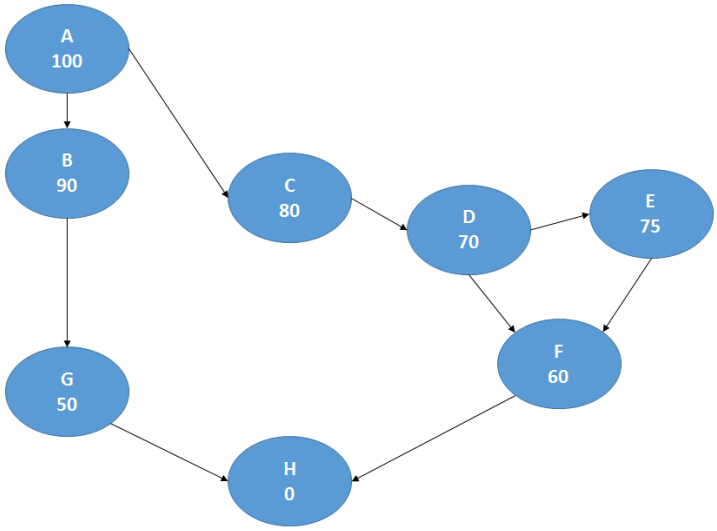
Para que a busca heurística seja possível, nestes algoritmos é importante que haja uma função de avaliação $f(n)$ que poderá informar valores ao algoritmo com base em informações do problema. Esta função $f(n)$ tem o intuito de avaliar numericamente informações (distância, por exemplo) entre todos os nós expandidos. Desta forma, o algoritmo poderá escolher o estado ou nó de acordo com os valores da função (ARTERO, 2009).

Com base em todas estas informações e funções presentes na busca heurística, ela tende a expandir os nós mais próximos da meta ideal, esperando que talvez aquela informação futuramente chegue de maneira mais rápida à solução desejada (ARTERO, 2009; NOVIG, 2013).

Para a busca gulosa, a função de avaliação é regida por uma função especial chamada de função heurística $h(n)$ que contém o custo estimado do caminho de menor custo do estado do nó n para um estado objetivo. De maneira geral, pode-se dizer que para a busca gulosa, tem-se que $f(n) = h(n)$. Desta maneira, pretende-se que no nó objetivo da árvore de decisão a função heurística seja nula, ou seja, $h(n) = 0$ (NOVIG, 2013).

Para deixar clara a função deste tipo de busca em um exemplo prático, veja a Figura 9 contendo a distância entre pontos diferentes, cujo objetivo é sair do estado A e chegar em H.

Figura 9 – Exemplo para busca gulosa



Fonte: Do autor (2019)

Os valores dentro dos estados são a distância em linha reta entre o estado atual e o objetivo final. Com esta informação é que o algoritmo irá expandir novos estados e realizar o teste de objetivo final.

A busca gulosa não encontra a solução ótima global do problema, podendo inclusive em alguns casos, não encontrar a solução final, pois pode ficar apostando que alguns caminhos sejam melhores que os demais e ficar girando em círculos dentro do código, portanto pode-se dizer que esta estratégia de busca é incompleta e não ótima.

2.4.5 Busca A-estrela (A*)

A busca do tipo A-estrela visa melhorar a função de avaliação $f(n)$ inserindo novas informações a ela, evitando que caminhos muito caros sejam expandidos. Assim como a busca gulosa, esta estratégia ainda não garante solução ótima global podendo encontrar apenas soluções ótimas locais. No entanto, esta estratégia, permite que o algoritmo sempre encontre uma solução, ou seja, este tipo de busca é completo (ARTERO, 2009; NOVIG, 2013).

Assim como na busca gulosa, o algoritmo da busca A-estrela irá escolher/expandir caminhos que proporcionem o menor valor de $f(n)$. Uma característica interessante desta estratégia é que sempre que um nó é expandido, o caminho ótimo até este nó foi encontrado. A heurística escolhida para cada problema é particular, ou seja, para

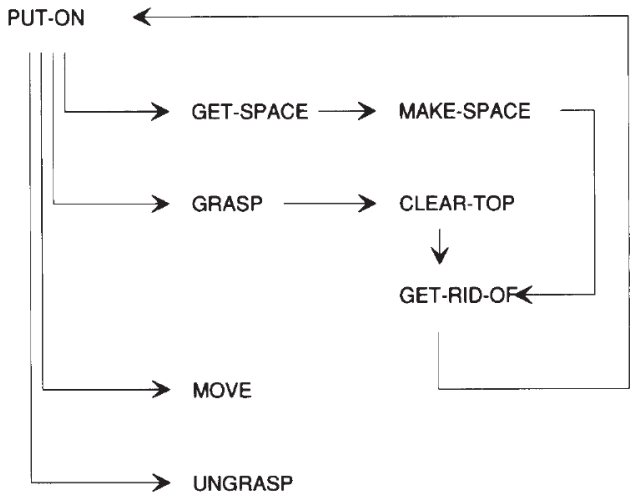
cada problema, podem ser cedidos diferentes tipos de informações que proporcionem uma melhora no caminho a cada iteração.

2.4.6 Algoritmo de Terry Winograd

Para solucionar alguns problemas utilizando inteligência artificial, pode-se utilizar uma técnica chamada redução de problemas (WINSTON, 1992). Esta técnica é aplicada quando se compreende um comportamento padrão do sistema a ser controlado. Desta forma é possível resumir as principais ações do sistema em pequenas funções ou objetivos específicos.

Segundo Winston (1992), Terry Winograd desenvolveu um algoritmo capaz de movimentar peças posicionando um bloco principal ou bloco viajante no topo de blocos alvos. Winograd percebeu que as ações de encontrar espaço, agarrar, movimentar, mover, soltar um bloco, obter espaço, limpar o topo de um bloco alvo e se livrar de um bloco viajante eram ações padrão do sistema de movimentação de blocos. Desta forma, Terry reduziu o problema de movimentação de blocos, também conhecido como *block's world*, de acordo com às seguintes funções.

Figura 10 – Algoritmo de Terry Winograd para o problema *block's world*



Fonte: Adaptado de Winston (1992)

No algoritmo de Terry Winograd, tem-se as funções *PUT-ON, GET-SPACE, MAKE-SPACE, GRASP, CLEAR-TOP, GET-RID-OF, MOVE, UNGRASP*.

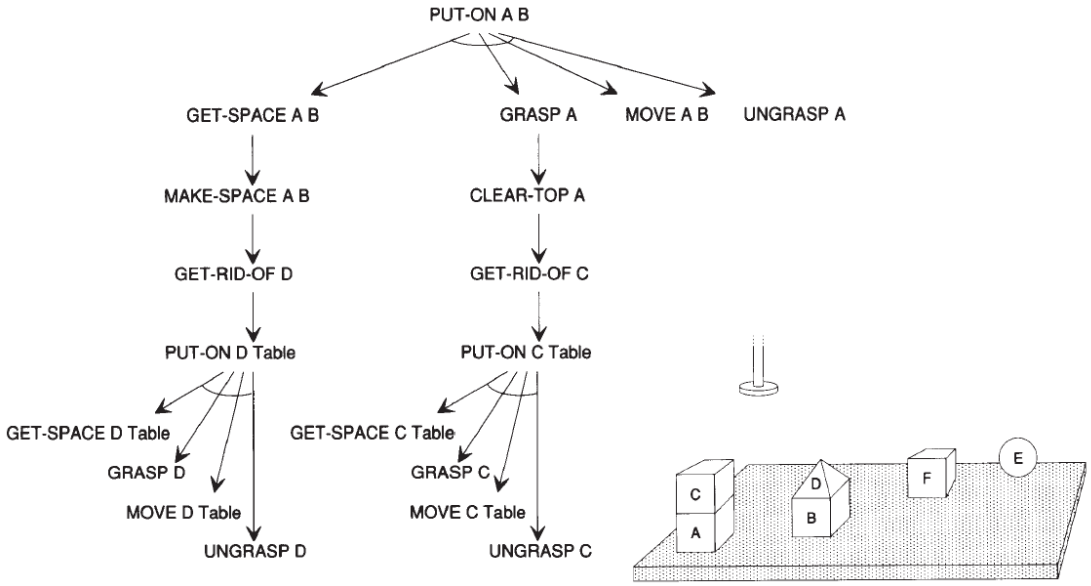
- *PUT-ON*: organiza a inserção de um bloco no topo de outro bloco. Para isto, ativa outros procedimentos que encontram um lugar específico no topo de

um bloco alvo, agarrando o bloco viajante, movimentando-o e soltando-o em um lugar específico.

- *GET-SPACE*: encontra espaço no topo do bloco alvo para acomodar o bloco viajante.
- *MAKE-SPACE*: ajuda a função *GET-SPACE*, quando necessário, pela movimentação de blocos-obstáculos até que haja espaço suficiente para um bloco viajante.
- *GRASP*: agarra um bloco. Esta função auxilia as funções *CLEAR-TOP* e MAKE-SPACE.
- *CLEAR-TOP*: faz a limpeza do topo de um bloco. Chama a função *GET-RID-OF* para se livrar do bloco-obstáculo que está acima do bloco alvo.
- *GET-RID-OF*: se livra de um bloco-obstáculo colocando-o na mesa.
- *MOVE*: move os objetos utilizando uma garra robótica.
- *UNGRASP*: faz a garra robótica soltar o objeto.

A Figura 11 mostra um exemplo da aplicação da movimentação de blocos no sistema block's world, onde deseja-se inserir o bloco A no topo do bloco B.

Figura 11 – Exemplo da aplicação do algoritmo de Terry Winograd



Fonte: Adaptado de Winston (1992)

Este é um algoritmo recursivo, pois para realizar a função GET-RID-OF, a função principal PUT-ON é chamada novamente para inserir um bloco sobre a mesa.

2.5 ARMAZÉNS E SEU CONTEXTO NA INDÚSTRIA

Antes de iniciar a teoria do controle dos ASRS, é importante entender a estrutura da organização de um armazém. Ao organizar um armazém, os profissionais separaram os itens seguindo alguns critérios. Segundo Tompkins et. al. (1996), estes critérios auxiliam o operador a encontrar estes itens mais facilmente quando se deseja encontrar ou retirar um item anteriormente estocado. Dos critérios utilizados na organização do armazém são:

- Tipo de material (ex.: produto A na primeira fileira e produto B na segunda);
- Aplicação do material (ex.: produtos que serão usados no processo ou que saíram do processo);
- Peso do material (ex.: materiais mais leves em cima e mais pesados embaixo);
- Mercado (ex.: produtos internacionais no andar superior e nacionais no andar inferior);
- Cor do material;
- Código do material (ex.: do número 100 ao número 200 armazenados nas primeiras fileiras e do 201 ao número 300 nas demais fileiras);
- Tamanho do material;
- Utilização do material (ex.: produtos mais usados nas primeiras fileiras e menos utilizados nas últimas);
- Semelhança no conjunto de materiais (ex.: se os materiais foram entregues ou devem sair juntos, devem ficar em posições próximas no armazém).

No geral, existem várias formas de se organizar o leiaute de um armazém. A aplicação e tipo da indústria, bem como a utilização dos processos produtivos são quem determinam a escolha do tipo de armazém (TOMPKINS et. al., 1996).

Observa-se até o momento que organizar um armazém requer uma série de parâmetros importantes, mas também um estudo sobre os materiais que serão armazenados e, principalmente, qual a sua função dentro de um processo.

Segundo Mingru (2009), para organizar um armazém, deve-se estudar uma estratégia, implementá-la e acompanhar o seu desempenho. Quando se está buscando uma estratégia de organização do armazém, é importante ter inúmeros dados e, muitas vezes, realizar alguns cálculos para tal. Todos estes estudos buscam a otimização do espaço disponível de acordo com a entrada e saída de materiais.

Além disso, visam a agilidade no processo de armazenagem e recuperação, buscando minimizar o tempo destas funções (MINGRU, 2009; DAHUA, 2009).

Segundo Tompkins et. al. (1996, p. 426), “O leiaute ideal é aquele que procura minimizar a distância total percorrida com uma movimentação eficiente entre os materiais, com a maior flexibilidade possível e com custos de armazenagem reduzidos”.

O estudo deste tema leva a compreender que a otimização de um armazém é uma situação importante a ser levada em consideração, antes mesmo de pensar em automatizá-lo.

2.5.1 Células de manufatura e seus elementos

A indústria atual possui tecnologias muito avançadas dentro do contexto de automação da manufatura. Outras soluções em um contexto industrial do futuro virão a surgir e talvez substituam alguns dos modelos presentes nos processos produtivos atuais. Segundo Khasasi (2016), atualmente, as plantas industriais são caracterizadas por vários elementos, nos quais podem ser citados e classificados como FMS (*Flexible Manufacturing System*):

- Máquinas e equipamentos para a montagem de peças;
- Máquinas e equipamentos de fabricação mecânicas em geral;
- Máquinas e equipamentos de teste e qualidade;
- Máquinas e equipamentos para o tratamento químico ou térmico de produtos;
- Sistemas de entrada e saídas de materiais (PCP);
- Sistemas de Gestão em todos os níveis da manufatura;
- Robôs;
- Sistemas de controle em geral;
- Redes industriais;
- Transportadoras e esteiras;
- AGVs (*Automated Guided Vehicles*);
- Paletizadoras;
- Sistemas de armazenamento e recuperação de objetos (armazéns).

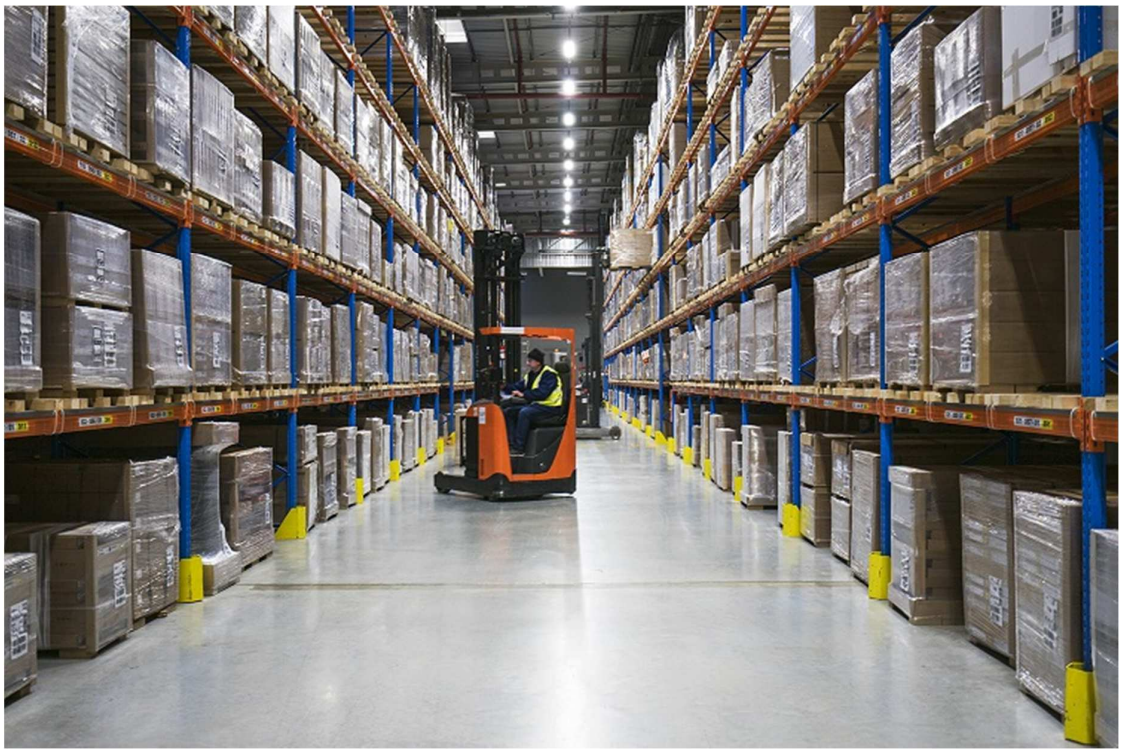
Compõe o objetivo deste trabalho explorar o funcionamento de sistemas de armazenamento e recuperação de objetos de maneira automática. Estes sistemas

também são conhecidos como ASRS (*Automated Storage and Retrieval System*) ou, ainda, chamados de armazéns automáticos.

2.5.2 Automated Storage and Retrieval Systems (ASRS)

O armazém comum é composto de um sistema matricial de prateleiras, composto por um número X de linhas e por um número Y de colunas (DAUHA, 2009). A ideia é que, nesta configuração, o armazém possa conter um número de X vezes Y produtos de uma única vez. A Figura 12, mostra um armazém industrial presente em uma grande empresa.

Figura 12 – Empilhadeira em um armazém comum



Fonte: Eurobuild (2018)

Os armazéns têm a função simples de armazenar itens que participarão de um processo industrial, seja apenas pela simples estocagem de materiais, produtos prontos esperando para serem transportados para o cliente, produtos que vieram do fornecedor e estão esperando para atuar no processo de manufatura, entre outros (LEE, 2006).

Estes armazéns industriais, quando atuam de maneira comum, devem ser manuseados com o auxílio de equipamentos motorizados, na sua maioria por

operários, como mostra a Figura 12. Este processo, pode ser auxiliado por equipamentos especiais como empilhadeiras ou talhas industriais, que apresentam uma dinâmica de processo de armazenagem lenta, devido à dependência de destreza do operário.

O processo de armazenagem, quando realizado de maneira simples e puramente manual, demanda tempo, treinamento de profissionais que atuam direta e indiretamente, além de um planejamento e acompanhamento manual dos itens armazenados (LEE, 2006).

Todas estas situações ocorrendo em um simples processo de armazenagem, podem comprometer a eficiência e velocidade de um processo de manufatura. Ainda mais quando este contexto é aumentado pelos seguintes fatores: tamanho do armazém, rotatividade e diversidade de materiais (LEE, 2006).

Diante deste contexto apresentado, muitas empresas já adotam o uso de armazéns automáticos, que possibilitam maior controle, agilidade, precisão, segurança, eficiência e organização (LEE, 2006; DAUHA, 2009; PIZANI, 2007; SERAFINI, 1988).

Em um sistema automático, as garras ou pás de um robô podem trabalhar ao lado dos armazéns, ou então, o sistema pode conter robôs que deslizam entre os eixos X e Y da matriz e possibilitam a inserção ou retirada de objetos, substituindo os operadores, talhas e empilhadeiras (LEE, 2006; KHASASI, 2015).

Os robôs que trabalham junto ao armazém são comandados por sistemas integrando supervisórios e sensores que facilitam o operador a controlar e decidir a posição dos materiais na matriz. Neste sistema automático, diferentes tipos de controle podem ser implementados de acordo com o nível de exigência e rigor do processo (KHASASI, 2016).

2.6 SÍNTESE DO CAPÍTULO

O capítulo 2 expôs alguns conceitos da indústria 4.0 e qual o papel de instituições públicas e privadas brasileiras no desenvolvimento e preparação das tecnologias necessárias para a aplicação da quarta revolução industrial. Além disso, foram expostos alguns pilares da indústria 4.0 como a inteligência artificial aplicada aos processos de manufatura contribuindo com a flexibilidade e tomadas de decisões nos processos produtivos.

Dada a importância da inteligência artificial no âmbito da manufatura avançada, alguns algoritmos foram abordados afim de estudar sua aplicabilidade a processos diversos, que posteriormente possam ser adaptados para o contexto industrial.

O capítulo 2 abordou o processo de armazenamento na indústria contextualizando o uso de máquinas automáticas. Foi apresentada a estrutura do ASRS e suas vantagens em relação ao processo de armazenagem manual. O sistema ASRS foi escolhido, pois a maioria das empresas de médio e grande porte possuem algum tipo de armazenamento, seja de insumos ou produtos prontos para a entrega. A gestão da logística de materiais é uma realidade encarada pela maioria das empresas. Melhorar este processo utilizando a IA pode ser uma alternativa para atingir bons resultados e ao mesmo tempo conectar este processo à necessidades da indústria 4.0.

O próximo capítulo tratará um estudo de caso de um ASRS descrevendo a sua dinâmica e definindo parâmetros específicos de seu funcionamento. Além disso, alguns algoritmos de IA já são abordados no próximo capítulo como o algoritmo de movimentação de peças em acordo com um armazém de referência.

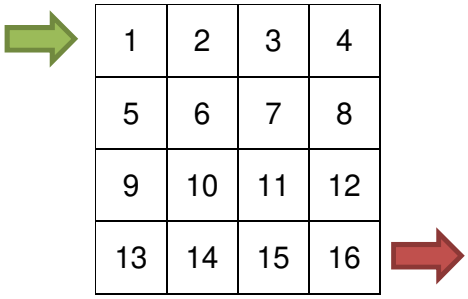
3 DESCRIÇÃO DO PROBLEMA

3.1 DINÂMICA DO ARMAZÉM

Para iniciar a descrição do problema é necessário entender qual a dinâmica real do armazém. Considera-se um armazém com uma única garra que irá movimentar os objetos em uma estrutura de dimensões iguais a $A \times B$, sendo A (número de linhas) e B (número de colunas) iguais (armazém quadrado).

Cada posição do armazém pode alocar apenas uma única peça por vez. O número máximo de peças que cabem no armazém é dada pela multiplicação do número de linhas pelo número de colunas. Uma ilustração do armazém pode ser vista na Figura 13.

Figura 13 – Matriz que representa um armazém industrial



Fonte: Do autor (2019)

Como o tamanho do armazém exemplo é de 4 linhas e 4 colunas, sua capacidade máxima é de 16 peças. A Figura 13 mostra exatamente cada uma das posições do armazém com um endereço particular. As posições do armazém são dadas pelos endereços da Tabela 1.

Tabela 1 – Endereços e posições de um armazém exemplo

Posição	Endereço	Posição	Endereço
1	(0,0)	9	(2,0)
2	(0,1)	10	(2,1)
3	(0,2)	11	(2,2)
4	(0,3)	12	(2,3)
5	(1,0)	13	(3,0)
6	(1,1)	14	(3,1)
7	(1,2)	15	(3,2)
8	(1,3)	16	(3,3)

Fonte: Do autor (2019)

Pela Figura 13 é possível observar que no armazém as peças serão inseridas através do endereço (0,0), representado pela seta verde, enquanto que as peças sairão pelo endereço (3,3), representado pela seta vermelha. Esta dinâmica foi escolhida de maneira arbitrária e poderia ter sido escolhida qualquer uma das outras posições do armazém para a inserção ou retirada de peças, inclusive posições iguais para entrada e saída.

Neste armazém, poderão entrar peças de diversos tipos. Para classificar as peças poderão ser utilizados números inteiros e positivos. Além disso, cada número também está representando uma espécie de peso do material que será colocado no armazém. Este peso está sendo atribuído livre de dimensões padronizadas, apenas para controle e identificação dos parâmetros nas futuras simulações.

Assim, neste trabalho o armazém é representado por uma matriz, onde o valor ou tipo das peças irão assumir as posições previamente estabelecidas. A Figura 14 mostra que são inseridas peças de 4 tipos: 1, 2, 3 e 4. As posições com valor 0 significam a ausência de peças, ou seja, o espaço vazio.

Na Figura 14 ainda é possível observar que um determinado tipo de peça pode ser inserido mais de uma vez no armazém. Note que este armazém não possui nenhuma organização em relação ao tipo ou peso da peça.

Figura 14 – Exemplo de armazém dotado de algumas peças arbitrárias

1	0	2	0
0	3	0	4
3	0	2	0
0	1	0	2

Fonte: Do autor (2019)

Como já comentado, a organização dos armazéns traz uma série de vantagens em relação ao tempo de logística e eficiência ao longo de um processo de inserção ou retirada de peças. Para que este armazém exemplo possa ser organizado, uma garra de robô, por exemplo, deverá deslocar uma peça de um local para outro de acordo com as seguintes ações:

1. Deslocar a garra através do endereço de sua posição inicial até o endereço da peça escolhida;

2. Agarrar a peça escolhida;
3. Deslocar a garra com o objeto agarrado até a posição final desejada;
4. Soltar o objeto na posição desejada.

As próximas subseções detalham um pouco mais a dinâmica da garra no armazém.

3.1.1 Dinâmica da garra no armazém

A garra é o principal agente dinâmico no armazém, pois é o responsável pelo deslocamento de todas as peças, inclusive receber as peças na entrada e fornece-las na saída. Nenhuma peça muda de posição no armazém se não for realizado um deslocamento da garra. Este deslocamento influencia o armazém em três principais aspectos: tempo de deslocamento, custo de deslocamento e organização do armazém.

Neste trabalho o custo de deslocamento do armazém também será adimensional, levando em consideração a quantidade de posições do armazém como a distância percorrida pela garra tanto no sentido vertical quanto no sentido horizontal. Considerar-se que a garra não irá se deslocar na diagonal. Para compreender a dinâmica da garra e a análise do custo de seu deslocamento, é apresentado um exemplo.

Em um armazém com as peças 1, 2, 3 e 4 são analisadas as seguintes situações: a peça 2 posicionada no endereço (2,2) será inserida na posição vazia (2,3). Em seguida, a peça 1 do endereço (3,1) será direcionada para a posição vazia (0,1). A posição inicial da garra é (0,0). A Figura 15 mostra o armazém com as posições de origem (em azul) e a posição de destino (em roxo). A posição inicial da garra é representada pela cor laranja.

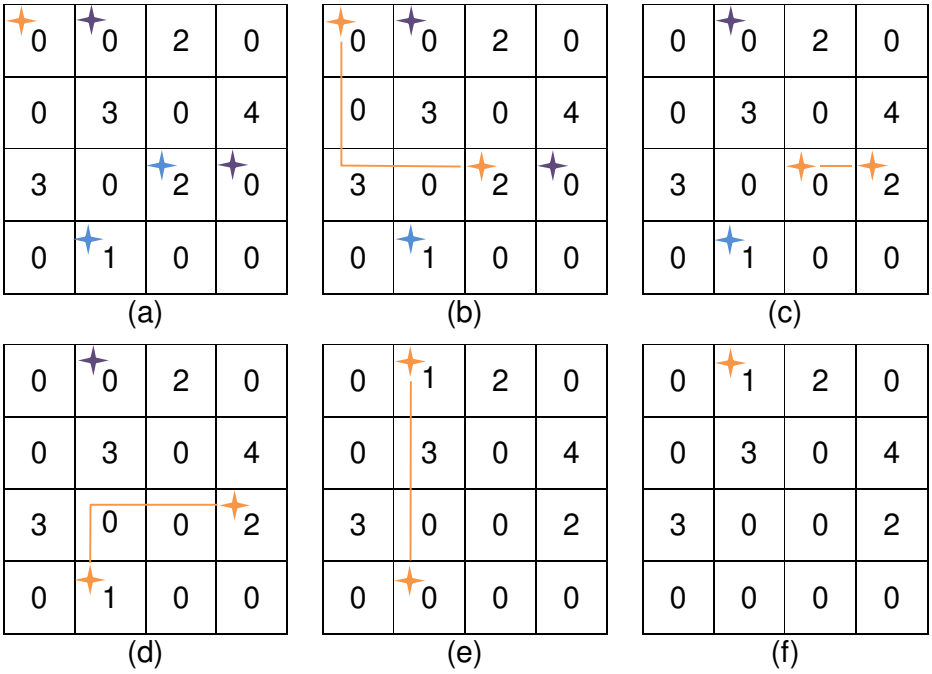
Figura 15 – Pontos de origem-destino no movimento de peças de um armazém

0	0	2	0
0	3	0	4
3	0	2	0
0	1	0	0

Fonte: Do autor (2019)

A Figura 16 demonstra exatamente os movimentos necessários e deslocamentos da garra até que as duas peças sejam de fato posicionadas em seu local correto.

Figura 16 - Deslocamento de peças no armazém exemplo



Fonte: Do autor (2019)

Na Figura 16(a) é possível observar a posição inicial da garra (laranja) e as posições origem (azul) e destino (roxo) das peças 1 e 2. A Figura 16(b) demonstra o deslocamento da garra de sua posição inicial até a peça do tipo 2, com custo de 4 blocos. Na Figura 16(c) movimenta-se a peça 2 para o espaço vazio objetivo, com custo de apenas 1 bloco. Analisando a Figura 16(d), pode-se observar o deslocamento da garra vazia entre as peças 2 e 1, com custo de 3 blocos. Na Figura 16(e), pode-se observar que a peça 1 foi deslocada para o espaço vazio objetivo, com custo de 3 blocos. Na Figura 16(f) é possível apenas observar o estado final do armazém após a movimentação das peças e a posição final da garra (laranja).

O custo em cada passo da Figura 16 (b), (c), (d) e (e) foi, respectivamente: $C_b = 4$, $C_c = 1$, $C_d = 3$ e $C_e = 3$. O custo total, C_t, dos deslocamentos realizados pela garra pode ser representado pelo somatório de todos eles. Assim, o custo total de deslocamento da garra, nesta tarefa de movimentação de duas peças foi de 11 blocos, ou seja, $C_t = 11$. Mesmo tendo valores adimensionais, o custo de deslocamento das peças levou a indagações como:

- O deslocamento das peças na vertical tem o mesmo peso que na horizontal?
- O deslocamento da garra sem peças tem o mesmo preço quando com peças?

Observando estas questões foram realizadas algumas considerações arbitrárias que deixaria o armazém com valores um pouco mais interessantes como:

- O valor de deslocamento vertical de uma peça teria 10% a mais de custo no bloco;
- O valor de deslocamento sem peças deveria ter 50% a menos que o valor do deslocamento com peças.

Estas questões foram avaliadas pensando na energia adicional que o motor deveria consumir a fim de levantar determinadas peças. Levando estes novos valores em consideração, pode-se prever novos valores para os custos em cada setor.

Na Figura 16(b) temos duas posições de deslocamento vertical e duas posições horizontais, gerando um custo de 2,2 vertical e 2 horizontal, ou seja, seria $C_b = 4{,}2$ porém o deslocamento é executado sem carregamento de carga, portanto seu peso deve ser reduzido pela metade, ou seja, o correto é $C_b = 2{,}1$.

Na Figura 16(c) o custo permanece o mesmo, $C_c = 1$, pois o deslocamento foi totalmente na horizontal e com peça. Já no deslocamento da Figura 16(d) há uma movimentação na vertical de custo 1,1, uma na horizontal de custo 2 e ambas sem carregamento de peça, desta forma $C_d = 1{,}55$.

Por fim, na Figura 16(e) tem-se apenas um deslocamento vertical com carga de 3 posições, ou seja, $C_e = 3{,}3$.

Com estas novas considerações de custo, totaliza-se $C_t = 7{,}95$. Essa abordagem permite ter uma análise mais realista da dinâmica da garra dentro dos possíveis deslocamentos em um armazém.

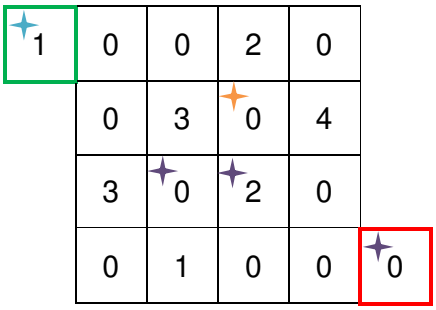
3.1.2 Dinâmica da garra na entrada e saída de peças do armazém

Outra consideração importante quanto à entrada e saídas de peça no armazém foi a adição de um custo extra, que é de cerca de um bloco no custo final do deslocamento e é adicionado com base nas seguintes considerações:

- Quando uma peça deve sair do armazém, esta acaba se deslocando para um bloco “invisível”, uma posição de saída que não está representada no armazém;
- Quando uma peça sai o robô, deve fazer uma série de ações como retirar a peça do último local do armazém, rotacionar a garra, colocar a peça no local “invisível” e posteriormente voltar à sua posição de frente ao armazém.

A Figura 17 demonstra a posição do bloco “invisível” do armazém.

Figura 17 – Armazém com posições invisíveis do armazém exemplo

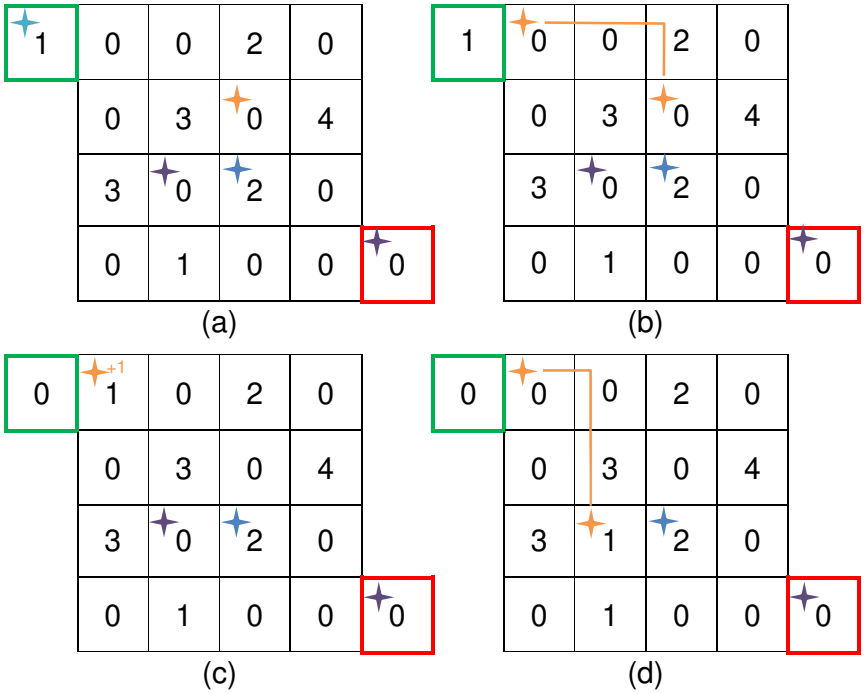


Fonte: Do autor (2019)

Pela Figura 17 pode-se observar que existe uma peça na posição inicial de entrada, representada por um contorno verde. Esta peça de entrada será posicionada no armazém na posição de endereço (2,1). Além disso, existe uma peça do tipo 2, localizada no endereço (2,2) que será colocada na posição de saída, representada pelo contorno vermelho. Lembrando que, na Figura 17, as peças com a marcação azul são posições de origem, as marcações roxas são as posições de destino e a marcação laranja é a posição atual da garra.

A Figura 18 a seguir apresenta as movimentações descritas sobre a inserção de uma peça no armazém. Na Figura 18(b) percebe-se que a garra se desloca da posição da garra (1,2) até a posição (0,0), $C_b = 1{,}55$. A partir daí, na Figura 18(c), a garra adiciona um custo extra indicando que pegou a peça, mas a posição da garra continua em (0,0), $C_c = 1$. Na Figura 18(d) a peça é deslocada até a posição destino (2,1), onde esta posição recebe a peça 1 que estava originalmente na entrada (posição invisível do armazém), $C_d = 3{,}2$. O custo total desta etapa de inserção no armazém, para este exemplo, é $C_{in} = 5{,}75$.

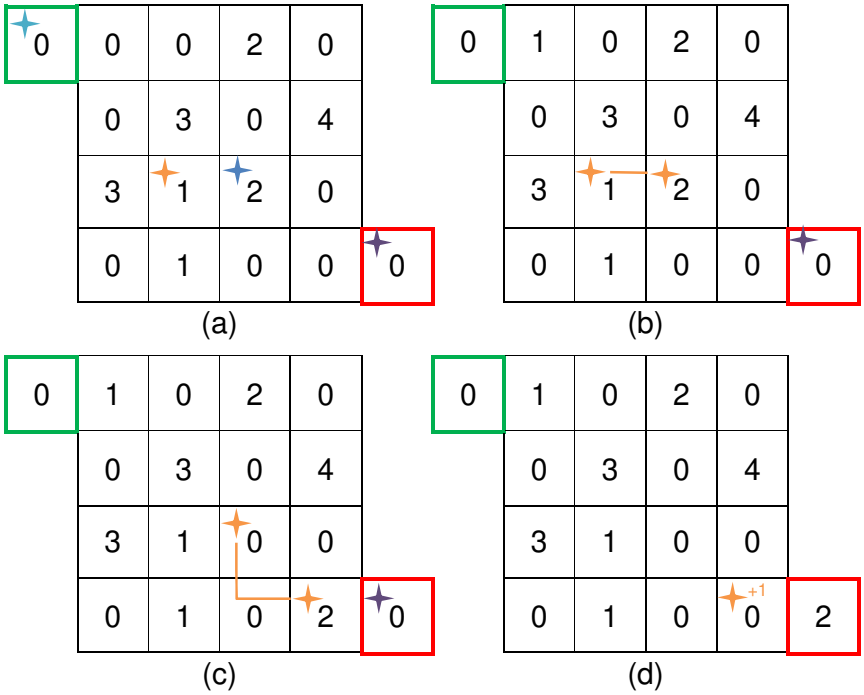
Figura 18 – Exemplo de inserção de uma peça do armazém



Fonte: Do autor (2019)

A Figura 19 apresenta as movimentações descritas sobre a retirada de uma peça do armazém.

Figura 19 – Exemplo de retirada de peças do armazém



Fonte: Do autor (2019)

Na Figura 19(b) percebe-se que a garra se desloca da posição inicial (2,1) até a posição (2,2), $C_b = 0{,}5$. Na Figura 19(c) a garra segura a peça do tipo 2 e a leva até a posição de endereço (3,3), mas ainda não é solta, $C_c = 2{,}2$. Na Figura 19(d) a peça é solta no bloco vermelho de saída do armazém, adicionando o peso 1 ao custo de deslocamento, $C_d = 1$. O custo total de saída do armazém é de $C_{out} = 3{,}7$

No total, a execução das duas tarefas resulta em um custo total de $C_t = 9{,}45$.

3.1.3 Organização do armazém e aplicação do algoritmo de Terry Winograd

A dinâmica do armazém presente neste trabalho irá incluir a organização do armazém, ou seja, de acordo com algum parâmetro ou referência, o armazém deverá conter determinadas peças em locais específicos, a fim de promover melhor distribuição das peças contidas no seu interior. Organizar o armazém significa melhorar a estrutura de armazenamento das peças de acordo com um parâmetro.

Quais serão estes parâmetros? Como o armazém sabe que deve ser organizado? A organização será necessária quando? Estas e outras indagações são respondidas ao longo deste trabalho e detalhadas de acordo com o algoritmo específico que tratará delas.

O que se pode argumentar até o momento é que a organização do armazém é uma decisão que o algoritmo deve tomar e não um comando dado pelo operador do armazém.

Para que o armazém seja organizado é necessário avaliar duas coisas: o estado atual do armazém e um estado referência de um armazém. A Figura 20 mostra em detalhes estes dois parâmetros.

Figura 20 – Estado atual e matriz de referência

0	0	2	0
0	3	0	4
3	0	2	0
0	1	0	0

(a) Estado atual

0	0	0	0
0	3	0	4
3	0	0	2
0	1	2	0

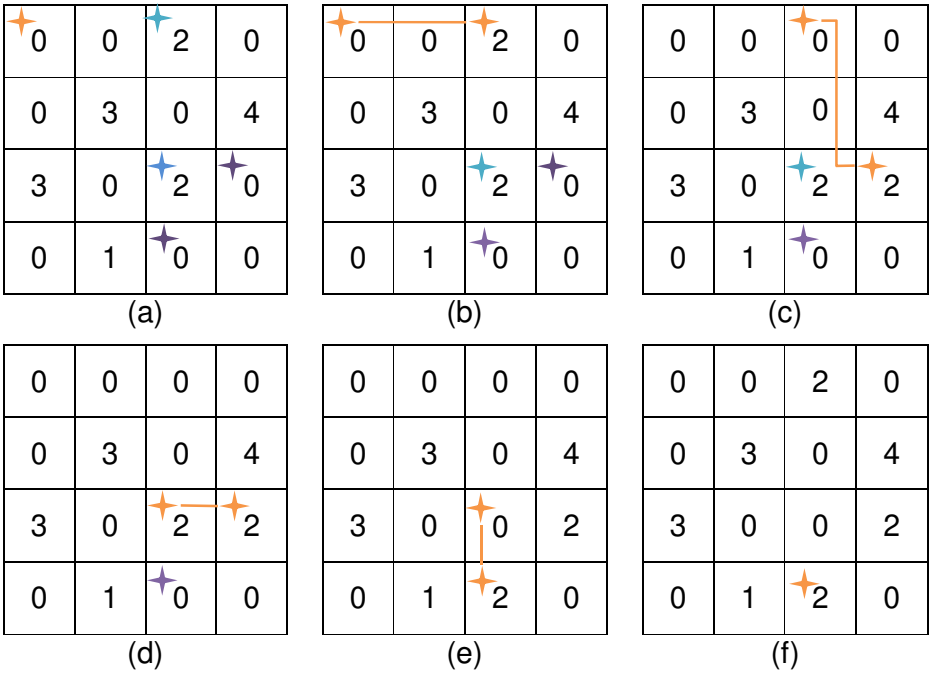
(b) Referência

Fonte: Do autor (2019)

A Figura 20(a) mostra o estado atual do armazém (aparentemente desorganizado). Além disso, é possível observar os pontos de origem (azul) e destino (roxo) das peças. Quem deu o parâmetro do destino das peças foi, justamente o armazém de referência na Figura 20(b). Observa-se, neste exemplo, que segundo a referência, todas as peças do tipo 2 deveriam estar mais próximas do ponto de saída.

A Figura 21 mostra os passos que a garra poderá utilizar para organizar o armazém de acordo com a referência.

Figura 21 – Organização simples de um armazém



Fonte: Do autor (2019)

Na Figura 21(a), tem-se o estado inicial do armazém e na Figura 21(f) é possível observar a referência final do armazém. Os custos parciais em cada uma das etapas de deslocamento da garra são: $C_b = 1$, $C_c = 3{,}2$, $C_d = 0{,}5$, $C_e = 1{,}1$. Desta forma, o custo total de organização do armazém é $C_{org} = 5{,}8$.

Através deste argumento, pode-se observar que a tarefa de organizar o armazém, exige um determinado custo, pois o movimento da garra para a organização do armazém, pode ser tão ou mais custoso que inserir ou retirar uma peça do armazém.

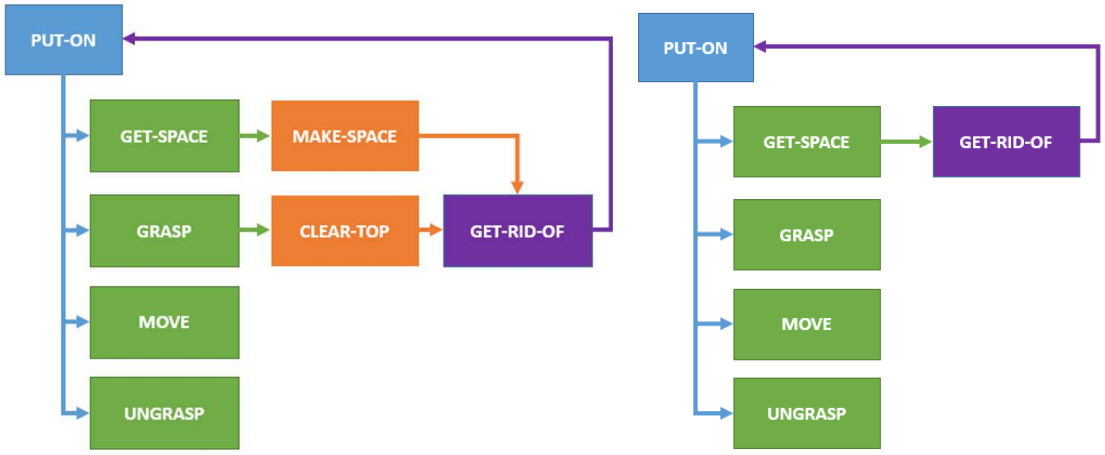
Um outo ponto importante para se tratar é a sobreposição de peças quanto ao seu destino. No exemplo acima, foi possível observar que as posições de destino

estavam vazias, mas e se estas mesmas posições estivessem sendo ocupadas por outras peças? Esta tratativa poderá ser solucionada com a técnica de solução de problemas de Terry Winograd.

No armazém, um determinado espaço só pode ocupar um único lugar, portanto, algumas mudanças serão realizadas neste algoritmo a fim de adaptá-lo à realidade do armazém.

A Figura 22(a) representa o algoritmo completo de Terry Winograd para a solução do problema *block's world* (mundo de blocos) enquanto que a Figura 22(b), o algoritmo foi adaptado para o armazém deste problema.

Figura 22 – Adaptação do algoritmo de Winograd para aplicação do armazém



(a) Alg. original (b) Alg. modificado

Fonte: Adaptado de Winston (1992)

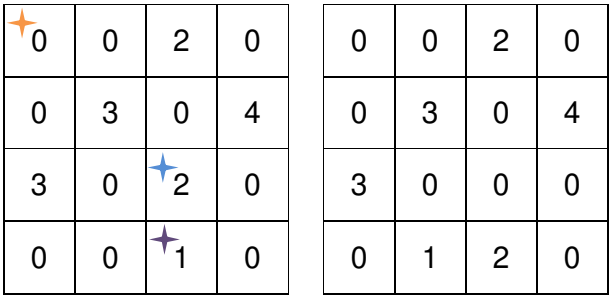
É possível observar que, na Figura 22(b), as funções *MAKE-SPACE* e *CLEAR-TOP* não estão mais presentes no algoritmo. A função *MAKE-SPACE* foi adaptada e integrada à função *GET-SPACE.* Desta forma, a função *GET-RID-OF* é chamada quando a função *GET-SPACE* realiza o teste de espaço na prateleira do armazém específica. A função *CLEAR-TOP* foi simplesmente excluída, pois nenhuma posição neste armazém poderá ter uma peça sobreposta no mesmo lugar.

É possível utilizar esse algoritmo em um exemplo semelhante de organização do armazém, contido na Figura 23, a fim de encontrar soluções, caso o destino em questão esteja sendo utilizado por outras peças.

A peça do tipo 2 disponível no endereço (2,2) deverá ser colocada no endereço (3,2), no entanto, o endereço está ocupado pela peça do tipo 1. Mesmo que a peça 1

não seja o objetivo principal da garra no primeiro instante, ela deverá ser movida de posição para dar espaço para a peça do tipo 2. Este tipo de ação pode ser visualizado na Figura 24.

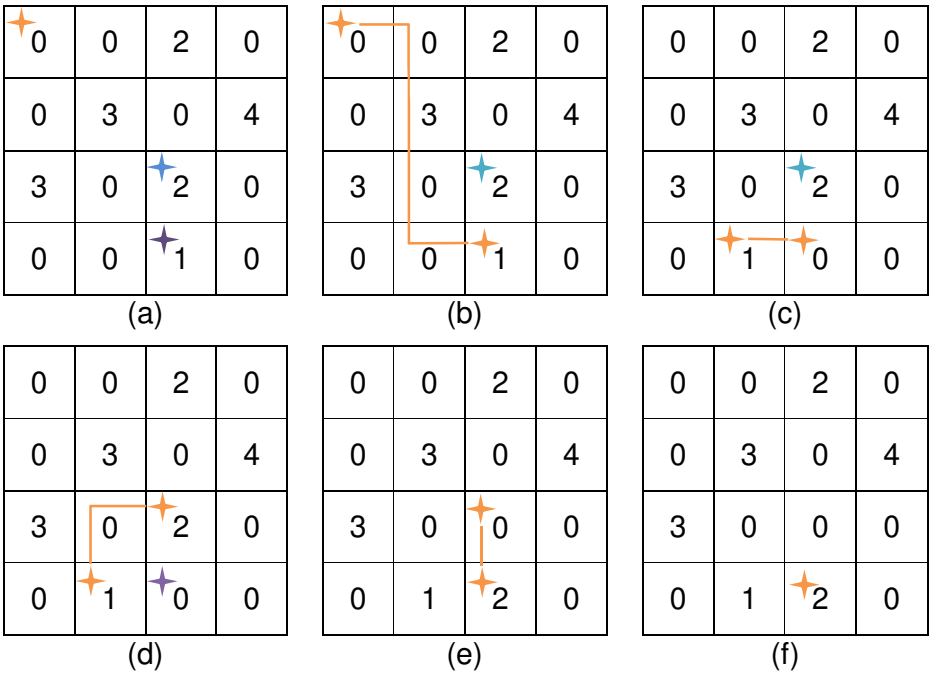
Figura 23 – Organização utilizando algoritmo adaptado de Terry Winograd



(a) Estado atual (b) Referência

Fonte: Do autor (2019)

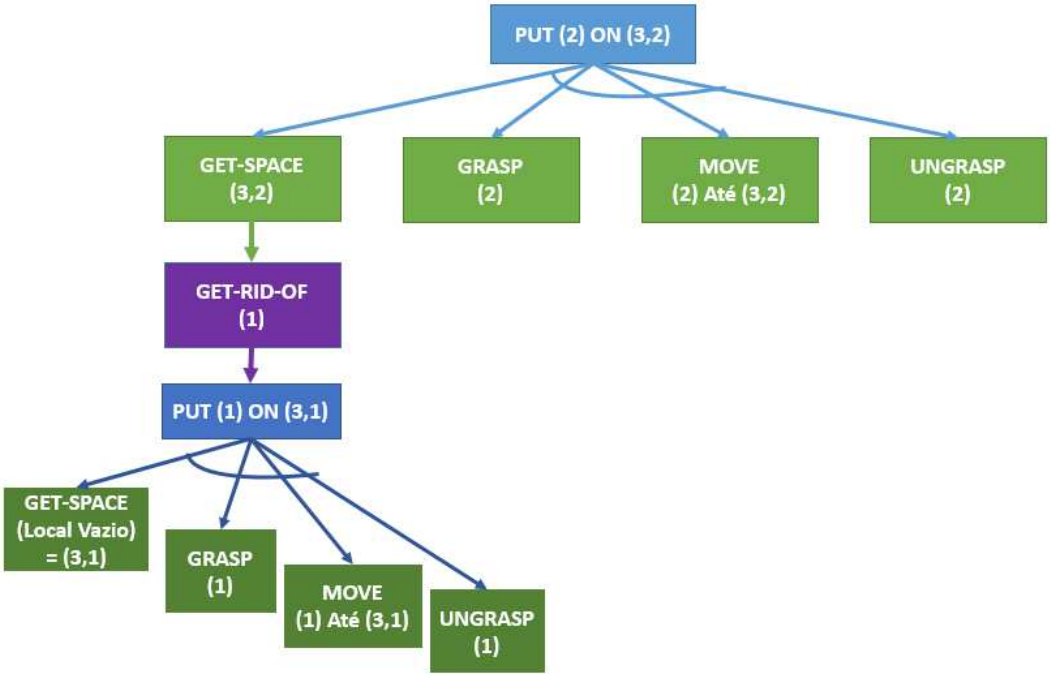
Figura 24 – Organização do armazém quando o endereço destino está ocupado



Fonte: Do autor (2019)

A Figura 24 mostra claramente que o algoritmo foi capaz de encontrar uma posição para a peça 1 antes de posicionar a peça 2 em seu destino correto. A Figura 25 mostra como estas ações foram organizadas segundo o algoritmo adaptado.

Figura 25 – Organização do armazém de acordo com o algoritmo proposto



Fonte: Do autor (2019)

Perceba que o algoritmo é recursivo, ou seja, as funções podem ser chamadas novamente caso as posições desejadas estejam ocupadas. Foi o caso da função *GET-RID-OF* que chamou novamente a função *PUT-ON* para inserir a peça 1 em uma posição vazia, neste caso, escolheu a posição de endereço (3,1). A função *MOVE* foi chamada duas vezes, uma para mover a peça 1 e outra vez para mover a peça 2. Da mesma forma as funções *UNGRASP*, *GRASP* e *GET-SPACE*.

É possível observar que este algoritmo utiliza uma linguagem simbólica. No entanto, pode-se questionar o seguinte: Por que a função *GET-SPACE* encontrou especificamente o endereço (3,1) vazio para se livrar da peça 1 mesmo sabendo que no armazém existem diversos outros endereços vazios?

Esta foi uma consideração avaliada no algoritmo, onde foi avaliado o menor custo de deslocamento das peças indesejadas, portanto, o algoritmo destina o endereço mais próximo (com menor custo).

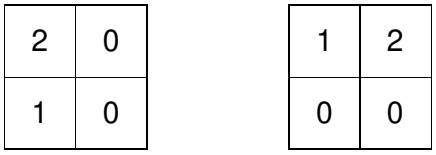
Outro ponto importante desta parte do algoritmo é como o estado atual se aproxima do estado do armazém? O que o algoritmo faz para chegar próximo ao armazém de referência?

Uma abordagem que foi tratada neste trabalho foi o uso de um segundo algoritmo que vai produzindo estados de armazém diferentes através do uso da

função *PUT-ON* de uma peça perante todos os demais endereços do armazém. O algoritmo avalia se todas as posições (endereços) contém as peças presentes no armazém de referência. Dentre todas as movimentações criadas pela função *PUT-ON*, a que mais se aproxima do armazém referência, utilizando o parâmetro de número de peças no local correto, irá ser o novo estado atual do armazém.

Para exemplificar este algoritmo, apresenta-se uma discussão acerca de um armazém um pouco menor, com apenas 4 posições, conforme visto na Figura 26.

Figura 26 – Exemplo armazém reduzido



(a) Estado atual (b) Referência

Fonte: Do autor (2019)

Na Figura 26(a) é possível observar o armazém atual enquanto na Figura 26(b), tem-se a referência. Antes de tudo, o armazém atual deverá movimentar as peças 1 e 2 de forma que chegue no armazém de referência. A cada movimento de uma das peças, será gerado um novo vizinho com as seguintes informações: número de posições corretas e custo do movimento.

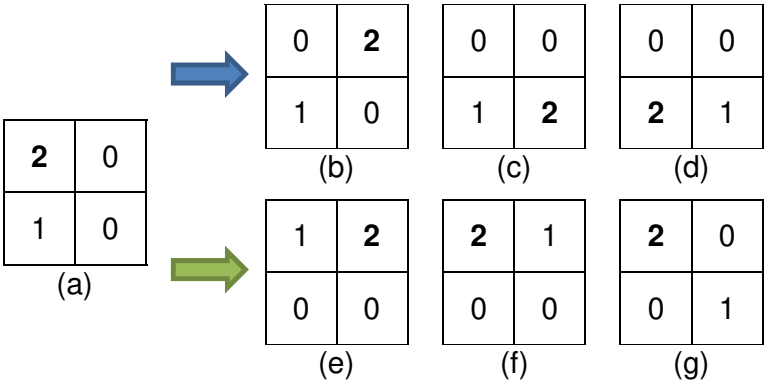
Inicialmente, a peça do tipo 2 dada no endereço (0,0) poderá ser deslocada para os demais endereços como (0,1), (1,0) e (1,1), enquanto a peça do tipo 1 dada no endereço (1,0), poderá ser deslocada para os endereços (0,0), (0,1) e (1,1). O movimento que estiver mais próximo da referência com menor custo de movimentação para tal, será o novo estado do armazém, possibilitando a garra realizar novos movimentos. A Figura 27 mostra os novos vizinhos produzidos pelas movimentações das peças 1 e 2 ao logo do armazém.

É possível observar na Figura 27, na flecha azul, todas as movimentações dadas para a peça do tipo 2 ao longo do armazém. Na Figura 27(d) a peça 1 teve de ser movimentada para dar lugar à peça do tipo 2 no endereço (1,0).

Já as matrizes representadas pela flecha verde representam os vizinhos produzidos pela movimentação da peça do tipo 1. Na Figura 27(e), a peça 2 é movimentada para dar lugar à peça do tipo 1 no endereço (0,0).

A Tabela 2 detalha as movimentações geradas desde o armazém original, englobando custos de movimento e organização do armazém, até chegar na sua posição de referência.

Figura 27 – Movimentação de peças no armazém e produção de vizinhos



Fonte: Do autor (2019)

Tabela 2 - Vizinhos gerados pelo movimento de peças em um armazém exemplo

Vizinho Armazém	Posições = referência	Custo de movimento
(a)	1	0
(b)	2	1
(c)	0	2,1
(d)	0	3,7
(e)	4	3,15
(f)	2	2,65
(g)	1	1,55

Fonte: Do autor (2019)

Através da Tabela 2, nota-se que o vizinho gerado na Figura 27(e) possui 4 posições iguais ao armazém de referência, ou seja, todas as peças do estado do armazém seguem o mesmo padrão da referência, sendo este o próximo vizinho ou próximo estado do armazém e consequentemente a resposta do sistema.

Através deste exemplo, é possível observar que quanto maior o número de peças, dentro do armazém e quanto maior for a estrutura física dele (tamanho da matriz) também será maior o número de possibilidades ou vizinhos produzidos pelos movimentos das peças em seu interior. O número de vizinhos, gerados a cada verificação da resposta é dado por

$$N_v = N_p\,(T_m - 1), \qquad (1)$$

sendo N_v o número de vizinhos que serão produzidos a cada movimento das peças no interior do armazém, N_p o número de peças não nulas dentro do armazém e T_m o número de endereços possíveis dentro do armazém.

A Equação (1), no entanto, não mostra todas as possibilidades de organização do armazém. Para descobrir isso, pode-se utilizar a permutação com eliminação de duplicatas, dada pela seguinte expressão:

$$B_o = \frac{T_m!}{B_{r_0}!\, B_{r_1}! \ldots B_{r_n}!}, \quad (2)$$

sendo que B_o é o total de possiblidades de organização do armazém, T_m é o tamanho do armazém e B_{r_i} é o número de endereços que contém estados repetidos da peça do tipo i (sendo B_{r_0} o número de casas vazias).

Considerando um armário com $T_m = 4$ e $B_{r_0} = 2$, onde apenas as posições nulas são repetidas (como as peças 1 e 2 não tem repetição não é necessário colocá-los na expressão). Deste modo, tem-se a partir da Equação (2) que o número de possíveis vizinhos neste armazém é $B_o = 12$.

O método de busca dado pelo parâmetro de posições semelhantes ao armazém referência, no entanto, produziu apenas 6 vizinhos, concluindo a sua busca antes mesmo de produzir todos os vizinhos possíveis. Esta análise é muito importante, pois em um armazém um pouco maior, por exemplo 5x5 contendo cerca de 15 peças diferentes e os outros demais espaços vazios, tem-se, a partir da Equação (2), $4{,}27 \times 10^{18}$ possibilidades.

Se o algoritmo de busca da solução tiver que passar por todas as possibilidades do sistema para encontrar a solução, certamente geraria um custo computacional muito grande e desnecessário, gerando também demora para encontrar a solução do problema. Por isso, é importante que o algoritmo escolhido para a busca de soluções devem ser rápidas e eficientes dentro do processo.

Uma técnica aplicada nesta seção foi utilizar a busca gulosa, que permite gerar vizinhos em um processo de busca e escolher aquele que aparentemente tem características mais semelhantes à solução esperada.

Até aqui, foi observado que o armazém possui três principais ações, sendo elas: inserir peças, retirar peças e organizar o armazém de acordo com uma determinada referência. Nas próximas seções serão discutidas algumas formas de gerar as referências do armazém.

3.2 CRITÉRIOS REFERENCIAIS DO ARMAZÉM

Quando um armário de roupas é organizado, tipos iguais de peças são colocados em prateleiras próximas ou na mesma prateleira. Um exemplo, seria colocar camisetas em uma prateleira, calças em outras, bermudas em outras, meias e roupas de baixo em gavetas separadas etc. Este tipo de organização facilita o trabalho de busca de uma determinada peça de roupa, o que é muito cômodo para o usuário que deseja retirar esta peça do armário.

Quando uma peça é inserida no armário, deve-se obedecer a este mesmo padrão, caso contrário, depois de um tempo, o armário perderia sua organização. Este tipo de abordagem é muito interessante quando os processos de inserção, retirada e organização de materiais é realizado de maneira manual. Ele pode ser aplicado para armazéns ou depósitos industriais. Em uma seção deve conter materiais de um tipo, outra seção de materiais de outro tipo e assim por diante. Mas será que o mesmo padrão deve ser aplicado para armazéns com uma inteligência artificial?

Imagine que agora as suas roupas dentro do guarda-roupa não estão mais à vista, pois o guarda-roupa se tornou uma imensa caixa preta. As peças que foram lavadas vão diretamente para o guarda-roupa automaticamente, você não sabe como elas foram guardadas lá dentro. Suponha que você deseja vestir uma camiseta azul e uma bermuda jeans. Você apenas seleciona em uma tela na frente do armário ou até mesmo por comando de voz as peças que deseja retirar do armazém de roupas. Depois do comando, suas roupas aparecem em uma gaveta específica para a retirada. O que muda no interior dos guarda-roupas nos dois casos citados? Qual a diferença entre a organização do guarda-roupa atual manual e um com mais tecnologia comandado por robôs?

A grande questão aqui é a organização. Um armazém que obtém retiradas de peças de forma manual, requer que o endereçamento destas peças seja realizado de maneira padrão. Pois não importa a peça em si, mas sim o seu tipo de classificação na retirada. Por exemplo, se quiser um par de meias nos guarda-roupas, basta procurar em sua gaveta específica. Não será, neste caso, realizada uma busca em todos as prateleiras até encontrar este mesmo par de meias. A classificação auxilia nesta busca quando o processo é realizado manualmente. Se os guarda-roupas não tivessem essa classificação seria necessário sempre gravar ou anotar a posição de uma roupa ao inserir ou retirar esta peça, o que particularmente daria muito trabalho.

A organização de um armazém automático não necessariamente precisa classificar as peças em prateleiras específicas, pois os controladores deste sistema poderiam armazenar os endereços específicos de cada uma das peças. Assim é feito hoje nos sistemas ASRS. O guarda-roupas automático funcionaria como um grande armazém industrial que teria cada peça de roupa com sua posição específica gravada no banco de dados do sistema.

Mesmo cada peça possuindo uma posição definida e gravada pelo sistema, não seria interessante armazená-los próximos um do outro para que o sistema em si fique mais organizado? Na verdade, não, pois basta dar um comando e a peça desejada aparece. O sistema irá traçar uma trajetória entre a posição atual do dispositivo de locomoção das peças, que aqui será chamado de garra do armazém ou apenas garra, até o objeto desejado. E assim, a garra irá trazer a peça solicitada até uma esteira ou posição final de saída de peças.

Se o armazém não precisa de uma classificação específica por tipo de materiais, não é necessário organizar o armazém? O estudo que se deseja realizar neste trabalho vem de encontro com esta pergunta. Identificar as vantagens de se organizar o armazém ou sistema ASRS entre os intervalos de retirada ou inserção de peças. Um dos principais parâmetros utilizados para avaliar a viabilidade deste sistema é o custo de deslocamento do armazém, ou seja, um estudo referente ao deslocamento horizontal e vertical da garra para realizar as 3 ações principais do sistema.

As subseções a seguir irão tratar alguns dos principais critérios para a organização do armazém: critério de prioridade de peças de saída e critério de avaliação do centro de massa.

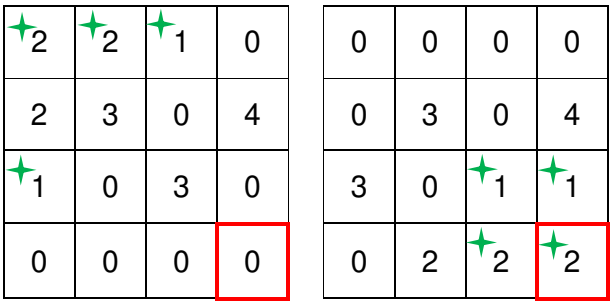
3.2.1 Critério de prioridade de peças de saída

Como já apresentado, a Indústria 4.0 terá um controle total sobre as peças, ferramentas e insumos que se deslocam dentro de um processo produtivo. Isso quer dizer que a requisição de peças para retirada do armazém é um parâmetro conhecido do sistema. Neste contexto, não é necessário que um determinado comando para a retirada de uma peça seja dado por um operador do ASRS e, sim, que o comando seja produzido automaticamente de acordo com as demandas do processo.

Diante desta explanação, será discutido um novo exemplo. Suponha que o armazém tem uma demanda de duas peças do tipo 2 e duas peças do tipo 1. Assim,

pode ser criado um vetor com prioridades de peças de saídas como o vetor $p_s = [2\ 2\ 1\ 1]$. Neste vetor, a primeira peça do tipo 2 tem maior prioridade, enquanto a segunda peça do tipo 2 tem a segunda maior prioridade, a primeira peça 1 tem a terceira maior prioridade e a outra peça do tipo 1 tem a menor prioridade. Diante do vetor de peças de saídas, pode-se analisar a Figura 28 avaliando qual das duas matrizes se torna mais eficaz no processo de retirada de peças.

Figura 28 – Exemplo de dois armazéns



(a) (b)

Fonte: Do autor (2019)

Na Figura 28, a posição de endereço (3,3) é onde as peças irão sair. Esta posição está marcada com o contorno vermelho. As peças com prioridade de saída, estão marcadas com estrelas verdes. Desta forma, é possível observar que o armazém representado na Figura 28(b) possui mais vantagem em relação ao armazém na Figura 28(a), pois as peças que sairão estão mais próximas do endereço de saída. Desta forma, a garra não deverá percorrer longas distâncias a fim de buscar as peças desejadas. A Tabela 3 mostra uma comparação entre os custos de deslocamento de saída de peças entre a Figura 28(a) e a Figura 28(b). Estes custos de deslocamento consideraram a posição inicial da garra no endereço (3,3).

Tabela 3 - Comparação entre custos de deslocamento entre dois armazéns

Armazém	Custo de deslocamento
(a)	35,5
(b)	10,3

Fonte: Do autor (2019)

Entre os dois exemplos da Figura 28, considerando a Tabela 3, seria mais interessante que um armazém de referência fosse dado pelo armazém da Tabela 3(b) ao invés do armazém da Tabela 3(a). Isso quer dizer que se o armazém fosse organizado, deveria seguir um padrão mais próximo da referência da Figura 28(b).

A partir do exposto pode-se concluir que um armazém de referência deve ser originado utilizando como informação um vetor de peças de saída.

Conforme as posições das prateleiras do armazém vão se afastando do endereço de saída de peças, mais custoso ficará o seu deslocamento. Desta forma é possível destacar no armazém a sua posição relativa de acordo com a prioridade das peças de saída (Figura 29).

Figura 29 – Alocação de peças de acordo com a sua prioridade de saída

15	14	12	9
13	11	8	5
10	7	4	2
6	3	1	0

Fonte: Do autor (2019)

Para o exemplo da imagem da Figura 29, a peça que tiver mais prioridade será alocada na posição de endereço (3,3), a peça que for a segunda com mais prioridade será alocada na posição de endereço (3,2), a peça que for a terceira com mais prioridade será alocada na posição de endereço (2,3) e assim por diante. Seguindo os números dados na matriz de alocação de peças de acordo com sua prioridade. Esta representação foi designada de matriz peso, pois foi detalhado o peso relativo à cada uma das posições do armazém.

Quando o algoritmo de organização de peças é gerado em relação à matriz de referência, é importante que um número limitado de peças receba prioridade, pois se todas as peças receberem algum tipo de prioridade, pode ser que todas as posições do armazém passem por algum tipo de mudança. Uma mudança radical em todas as posições do armazém pode trazer um custo de deslocamento da garra muito alto, não sendo vantajosa sua organização em um primeiro momento.

3.2.2 Critério de organização por centro de massa

Como visto até agora, são várias as formas que um armazém poderá ser organizado, seja pelo tipo de peça, por cor, por material, entre vários outros métodos de classificação. Um aspecto que será tratado aqui é uma abordagem em relação ao centro de massa do armazém. Imagine que o armazém possui uma estrutura na qual

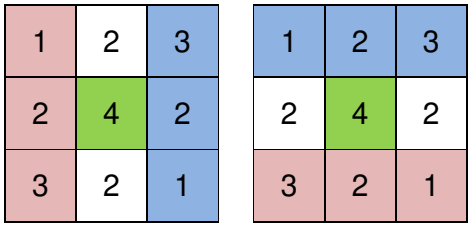
deva estar em equilíbrio, se existe um peso de 100 Kg do lado esquerdo no armazém, deve ter aproximadamente 100 Kg no lado direito do armazém.

Esta estratégia é interessante para não trazer danos a longo prazo à estrutura do armazém, tentando ao máximo dar um aproveitamento para endereços das prateleiras que porventura não tenham um índice de uso muito grande. No estudo de caso realizado neste trabalho, as localidades dos endereços (0,0) e (X-1,Y-1), lembrando que X e Y são os números de linhas e colunas do armazém, são as posições mais utilizadas, pois há um fluxo de peças maior devido à inserção e retiradas de peças. A ideia principal do critério de organização de centro de massa é propor um uso maior nos demais endereços do armário industrial.

Para tentar abordar este aspecto de aproveitamento do armazém, foi proposto inserir o centro de massa bem no seu endereço central. Desta forma, um armazém 3×3 terá seu centro de massa em (1,1). Da mesma forma, um armazém 5×5 terá seu centro de massa em (2, 2).

Para implementar este aspecto, é sugerido que cada tipo de peça tenha um determinado peso de valor adimensional, por exemplo, a peça do tipo 1 tem peso 1 e portanto é mais leve que a peça do tipo 2 que terá seu peso adimensional de valor 2. A Figura 30 mostra um exemplo da escolha do centro de massa em um armazém de dimensão 3×3.

Figura 30 – Armazéns 3×3 com centro de massa equilibrado



Fonte: Do autor (2019)

Observe que na Figura 30 que o centro de massa de ambos os armazéns é dado pelo bloco central com peso 4 e destacado pela cor verde. Este bloco não irá intervir no centro de massa, pois está localizado na posição de equilíbrio central do armazém. Além disso, na Figura 30(a) o somatório dos pesos dos objetos presentes na coluna azul é igual a 6. Da mesma forma, o somatório dos pesos dos objetos

presentes na coluna rosa também é igual a 6. Isso quer dizer que se pode calcular um peso resultante envolvendo os dois lados do armazém da seguinte forma:

$$P_c = \sum_{i=1}^{n} P_{d_i} - \sum_{i=1}^{n} P_{e_i} \ , \tag{3}$$

sendo que P_c é o peso resultante entre as colunas do lado direito e esquerdo do armazém, n é número total de peças do lado específico do armazém, P_{d_i} é o peso da peça de índice i à direita do armazém e P_{e_i} é o peso da peça i à esquerda do armazém. Assim, para o armazém da Figura 30(a) tem-se $P_c = 0$.

De forma análoga à análise do peso das colunas do armazém, o mesmo pode ser feito na Figura 30(b) analisando o peso em relação à distribuição de linhas. A linha azul tem um somatório de pesos iguais a 6 enquanto a linha rosa também tem um somatório ide pesos iguais a 6. Portanto, a expressão que calcula o equilíbrio entre os pesos resultantes das linhas do armazém pode ser descrita como:

$$P_l = \sum_{i=1}^{n} P_{a_i} - \sum_{i=1}^{n} P_{b_i} \tag{4}$$

sendo P_l o peso resultante entre as linhas da parte de cima e de baixo do armazém, P_{a_i} o peso da peça i na parte acima do centro de massa armário e P_{b_i} é o peso da peça i abaixo do centro de massa do armário. Assim, para o armazém da Figura 30(b) tem-se $P_l = 0$.

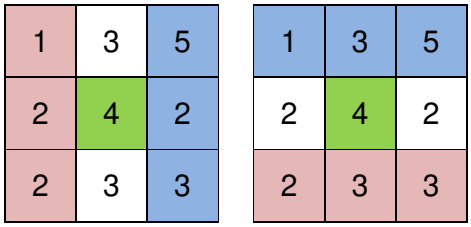
Ao final das análises de P_c e P_l, pode-se realizar um somatório destas duas variáveis afim de obter um peso resultante total,

$$P_r = \sqrt{{P_c}^2 + {P_l}^2} \tag{5}$$

Desta forma, tem-se que o peso resultante, P_r do armazém de exemplo da Figura 30 é igual a zero.

A mesma metodologia é aplicada agora a um exemplo de um armazém desequilibrado. A Figura 31 mostra um exemplo de armário, cujo peso das peças está distribuído de modo que o peso possa pender para um dos lados da matriz.

Figura 31 – Armazém 3×3 com centro de massa desequilibrado



(a) (b)

Fonte: Do autor (2019)

Utilizando as Equações (3), (4) e (5), para este exemplo chega-se ao valor de $P_r = 6$. Neste trabalho, o índice de P_r é utilizado para auxiliar na escolha do melhor armazém referência como um dos critérios para sua determinação.

Na seção a seguir é detalhado como utilizar os dois critérios aqui apresentados para a construção da matriz referência no armazém.

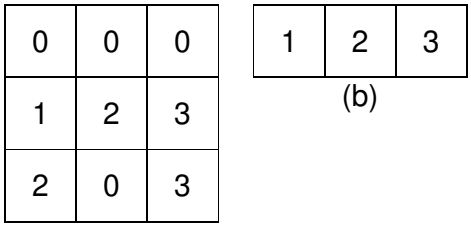
3.2.3 Critério de centro de massa com prioridade nas peças de saída

Na subseção 3.2.2, foi apresentado a importância de se utilizar um critério de organização do armazém segundo o seu centro de massa. No entanto, isso não é eficaz em relação à posição de saída de peças. Isso quer dizer que mesmo o armazém estando organizado de acordo com seu centro de massa, as peças mais necessárias podem estar longe da posição de saída de peças criando um prejuízo no que diz respeito ao deslocamento da garra no decorrer do armazém.

Para solucionar a questão apresentada, serão integrados os dois critérios de prioridade de peças de saída e centro de massa. Essa integração ocorrerá na forma de contagem de pontos. Como o critério de centro de massa busca encontrar o armazém ideal com equilíbrio próximo ao valor zero, busca-se neste novo critério de integração, aquele armário referência que contenha o menor índice de pontuação que será chamado aqui nesta seção de $Score$.

Se as peças com mais prioridade de saída estiverem em seu local correto, ou seja, respeitando a matriz peso, o $Score$ será subtraído de 0,1 a cada peça no local correto. Para detalhar um pouco melhor este critério integrado, será trazido um exemplo de uma aplicação que propõe uma nova matriz e uma nova lista de requisição de materiais. Essas informações podem ser visualizadas na Figura 32.

Figura 32 – Exemplo de armazém 3×3 e suas peças de saída



Fonte: Do autor (2019)

Na Figura 32(a) é possível observar a matriz 3×3 completamente desequilibrada e na Figura 32(b) tem-se a sequência de peças desejadas para a saída. Lembrando que o $Score$ pode ser substituído por 1, caso as peças desejadas estejam na posição correta de acordo com a matriz peso, ou seja, se a peça do tipo 1 estiver no endereço (2,2), a peça do tipo 2 estiver no endereço (2,1) e a peça do tipo 3 estiver no endereço (1,2).

Para encontrar a melhor referência, é necessário gerar vários armazéns modificando a posição das peças em seu interior. Para cada novo armazém gerado, deve-se calcular o resultante do centro de massa P_r e observar a localidade da lista de peças de saída. Desta forma, pode-se calcular

$$Score = P_r - 0{,}1N_c\,, \tag{6}$$

sendo N_c coresponde ao número de peças de saída que pertence a seu endereço correto de acordo com a sua prioridade na lista.

Para o exemplo da Figura 32 $N_c = 1$, pois a peça do tipo 3 está localizada no endereço (1,2) de acordo com a matriz peso. Para este exemplo, utilizando a Equação (6), com $P_r = 5{,}83$, obtém-se $Score = 5{,}73$.

Certamente, $Score = 5{,}73$ não é o índice mais baixo que se pode encontrar neste armazém com estas peças e com esta lista de saída. Com o uso de um algoritmo que contém estes dados são apresentados alguns resultados contendo as possíveis matrizes referência, o valor do índice de centro de massa, número de locais corretos e $Score$ final.

Para o armazém da Figura 32, tem-se um número de possíveis armazém-referência, segundo a Equação (2), $B_o = 3780$. E dentro destas possibilidades de armazéns de referência a melhor é dada pela vista na Figura 33.

Figura 33 – Melhor armazém de referência exemplo e seus índices

3	0	0
0	3	2
0	2	1

(a)

P_r	0
N_c	2
$Score$	-0,2

(b)

Fonte: Do autor (2019)

Se o estado atual do armazém for comparado com o armazém de referência, é possível calcular o custo de deslocamento de todas as peças do armazém até que ele encontre a referência estabelecida acima.

Comparando as duas matrizes, da Figura 32(a) e da Figura 33(a) é possível observar que todas as peças sofreram algum tipo de deslocamento. Neste sentido, pode-se concluir que o custo de deslocamento para a organização do armazém será muito grande. Se ao invés de deslocar todas as peças a fim de encontrar um armazém de referência, fosse deslocada apenas uma peça, geraria um custo menor de organização. Claro que é possível que, movendo apenas uma peça, o armazém não tenha um índice P_r igual a zero, mas pode promover uma organização significativa no armazém ao longo das tarefas que serão impostas no decorrer das movimentações de peças.

Critério de massa com prioridade de saída com movimento de uma única peça

Para diminuir o custo de organização do armazém, pode-se pensar em movimentar menos peças por vez, na busca de encontrar um armazém de referência mais barato em termos de movimentação. Desta forma, para encontrar este armazém referência, serão geradas todas as movimentações de todas as peças para as posições do armazém (sejam elas vazias ou não). No exemplo da Figura 32 tem-se 5 peças para nove espaços. Considerando que algumas peças são repetidas, tem-se que o número de possibilidades do armazém referência, levando em consideração apenas uma das peças é de 29 possibilidades. Dentre elas, a melhor referência em relação ao centro de massa pode ser vista na Figura 34.

Perceba que a organização realizou apenas a troca do item 3 no endereço (2,2) para (0,1). Desta forma, o custo de organização reduziu. No entanto, o índice P_r subiu de 0 para 1 em relação à melhor referência possível (Figura 33) e o $Score$ final obteve

0,9 ao invés de -0,2, pois apenas a peça 3 se encontra no local correto segundo a matriz peso, sendo assim, N_c é igual a 1.

Figura 34 – Melhor armazém de referência de centro de massa e seus índices

0	3	0
1	2	3
2	0	0

(a)

P_r	1
N_c	1
$Score$	0,9

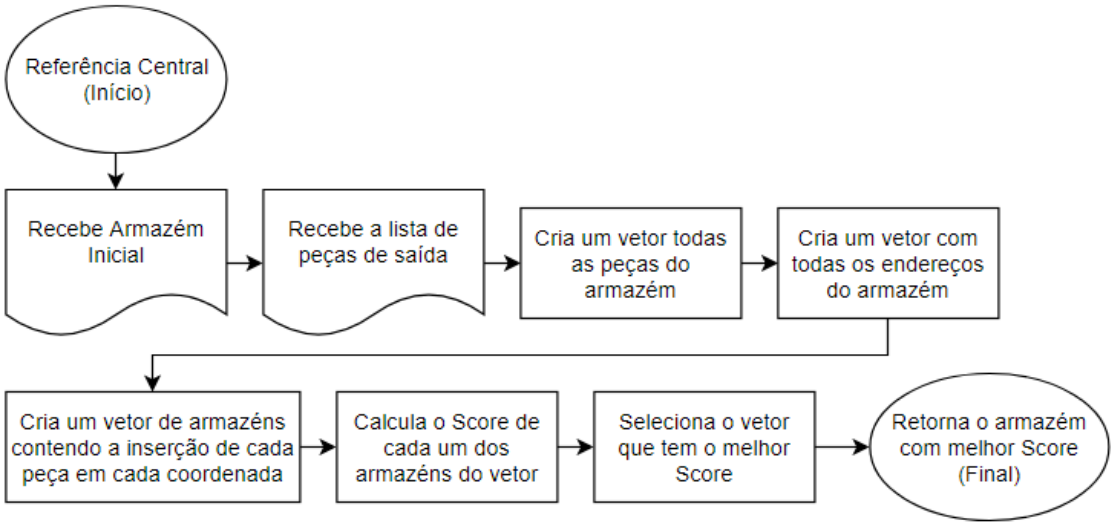
(b)

Fonte: Do autor (2019)

Este tipo de abordagem é interessante pois vai melhorando a organização do armazém a cada inserção ou retirada de peça. Não adiantaria gerar um custo grande para gerar o melhor armazém sendo que, ao adicionar ou retirar uma próxima peça, o índice de organização mudaria novamente.

O algoritmo que realiza esta ação pode ser visualizado na Figura 35, a seguir.

Figura 35 – Algoritmo de geração da referência central de organização



Fonte: Do autor (2019)

Com os critérios referenciais do armazém bem determinados, é possível iniciar uma análise de busca e geração de vizinhos, bem como escolher heurísticas que facilitem a dinâmica de uma inteligência artificial na busca por soluções para o armazém. Este tipo de análise é discutido no próximo capítulo.

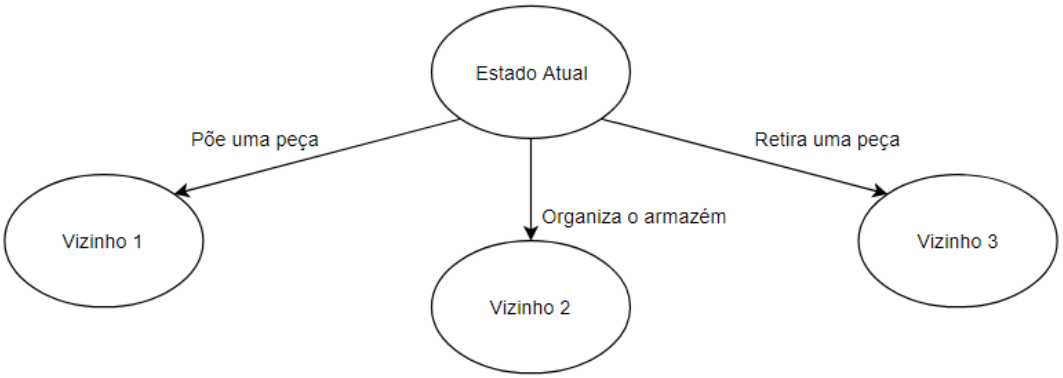
4 SOLUÇÕES PROPOSTAS

Com a dinâmica do armazém modelada, é possível realizar diferentes análises de busca utilizando os algoritmos de IA apresentados no Capítulo 2. Neste capítulo, é possível compreender algumas das soluções propostas levando em consideração a geração de vizinhos, técnicas de busca, algoritmos, heurísticas auxiliares e análise de resultados.

4.1 GERAÇÃO DE VIZINHOS

O possível vizinho no problema do armazém se dá pela movimentação de uma determinada peça, seja ela para a inserção de uma peça no armazém, a organização interna e/ou a saída de peças. A Figura 36 demonstra estas opções.

Figura 36 – Geração de Vizinhos



Fonte: Do autor (2019)

O movimento de uma peça irá impactar em várias variáveis como o estado do armazém, a posição da garra, o custo de movimento, a organização do armazém, número e lista de peças de entrada restantes, número e lista de peças de saída restantes, o índice de massa e $Score$ do armazém e o registro de ações que o armazém realizou.

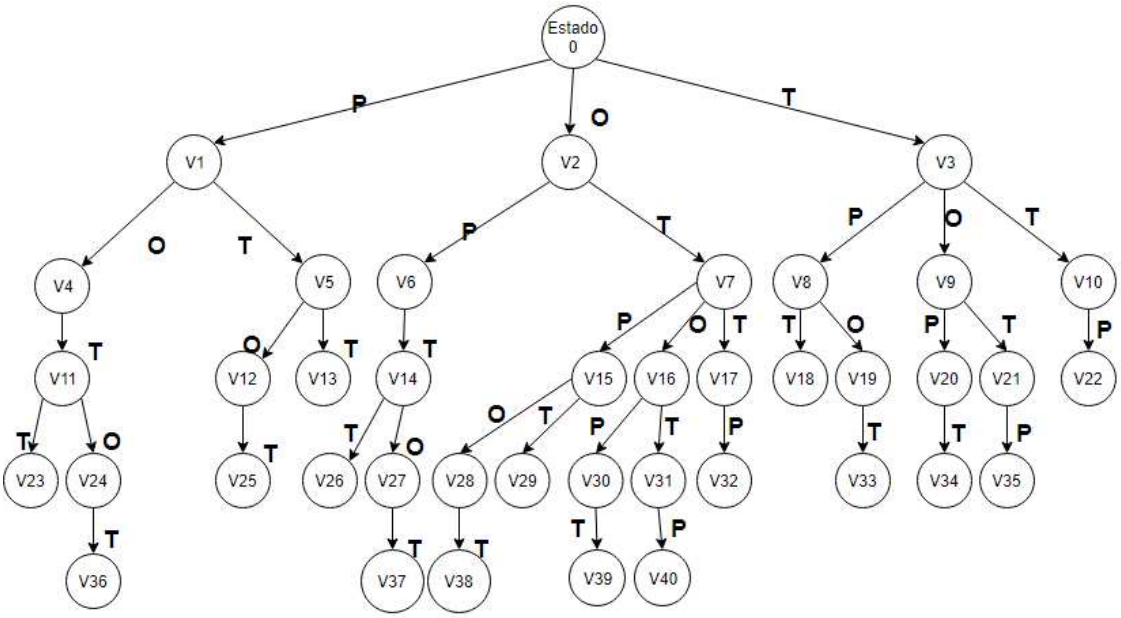
Desta forma, os vizinhos foram organizados em grandes vetores contendo cada uma destas variáveis. Quando um vizinho é gerado, automaticamente, cada uma destas variáveis é modificada nestes vizinhos. Na maioria dos códigos e algoritmos realizados neste trabalho, a geração de vizinhos se manteve a mesma, no entanto é importante pensar como estes vizinhos serão armazenados.

A geração de vizinhos possui uma determinada regra com relação à ordem de ação no armazém. Por exemplo, se o armazém decide por 1 peça e retirar 2 peças,

após a peça inserida, não é possível gerar novamente um vizinho que insere a peça no armazém. Da mesma forma que, após a segunda retirada de peças, não será possível gerar o vizinho da segunda retirada de peças. O mesmo fato será descrito para a ação de organizar. Após o armazém organizar uma única vez, este não poderá organizar novamente se uma peça não for retirada. Estes procedimentos foram tomados para que o armazém não realize muitas organizações a cada iteração.

No caso de uma inserção e retirada de duas peças, pode-se demonstrar a geração dos vizinhos de acordo com a Figura 37.

Figura 37 – Geração de vizinhos consideração P1 e T2



Fonte: Do autor (2019)

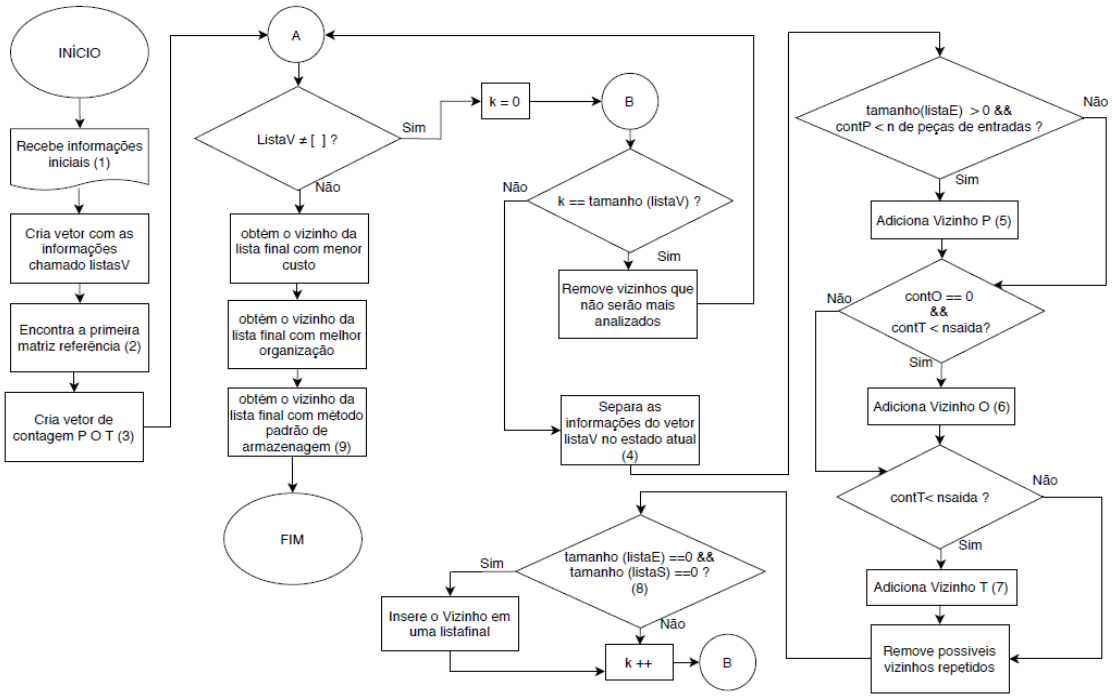
Observe na Figura 37 que foram gerados 40 vizinhos promovendo diferentes ordens de execução no armazém. Em cada folha, ou seja, ao término da execução da inserção de uma peça e retirada de duas peças, pode-se constatar diferentes sequência de ações. Cada tipo de ação irá gerar um *status*, por exemplo, se o vizinho gerado foi uma inserção, o *status* será P, se foi uma organização, o *status* será O e se foi uma retirada de peça, o *status* será T. Desta forma, nas folhas da geração de vizinhos, tem-se os seguintes *status* POTT, POTOT, PTOT, PTT, OPTT, OPTOT, OTPOT, OTPT, OTOPT, OTOTP, OTTP, TPT, TPOT, TOPT, TOTP e TTP, ou seja, são obtidos 16 diferentes métodos de inserir uma peça, organização (ou não) do armazém e retirar uma peça.

Como estes vizinhos serão armazenados e/ou como estes vizinhos serão utilizados dependerá da técnica de busca. A próxima seção trata o tema aplicando a técnica de busca no conjunto de ações do armazém.

4.2 TÉCNICA DE BUSCA APLICADA À VIZINHANÇA DO ARMAZÉM

Uma das primeiras técnicas de busca aplicada ao armazém foi a geração de todos os vizinhos possíveis, armazenando todos em uma lista. Todos os vizinhos folhas, ou seja, aqueles que completassem a tarefa são armazenados em uma lista final. Aquele vizinho-folha que apresentar o menor custo de deslocamento total ao final da execução será a solução do problema. Um fluxograma simplificado desta abordagem pode ser visualizado na Figura 38.

Figura 38 – Fluxograma simplificado da solução A



Fonte: Do autor (2019)

O código desenvolvido, do algoritmo visto na Figura 38, foi escrito na linguagem Python e está documentado no Apêndice A.

Na Figura 38, na fase identificada por (1), o algoritmo recebe às seguintes informações:

- Lista de peças de entradas (listaE);
- Lista de peças de saída (listaS);

- Estado original do armazém;
- Posição da garra;
- Status do armazém (‘P’, ‘O’ ou ‘T’).

Na fase identificada por (2), é chamada a função *encontra_ref*, que encontrar a matriz de referência que será utilizada para auxiliar no cálculo do $Score$. Em (3) é criado um vetor de 3 posições com a contagem de cada ação para cada vizinho, ou seja, quantas vezes o vizinho inseriu uma peça (contP), quantas vezes o vizinho organizou o armazém (contO) e quantas vezes o vizinho retirou uma peça (contT).

Na fase identificada por (4), todas as informações do vizinho são retiradas do vetor, para auxiliar o algoritmo na geração de novos vizinhos. Observe que cada vez mais o vetor listasV vai crescendo, pois a cada iteração são gerados mais 3 novos vizinhos. Cada vizinho é analisado. Em (5), ao adicionar a ação de inserir peça no armazém, são atualizados também outras variáveis, além de chamar a função *insere_melhor* para colocar a peça, *indice_score* para o cálculo do índice de custo de organização e $Score$, eliminação de uma peça da lista de entrada, adiciona ao status ‘P’, atualiza a posição da garra e calcula o custo de inserção da peça.

Em (6), ao adicionar a ação organizar, são atualizados também outras variáveis, além de chamar a função *referencia_central* para encontrar o melhor armazém, *busca_gulosa* para levar o armazém atual ao de referência, *indice_score* para o cálculo do índice de custo de organização e $Score$, adiciona ao *status* ‘O’, atualiza a posição da garra e calcula o custo de organização. Em (7), ao adicionar a ação de retirar peça no armazém, são atualizadas outras variáveis, além de chamar a função *tira_peca*, *indice_score* para o cálculo do índice de custo de organização e *score*, eliminação de uma peça da lista de saída do armazém, adiciona ao *status* ‘T’, atualiza a posição da garra e calcula o custo de retirada da peça. Por fim, em (8), é criada uma lista de vizinhos-folha, e conclui as tarefas de inserir e retirar peça.

No item G, da Figura 38 é possível observar são retirados 3 resultados:

1. O vizinho que ao final da execução das tarefas realizou o menor custo de movimento;
2. O vizinho que ao final da execução das tarefas apresentou o menor índice de organização;
3. O vizinho que se refere ao método padrão de organização do armazém, visto na fase (9).

O método de organização comumente utilizado em armazéns, não otimizado ou cotidiano, costuma inserir todas as peças desejadas no primeiro espaço vazio encontrado e depois retirar todas as peças da lista de saída. Este método comum é aplicado para se obter um ponto de comparação, de modo a verificar se a inteligência artificial inserida no sistema do armazém traz alguma vantagem na organização.

Os resultados derivados do algoritmo da Figura 38 e presente no Apêndice A são apresentados na Tabela 4 e na Tabela 5.

Tabela 4 – Dados iniciais de entrada no algoritmo solução A

Dados iniciais	Valores
pos_garra	[2,2]
ListaE	[1,2,3]
ListaS	[1,2,3]
Armazém	[1,0,2], [0,3,0], [3,0,2]

Fonte: Do autor (2019)

Tabela 5 – Resultados do código presente no Apêndice A

Índices	Menor custo de movimento	Menor $Score/P_r$	Método Padrão (sem IA)
Ordem	1	2	3
Armazém Final	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 0, 2] [2, 3, 3] [0, 0, 0]
Custo Movimento	19,6	20,1	21,6
pos_garra	[2,2]	[2,2]	[2,2]
status	TTPPPT	TTPPPOT	PPPTTT
$Score/P_r$	5,39	2,83	3,61

Fonte: Do autor (2019)

É possível observar na Tabela 5 que o vizinho de ordem 1 foi o que obteve menor resultado de custo, porém, seu $Score/P_r$ foi o pior encontrado. Além disso, observou-se que o vizinho de ordem 2 possui a melhor organização com índice $Score/P_r$ de 2,83, onde seu custo não teve tanta diferença em relação ao vizinho de ordem 1. O vizinho de ordem 3 no entanto, teve o maior custo de movimento, no entanto, seu índice $Score/P_r$ ficou próximo do melhor índice. Os *status* dos vizinhos de ordem 1 e 2 demonstram que, antes de realizar a última retirada de peça, o armazém sendo organizado uma vez já apresenta um índice $Score/P_r$ melhor do que se não fosse organizado nenhuma vez.

No algoritmo da Figura 38 são utilizadas algumas funções específicas para a realização da tarefa de gerenciamento do armazém, cada uma destas funções é detalhada nas subseções a seguir.

4.2.1 Função custo

A função custo tem como dados de entrada as variáveis das coordenadas inicial e final. A função calcula a distância entre o ponto A e o ponto B tanto na horizontal quanto na vertical. Na vertical é adicionado um custo de 10%. Ao final, a variável *dist* representa o custo de movimento. Veja o Algoritmo 1 em Python abaixo:

Algoritmo 1 - Função *custo* em Python

```
def custo(Ax, Bx):
   distl = abs(Bx[1] - Ax[1])
   distc = abs(Bx[0] - Ax[0])
   if distc>0:
      distc=distc*1.1
   dist = distl + distc
   return dist
```

Fonte: Do autor (2019)

4.2.2 Função *puton*

A função *puton* realiza o algoritmo de Terry Winograd e para isso conta com o auxílio de outras funções conhecidas como *get_space*, *get_rid_of*, *grasp*, *move* e *ungrasp*. Basicamente seus dados de entrada são as coordenadas inicial e final, a posição da garra e o objeto a ser carregado. Durante estas movimentações de objetos, os algoritmos chamam funções (como a função *custo*) para atualizar os parâmetros de custo do movimento. O código completo da função *puton* e suas subfunções estão disponíveis no Apêndice B.

A função *get_space* verifica se há um espaço vazio na coordenada desejada, caso contrário, chama a função *get_rid_of*. A função *get_rid_of* procura o espaço vazio com menor custo de deslocamento para colocar o bloco que está na coordenada desejada pela função *get_space*. Então, *get_rid_of* chama novamente a função *puton* para inserir a peça no espaço vazio. Basicamente a função *grasp* leva a garra até a posição do bloco que irá se deslocar no armazém. A função *move* retira a peça de sua coordenada original (inserindo o valor 0 em seu lugar) e leva para a coordenada desejada. A função *ungrasp* atualiza a posição da garra e o valor da peça na sua respectiva coordenada.

4.2.3 Função coordenadas

A função *coordenadas* tem como dado do entrada o estado atual do armazém e retorna um número de possíveis vizinhos, coordenadas dos objetos do armazém e as coordenadas dos espaços vazios. Essa função ajuda a função *busca_gulosa* em sua tarefa de encontrar possíveis armazéns a partir do armazém referência.

Algoritmo 2 - Função *coordenadas* em Python

```
def coordenadas(matriz_base):
  esp_vazios=0
  coord_vazio=[]
  coord_objetos=[]
  coord_possiveis=[]
  for i in range(0, tama):
    for j in range(0, tama):
      coord_possiveis+=[[i,j]]
      if matriz_base[i][j]==0:
        esp_vazios=esp_vazios+1
      else:
        coord_objetos+=[[i,j]]
  num_objetos=tama*tama - esp_vazios
  pvizinhos=(tama*tama - 1)*num_objetos
  return coord_objetos,coord_possiveis,pvizinhos
```

Fonte: Do autor (2019)

4.2.4 Função *saipeca*

A função *saipeca* (Algoritmo 3) auxilia a função *tira_peca* a inserir o valor 0 nas coordenadas onde uma determinada peça foi retirada.

Algoritmo 3 - Função *saipeca* em Python

```
def saipeca(coordenadas, matriz):
  if matriz[coordenadas[0]][coordenadas[1]]==0:
    print("A posição do armazém está vazia!")
  else:
    matriz[coordenadas[0]][coordenadas[1]]=0
  return matriz
```

Fonte: Do autor (2019)

4.2.5 Função *matrizpeso*

A função *matrizpeso* recebe o tamanho do armazém em questão e retorna a matriz com os pesos de cada posição em relação à coordenada de saída de peças. Além disso, irá retornar uma lista com estes pesos e um vetor com a coordenada de cada peso. Esta função auxilia o código principal e a função *encontra_ref* a verificar se as peças de saída estão próximas às coordenadas menos custosas do armazém,

obtendo o valor de peças corretas, N_c, para contabilizar o $Score$ do armazém em questão. A função pode ser visualizada no Algoritmo 4.

Algoritmo 4 - Função *matrizpeso* em Python

```
def matrizpeso(tama):
  matriz=cria_matriz(tama,tama,0)
  lista=[]
  lista2=[]
  lista3=[]
  novalista=[]
  for i in range(0,tama):
    for j in range(0,tama):
      matriz[i][j]=custo([i,j],[tama-1,tama-1])
      lista+=[custo([i,j],[tama-1,tama-1])]
      lista.sort()
      lista2+=[[i,j]]
      lista3+=[(i**2 + j**2)**0.5]
  for i in range(0, len(lista2)):
    novalista+=[lista2[lista3.index(max(lista3))]]
    lista3.pop(lista3.index(max(lista3)))
    lista2.remove(novalista[i])
  return matriz, lista, novalista
```

Fonte: Do autor (2019)

4.2.6 Função *proxima_vazia*

A função *proxima_vazia* (Algoritmo 5) auxilia a função *encontra_ref* a encontrar uma próxima coordenada vazia e retorna apenas uma coordenada.

Algoritmo 5 - Função *proxima_vazia* em Python

```
def proxima_vazia(matriz):
  for i in range(0, tama):
    for j in range(0, tama):
      if matriz[j][i]==0:
        pcoordenada=[j,i]
  return pcoordenada
```

Fonte: Do autor (2019)

4.2.7 Função *encontra_ref*

A função *encontra_ref* é uma das principais funções do código principal. Ela recebe o estado atual e uma lista de peças de saída, retornando uma matriz com as peças de saída mais próximas da coordenada de saída de peças. Essa matriz resultante irá auxiliar posteriormente o código principal e a função *referência_central* a calcular o $Score$ do vizinho em questão. Esta função está no Apêndice C.

4.2.8 Função *tira_peca*

A função *tira_peca* também é uma das principais funções do código principal. Esta função recebe os parâmetros de estado do armazém, lista de peças de saída, posição da garra e custo de movimento atual do armazém realizado por tarefas anteriores. Esta função ajuda o programa principal a gerar um vizinho que realizou a tarefa de retirada de peças, ou seja, gera um vizinho com uma peça a menos de acordo com a ordem de retirada de peça levando em consideração a lista de saída de peças, além disso, calcula seu custo de movimento de retirada de peça. A função *tira_peca* pode ser visualizada no Apêndice D.

4.2.9 Função *busca_gulosa*

A função *busca_gulosa* recebe os parâmetros de matriz de referência gerado pela função *referência_central*, o estado do armazém para o determinado vizinho atual em questão, a posição da garra e o custo de movimento acumulado até o momento. Ela movimenta todas as peças do armazém a fim de encontrar um armazém igual ao armazém de referência.

Esta função usa a busca gulosa, pois é comparado o número de peças do armazém que estão na mesma posição do armazém referência. Para cada movimentação das peças são gerados vizinhos com um índice de organização e quem tiver o melhor índice de organização será o próximo vizinho. As peças continuam a ser movimentadas até que o estado da matriz seja a solução do problema, ou seja, a matriz referência. Esta função auxilia o processo de organização do armazém e obtém como resposta o estado do armazém organizado de acordo com a referência, a posição da garra final e o custo acumulado após o processo de movimentação de peças. A função em Python é encontrada no Apêndice E.

4.2.10 Função *entrada_de_pecas*

A função *entrada_de_pecas* auxilia a função *insere_melhor* a colocar uma peça que está na lista de entrada no armazém. Esta função tem como parâmetros de entrada as coordenadas disponíveis no armazém, o estado do armazém do vizinho atual, o custo acumulado do vizinho até o momento, a posição da garra e a peça a ser inserida no armazém. Esta função insere a peça em todas as coordenadas possíveis que estão livres. Como resposta, a função entrega um vetor com estes possíveis movimentos para esta única peça. A função em Python está no Algoritmo 6.

Algoritmo 6 - Função *entrada_de_pecas* em Python

```
def entrada_de_pecas(e_disponiveis, estado_armazem_aux, custo_aux,
posicaogarra_aux, pecaX, vizinhosCP):
    gin=[0,0]
    for i in range(0, len(e_disponiveis)):
        daux=copy.deepcopy(e_disponiveis[i])
        estado_armazem_aux[daux[0]][daux[1]]=pecaX
        custoentrada = custo(gin, daux) + 1 + custo_aux +
custo(posicaogarra_aux, gin)*pesodesloca

vizinhosCP+=[[copy.deepcopy(estado_armazem_aux),custoentrada,
daux]]
        estado_armazem_aux[daux[0]][daux[1]] = 0
    return vizinhosCP
```

Fonte: Do autor (2019)

4.2.11 Função *custo_cm*

A função *custo_cm* auxilia a função *indice_score* a encontrar o índice de massa de um determinado armazém. Como parâmetros de entrada, esta função só recebe uma matriz e, como resposta, a função retorna um valor referente ao índice de massa desta matriz. Este dado irá auxiliar a função a encontrar também o *score* de um determinado vizinho durante a execução do algoritmo. A função *custo_cm* pode ser encontrado no Apêndice F.

4.2.12 Função *referencia_central*

A função *referencia_central* é uma das principais funções do código que tem o papel de encontrar a melhor referência a ser seguida por um determinado armazém, levando em consideração o índice de massa e o $Score$ de um determinado vizinho. Esta função pode ser encontrada no Apêndice G.

4.2.13 Função *insere_melhor*

A função *insere_melhor* (Algoritmo 7) analisa um vetor contendo armazéns que realizaram a inserção de uma peça. A função, então, escolhe o que possui menor custo de inserção de peça. Esta função retorna um armazém com custo de movimento e a posição final da garra.

Algoritmo 7 - Função *insere_melhor* em Python

```
def insere_melhor(estado_armazem_auxiliar,listaentrada, enter_cost,
posicaogarra_auxiliar):
   e_avaiable = estados_vazios(estado_armazem_auxiliar)
   pecaentrada = listaentrada[0]
   vizinhosEP = entrada_de_pecas(e_avaiable,
estado_armazem_auxiliar, enter_cost, posicaogarra_auxiliar,
pecaentrada,[])
   listaentrada.pop(0)
   custovizinho = []
   for i in range(0, copy.deepcopy(len(vizinhosEP))):
     custovizinho += [vizinhosEP[i][1]]
   indicemc = custovizinho.index(min(custovizinho))
   return vizinhosEP[indicemc]
```

Fonte: Do autor (2019)

4.3 EQUILÍBRIO ENTRE CUSTO DE MOVIMENTO E O ÍNDICE P_r

A solução de técnica de busca demonstrada na Figura 38 mostra soluções para a organização do armazém, pois estrutura de forma sólida os caminhos e funções necessárias para que o algoritmo tome diversas decisões com base no custo ou $Score$ do algoritmo. No entanto, observou-se pela tabela de resultados que nem sempre custo e organização andam juntos, pois o armazém com menor custo nem sempre é o mais organizado.

A proposta desta seção é aprimorar o algoritmo apresentado na Figura 38 levando em consideração um indicador de equilíbrio entre organização e custo. Para isso, foi pensando em criar um indicador adimensional para custo e um indicador adimensional para P_r. Entre estes indicadores poderiam ser gerados uma média que fosse o principal indicativo de custo e organização do armazém. Com base na média dos indicadores, poderia modificar parte do algoritmo da Figura 38 realizando buscas mais direcionadas a esta nova abordagem.

Para gerar o indicador de custo, deve-se buscar uma referência de custo, da mesma forma que para gerar o indicador de P_r, deve-se buscar uma referência de P_r. A referência de custo foi realizada com base no número de peças de entradas e no número de peças de saídas, ou seja,

$$C_{ref} = 2\left(N_{pe} + N_{ps}\right), \tag{7}$$

sendo que, C_{ref} é a referência do custo, N_{pe} é o número de peças de entrada e N_{ps} é o número de peças de saída. O multiplicativo “2” vem da ação de inserção ou retirada de peças, visto que o custo mínimo para realizar qualquer uma destas ações é 2.

Para criar a referência de P_r, levou-se em consideração o tamanho do armazém. Desta forma, a referência de P_r é dada pela seguinte expressão:

$$P_{r_{ref}} = T_m - 1 \tag{8}$$

onde, $P_{r_{ref}}$ é a referência de P_r e T_m é o tamanho do armazém levando em consideração o seu número de posições. O tamanho do armazém foi subtraído pelo número 1, pois cada posição poderá contribuir com no mínimo o indicador 1, com exceção da peça que estiver no centro de massa do armazém.

Com as referências estabelecidas, é possível calcular os indicadores de custo P_r, e assim, encontrar a médias entre estes dois parâmetros. Tem-se:

$$C_{ind} = \frac{custo}{C_{ref}}, \tag{9}$$

$$P_{r_{ind}} = \frac{P_r}{P_{r_{ref}}}, \tag{10}$$

$$G = \frac{C_{ind} + P_{r_{ind}}}{2}. \tag{11}$$

sendo G a média geral um parâmetro que leva em consideração os indicadores de custo e P_r na mesma proporção. Dependendo do tipo de abordagem, pode-se pensar em uma G utilizando a média ponderada.

Para que o código possa ser realizado levando em consideração estas médias, é imprescindível que para cada um dos novos vizinhos sejam calculados todos estes parâmetros. Desta forma o vetor de cada vizinho conterá o estado do armazém, o custo de movimento, a posição da garra final, um vetor com as contagens das execuções de P, O e T, o *status* “POT”, o vetor com peças da lista de entrada, o vetor com peças da lista de saída, o índice de organização P_r, o $Score$ e a média geral G.

Com este novo parâmetro é proposta uma solução B, onde é possível encontrar o vizinho com a melhor média geral. O código deste algoritmo está disponível no Apêndice H. Considerando os mesmos valores de entrada para a solução A (Figura

38), pode-se aplicar a técnica presente no Apêndice H e, desta forma, coletar os resultados presentes na Tabela 6.

Tabela 6 - Resultados do código presente no Apêndice H

Índices	Menor custo de movimento	Menor $Score/P_r$	Método Padrão (sem IA)	Menor G
Ordem	1	2	3	4
Armazém Final	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 0, 2] [2, 3, 3] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]
Custo Movimento	19,6	20,1	21,6	20,1
pos_garra	[2,2]	[2,2]	[2,2]	[2,2]
status	TTPPPT	TTPPPOT	PPPTTT	TTPPPOT
$Score/P_r$	5,39	2,83	3,61	2,83
G	1,15	1,01	1,13	1,01

Fonte: Do autor (2019)

Observando-se os dados apresentados na Tabela 6, é possível identificar que a solução de ordem 2 se igualou a solução de ordem 4. Isso quer dizer que realizando algumas pequenas organizações no armazém, já se melhora o índice P_r e, por sua vez a própria média geral do sistema. Se for levar em consideração as soluções de ordem 4 em relação à ordem 1, tem-se que o custo de movimento teve um pequeno acréscimo de 2,55% enquanto o índice P_r obteve uma queda de 47,5%, além de que a média geral também é cerca de 12,2% menor.

4.4 BUSCA GULOSA NA ESCOLHA DOS VIZINHOS POT

De maneira geral, as soluções A e B apresentadas, respectivamente nas Seções 4.2 e 4.3, apresentam todas as possibilidades de movimento salvando todos os vizinhos gerados na execução do algoritmo.

Uma forma interessante de trabalhar este problema é utilizando a busca gulosa, analisando a melhor média geral dos vizinhos gerados pelos movimentos de inserção, organização e/ou retirada de peças, ou seja, os vizinhos POT. Do estado atual, geram-se 3 vizinhos, cujos *status* podem ser P, O ou T, aquele que possuir a menor média geral será o novo estado atual. Desta forma não é necessário armazenar os demais vizinhos, evitando a geração de inúmeros vizinhos que não serão levados em consideração na resposta final.

O algoritmo que realiza a busca gulosa para os vizinhos POT pode ser encontrada no Apêndice I. Os resultados deste algoritmo levaram em consideração os mesmos parâmetros de entrada das soluções A e B. Estes resultados podem ser visualizados na Tabela 7.

Tabela 7 – Avaliando resultados da busca gulosa aplicada aos vizinhos POT

Índices	Menor custo de movimento	Menor $Score/P_r$	Método Padrão (sem IA)	Busca Gulosa com G
Ordem	1	2	3	4
Armazém Final	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 0, 2] [2, 3, 3] [0, 0, 0]	[1, 2, 2] [3, 3, 0] [0, 0, 0]
Custo Movimento	19,6	20,1	21,6	22,3
pos_garra	[2,2]	[2,2]	[2,2]	[2,2]
status	TTPPPT	TTPPPOT	PPPTTT	OPTOTOTPP
$Score/P_r$	5,39	2,83	3,61	5,39
G	1,15	1,01	1,13	1,27

Fonte: Do autor (2019)

Pela Tabela 7 observa-se que a busca gulosa aplicada à cada vizinho gerado pelo movimento P, O ou T não gera uma solução ótima. Na realidade, buscar pela melhor G logo na geração dos primeiros vizinhos não significa que o caminho percorrido total será o menor. Verifica-se que o algoritmo optou por organizar o armazém mais de 3 vezes, pois naquele momento organizar tinha um custo menor, no entanto, somando-se estes custos adicionais de organização, o caminho ficou mais custoso.

Comparando a melhor média G (ordem 2) com a técnica de busca gulosa (ordem 4), pode-se observar que o custo de movimento ficou 9,9% a mais, o índice P_r teve um crescimento de 90,5% e a média geral obteve um crescimento de 25,7%. Comparando os resultados do armazém sem inteligência artificial (ordem 3) com o composto por busca gulosa imediatista (ordem 4), verifica-se que a técnica de busca gulosa não apresenta nenhuma vantagem, apresentando resultados menos favoráveis inclusive no índice de organização do armazém.

Estes resultados são interessantes, pois mostram que não basta organizar o armazém a todo instante, mas sim, organizar o armazém no momento certo.

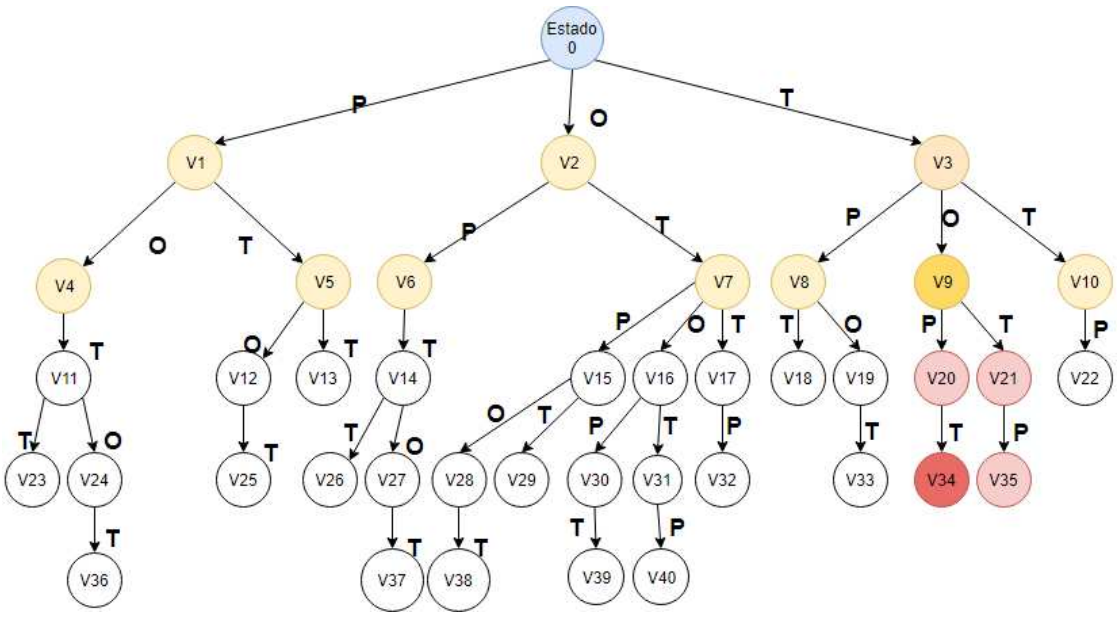
A próxima seção trabalha com a busca gulosa considerando uma abordagem um pouco diferente.

4.5 BUSCA GULOSA COM ÍNDICE DE BUSCA

Os resultados presentes na Seção 4.4 não foram satisfatórios, chegando à conclusão de que organizar o armazém apenas pelo fato de ser a opção mais viável no momento não significa que esta organização terá um impacto positivo no custo e organização geral do armazém. Na Seção 4.4, são sempre gerados apenas 3 vizinhos, escolhe-se um deles e exclui-se os demais, o que limita muito uma visão global do sistema.

Esta seção busca encontrar um equilíbrio entre as Seções 4.3 e 4.4, para entregar uma resposta rápida com bons índices de custo, P_r e média geral. Para encontrar este equilíbrio, é utilizada a técnica de busca com um número de vizinhos maior do que apenas três. A ideia principal é criar um índice de busca que gere vizinhos até um determinado nível da árvore de decisão. Os vizinhos mais próximos dos estados folhas serão analisados e aquele que tiver a melhor G será o escolhido. Desta forma, é possível ter um campo de visão maior sobre os processos do armazém tendendo a realizar uma decisão mais assertiva entre um número maior de opções. Basicamente, é como se o algoritmo pudesse visualizar alguns passos à frente. A Figura 39 demonstra esta nova abordagem.

Figura 39 – Exemplo da aplicação de busca gulosa com índice de busca $z = 2$



Fonte: Do autor (2019)

Considerando um armazém que tem duas peças retiradas e uma peça inserida, é possível detalhar a árvore de decisão da Figura 39 e ainda observar os diferentes níveis da árvore (variável z). O nível $z = 0$ representa a condição inicial do armazém. O nível $z = 1$ possui 3 estados, por sua vez, o nível $z = 2$ possui 7 estados e assim por diante.

A busca gulosa da Seção 4.4 verificava o armazém com um índice de busca igual a 1, ou seja, cada iteração de geração de vizinhos por vez e desta forma só analisava 3 vizinhos por vez. A Figura 39 mostra que é possível elevar os níveis de busca. Se utilizar o nível de busca igual a 2, o algoritmo não irá escolher apenas entre os vizinhos V1, V2 e/ou V3, mas sim escolher entre V4, V5, V6, V7, V8, V9 e/ou V10. No exemplo hipotético, dado pela Figura 39, o algoritmo escolheu o vizinho V9 que representa a sequência de operação “TO”. Depois é gerado um novo nível de vizinhos dado por V20 e V21. Como o índice de busca é igual a 2, o nível de busca irá descer mais um nível gerando os vizinhos V34 e V35. De forma hipotética, o algoritmo escolheu o vizinho V34 dado pela sequência de operação “TOPT”.

Perceba na Figura 39 que todos os vizinhos dados pela cor branca não foram analisados ou sequer gerados pelo algoritmo. Aumentando o nível busca do algoritmo, mais níveis são gerados e por sua vez mais vizinhos entram na análise da técnica de busca. O algoritmo em Python que realiza a técnica de busca com índice de busca é dado no Apêndice J. Na Tabela 8 são apresentados os resultados do algoritmo de busca gulosa com índice de busca aplicado aos dados de entrada presentes nas Seções 4.2 e 4.3.

Na Tabela 8 anterior foi possível observar que a mudança de cada conjunto de vizinhos distribuídos pelo índice de busca acaba determinando uma solução diferente. É possível observar também que não necessariamente o aumento do índice de busca é sinônimo de resultados melhores. No entanto, quanto maior o índice, maior são o número de possibilidades analisadas e, por sua vez, é possível que a busca gulosa encontre a resposta ótima neste espaço ampliado. Utilizando um índice de busca igual a 7, por exemplo, o algoritmo encontrou a menor média possível.

A partir de $z = 4$, o algoritmo já encontra uma resposta com índice $Score/P_r$ melhor do que as soluções presentes na ordem 1 e ordem 3. Portanto é considerado válido o uso da busca gulosa se forem utilizados níveis de busca. Para que os resultados destes algoritmos não fiquem restritos a apenas um exemplo, será determinado outros dados de entrada para a mesma matriz dos exemplos anteriores.

Os dados de entrada e resultados deste segundo exemplo podem ser encontrados no Apêndice L.

Tabela 8 – Resultados técnica de busca gulosa com índice de busca z

Índices	Menor custo de movimento	Menor $Score/P_r$	Método Padrão (sem IA)	Busca Gulosa com $z = 1$	Busca Gulosa com $z = 2$
Ordem	1	2	3	4	5
Armazém Final	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 0, 2] [2, 3, 3] [0, 0, 0]	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [3, 3, 0] [0, 0, 0]
Custo Movimento	19,6	20,1	21,6	22,3	22,9
pos_garra	[2,2]	[2,2]	[2,2]	[1,0]	[2,2]
status	TTPPPT	TTPPPOT	PPPTTT	OPTOTOTPP	OPTOTOPPT
$Score/P_r$	5,39	2,83	3,61	5,39	5,39
G	1,15	1,01	1,13	1,27	1,29
Índices	Busca Gulosa com $z = 3$	Busca Gulosa com $z = 4$	Busca Gulosa com $z = 5$	Busca Gulosa com $z = 6$	Busca Gulosa com $z = 7$
Ordem	6	7	8	9	10
Armazém Final	[1, 2, 2] [3, 3, 0] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]	[1, 0, 2] [2, 3, 3] [0, 0, 0]	[1, 2, 2] [0, 3, 0] [3, 0, 0]
Custo Movimento	22,3	22,4	22,4	21,6	20,1
pos_garra	[2,2]	[2,2]	[2,2]	[2,2]	[2,2]
status	OPTOTOTPP	OPTOTPPOT	OPTOTPPOT	PPPTTT	TTPPPOT
$Score/P_r$	5,39	2,83	2,83	3,61	2,83
G	1,27	1,11	1,11	1,13	1,01

Fonte: Do autor (2019)

A seguir são relatadas as conclusões principais do trabalho, referências bibliográficas e todos os apêndices que auxiliam a demonstração de algoritmos e resultados da dissertação.

5 CONCLUSÃO

Neste trabalho foram realizadas comparações entre diferentes técnicas de busca de inteligência artificial aplicadas a um sistema de armazenagem e recuperação de peças conhecido como ASRS. É importante destacar que este trabalho foi realizado analisando estudos de caso, inserindo alguns parâmetros e dinâmicas específicas na construção de um armazém, não um modelo geral para qualquer tipo de armazém.

Para a realização do deslocamento de peças, foi utilizado o algoritmo de Terry Winograd que se mostrou eficiente produzindo a dinâmica necessária do armazém, principalmente quando existe conflito de peças que ocupam o mesmo endereço. Este algoritmo foi adaptado colocando alguns parâmetros adicionais, como custo de movimento.

As técnicas de busca aplicadas ao sistema ASRS são efetivas, pois o sistema possui um número finito de possibilidades, basta realizar digitalmente a produção de vizinhos que correspondem aos movimentos do armazém e utilizar as técnicas de IA apresentadas para realizar a busca da melhor sequência de movimentos.

A criação dos índices de custo de movimento e de organização do armazém foram essenciais para criar parâmetros e referências para a análise dos resultados, possibilitando obter uma visão mais completa do sistema, facilitando também a análise dos resultados obtidos em cada uma das abordagens realizadas. Estes índices produzem um equilíbrio entre a organização e o custo do armazém, visto que nem sempre é possível obter os dois melhores resultados.

A criação do elemento *status* no vetor de dados de cada vizinho propicia observar o caminho realizado até a solução final. Ou seja, é possível observar a sequência de operações realizadas pelo armazém em cada uma das possibilidades representadas pelos vizinhos do algoritmo.

Com a aplicação de IA no problema foi observado que a operação de organização do armazém se torna viável dependendo do estado do armazém e a posição da garra. Durante o deslocamento de uma garra, é possível movimentar peças que não estão entre as peças de inserção ou lista de peças de saída, mas que seu movimento dentro do espaço do armazém se torna viável contribuindo para uma melhor organização do sistema.

Foi observado que a técnica de busca gulosa com critério de melhor média entre os índices de custo de movimento e organização foi efetiva, encontrando soluções que obtém valores de custo de movimento baixo e custo de organização. Obviamente esta técnica não encontra a solução ótima, mas possui valores melhores do que as médias produzidas por um armazém sem inteligência artificial, ou seja, realizando uma sequência fixa de operação.

A técnica de busca gulosa foi realizada em duas etapas principais deste trabalho: na busca por um melhor armazém de referência e na busca pela melhor média entre os vizinhos durante as operações realizadas pela garra no armazém. Na busca pela melhor média dos índices, foi observado que não é viável realizar uma busca instantânea logo na primeira produção de vizinhos, pois é necessário criar um campo de visão maior do caminho total ao invés de tomar uma decisão precipitada escolhendo uma solução entre as primeiras possibilidades. Desta forma, a utilização da busca gulosa com índice de busca produziu bons resultados a partir de índices de busca próximos à quarta iteração. Índices maiores produzem um campo de vizinhos maiores que levam a soluções próximas à melhor solução do sistema.

5.1 TRABALHOS FUTUROS

a) Implementar outros algoritmos de inteligência artificial para ampliação de resultados e discussão da melhor técnica de busca em relação à sistemas de armazenagem e recuperação de peças;
b) Aplicar algumas técnicas utilizadas em controlador específico da indústria afim de obter resultados práticos de uma célula de manufatura levando em consideração outros parâmetros como o tempo na execução de cada uma das tarefas realizadas pela garra no armazém. Além disso, a integração de IA à *hardwares* e controladores industriais poderá produzir bons estudos relacionados à metodologia de sua aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDI. **Agenda Brasileira para a indústria 4.0**. Disponível em: <http://www.industria40.gov.br/>. Acesso em: 18 set 2018

ARTERO, Almir Olivette. **Inteligência Artificial:** Teoria e prática. São Paulo: Editora Livraria da física, 2009.

BARTODZIEJ, Christoph Jan. **The Concept Industry 4.0:** An Empirical Analysis of Technologies and Applications in Production Logistics. Wiesbaden, Germany: Springer Gabler, 2017.

DAHUA, Qi; GUOQUAN, Cheng; ZHUAN, Wang, **The Study of Optimal Goods Distribution of Automated Storage/Retrieval System**, 2009 International Conference on Information Management, Innovation Management and Industrial Engineering, Xi'an, 2009, pp. 461-465. doi: 10.1109/ICIII.2009.269

EUROBUILD. **Armazém Industrial**. Disponível em: <http://english.eurobuildcee.com/?page=news&id=23475>. Acesso em 25 set 2018

KHASASI, Farah Hanani Mohammad; YUSOF, Zulkhairi Mohd; ALIAS, Mohd; ADAM, Ismail; RASIN, Mohamad, **Development of an Automated Storage and Retrieval System in dynamic industrial environment**, 2015 International Conference on BioSignal Analysis, Processing and Systems (ICBAPS), Kuala Lumpur, 2015, pp. 57-60. doi: 10.1109/ICBAPS.2015.7292218

KHASASI, Farah Hanani Mohammad; YUSOF, Zulkhairi Mohd; ALIAS, Mohd; ADAM, Ismail; RASIN, Mohamad (2016). **Development of Automated Storage and Retrieval System (ASRS) for Flexible Manufacturing System (FMS)**. Journal of Engineering Technology 2231-8798. 4. 43-50.

LEE, H. Felix; CHUNG Eunyon. **A Set-based Scheduling Problem for Square Rack Automated Storage and Retrieval Systems**; 2006 IEEE International Conference on Service Operations and Logistics, and Informatics, Shanghai, 2006, pp. 393-398. doi: 10.1109/SOLI.2006.329004

MELLO, Rachel. **Mapa Estratégico da Indústria 2018-2022**. Revista da Confederação Nacional da Indústria. Indústria Brasileira. Ano 3. nº20, p. 8-17, fev., 2018.

MINGRU, Zeng; HULIANG, Zong; WEI, Han, **Optimization for the Storage Management and Job Scheduling Based on Expert System,** 2009 IITA International Conference on Services Science, Management and Engineering, Zhangjiajie, 2009, pp. 47-49. doi: 10.1109/SSME.2009.55

NOVIG, Peter; RUSSEL, Stuart. **Inteligência Artificial**. 3a ed. Elsevier, 2013. USA.

RICH, Elaine; KNIGHT, Kevin. **Artificial Intelligence.** 2 ed. McGraw-Hill Higher Education. 1990.

SERAFINI, Paolo; UKOVICH, Walter, **Operating an automated storage and retrieval system** [Proceedings] 1988 International Conference on Computer Integrated Manufacturing, Troy, NY, USA, 1988, pp. 29-34. doi: 10.1109/CIM.1988.5388

SILVEIRA, Paulo Rogério da.; SANTOS, Winderson E.. **Automação e controle discreto**. 9ª ed. São Paulo: Érica, 2009.

TOMPKINS, James A. et al. - **Facilities plaining**. 2ª ed. Nova Iorque: John Wiley & Sons, 1996. ISBN 978-0-471-00252-9

TRAPPEY, Amy J. C. et al. **A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems in Advanced Manufacturing**. November, 2016. doi: 10.1109/ACCESS.2016.2619360

UNARCORACK. **Mini-load**. Disponível em: <https://www.unarcorack.com/asrs-systems-details/mini-load-asrs-system/>. Acesso em 25 set 2018

VENTURELLI, Marcio. **A evolução da automação industrial no contexto da digitalização e indústria 4.0.** 10 dez 2018. Disponível em: <https://www.automacaoindustrial.info/a-evolucao-da-automacao-industrial-no-contexto-da-digitalizacao-e-industria-4-0/>. Acesso em: 02 fev. 2019.

WINSTON, Patrick H. **Artificial Intelligence**. 3a ed. Pearson, 1992. USA.

YU, Xi; XU, Fangqin (2012). **Data Modeling and Analysis Based on the Automated Storage and Retrieval System**. Proceedings of the 2012 2nd International Conference on Business Computing and Global Informatization, BCGIN 2012. 633-636. 10.1109/BCGIN.2012.170.

APÊNDICE A - CÓDIGO DA SOLUÇÃO A

```
####################INÍCIO DO PROGRAMA RETIRA PECAS #############
estado_armazem_asrs = [[1,0,2],[0,3,0],[3,0,2]]
tama=len(estado_armazem_asrs)
validasaida = 0
validatipo = 0
matrizvalida = copy.deepcopy(estado_armazem_asrs)
pesopecas = []
pesodesloca = 0.5

####################INÍCIO DO PROGRAMA INSERE PECAS #############
listaentradasX=[]
entradavazia=[]
nentradas=0
valida=0
custoX=0
listapecX=[]

contP=0
contT=0
contO=0

print("ESTADO ARMAZÉM")
print('-=' * 30)
for i in range(0, tama):
  for j in range(0, tama):
    print(f'[{estado_armazem_asrs[i][j]:^5}]', end='')
  print()

validagarra=0
while(validagarra==0):
  posclawX = int(input('Digite as coordenadas iniciais da posição da
garra(linha): '))
  posclawY = int(input('Digite as coordenadas iniciais da posição da
garra(coluna): '))
  if posclawY<0 or posclawY>=tama or posclawX<0 or posclawX>=tama:
    print("Coordenadas inválidas. Digite novamente!")
    validagarra=0
  else:
    validagarra=1

#------------------------ENTRADA DE PEÇAS-------------------------
e_disponiveis=estados_vazios(estado_armazem_asrs)

nentradas = int(input('Digite a quantidade de peças que entrarão no armazém: '))

nsaida = int(input('Digite a quantidade de peças que sairão: '))

while(valida==0):
  if len(e_disponiveis) + nsaida < nentradas:
    print("O número de peças é incompatível com o de espaços possíveis no
armazém. Digite um número menor")
   nentradas = int(input('Digite a quantidade de peças que entrarão no armazém:
'))
    valida=0
  else:
    valida=1

validaentradas=0
while(validaentradas==0):
  for i in range(0,nentradas):
    listaentradasX += [int(input('Digite o tipo de objeto para inserir: '))]

  if 0 in listaentradasX:
    validaentradas=0
```

```
    print("Não é permitido digitar o valor 0. Insira os dados novamente")
    listaentradasX=[]
  else:
    validaentradas=1

print("As peças escolhidas foram:", listaentradasX)

#-------------------------SAÍDA DE PEÇAS-------------------------
npec = tama**2 - len(estados_vazios(estado_armazem_asrs))
while(validasaida==0):
  if npec + nentradas < nsaida:
    print("O número de peças de saida é incompatível com o número de peças do
armazém. Digite um número menor")
    nsaida = int(input('Digite a quantidade de peças que sairão: '))
    validasaida=0
  else:
    validasaida=1

listavalida=copy.deepcopy(listaentradasX)
for i in range(0,nsaida):
  listapecX += [int(input('Digite o tipo de objeto que você quer descartar: '))]
  validatipo=0

  while(validatipo==0):
    contapeca=0
    for x in range(0, tama):
      for y in range(0, tama):
        if matrizvalida[x][y]==listapecX[i] and contapeca==0:
          matrizvalida[x][y]=0
          print(matrizvalida)
          print(listavalida)
          validatipo=1
          contapeca+=1
        else:
          if listapecX[i] in listavalida:
            listavalida.remove(listapecX[i])
            print(matrizvalida)
            print(listavalida)
            validatipo = 1
            contapeca += 1

    if contapeca==0 or listapecX[i]==0:
      print("A peça que você busca não está disponível na matriz!")
      listapecX[i] = int(input('Digite o tipo de objeto para descartar: '))
      validatipo=0

print("As peças escolhidas foram:", listapecX)

posicaogarra = [posclawX,posclawY]

listasV=[[[copy.deepcopy(estado_armazem_asrs), 0, copy.deepcopy(posicaogarra)],
[0, 0, 0],'',copy.deepcopy(listaentradasX), copy.deepcopy(listapecX)]]
matriz_referencia = encontra_ref(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))

listasF=[]

listasA=copy.deepcopy(listasV)

listafinal=[]

while(listasV!=[]):
  for k in range(0, len(listasV)):
    contP = copy.deepcopy(listasV[k][1][0])
    contO = copy.deepcopy(listasV[k][1][1])
    contT = copy.deepcopy(listasV[k][1][2])

    estado_armazem_asrs = copy.deepcopy(listasV[k][0][0])
```

```
    custoX = copy.deepcopy(listasV[k][0][1])
    posicaogarra=copy.deepcopy(listasV[k][0][2])
    status = copy.deepcopy(listasV[k][2])
    listaentradasX = copy.deepcopy(listasV[k][3])
    listapecX = copy.deepcopy(listasV[k][4])

    if len(listaentradasX) > 0:
      if contP < nentradas:
        listaentradasZ = copy.deepcopy(listaentradasX)
        VP = insere_melhor(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listaentradasZ), copy.deepcopy(custoX),copy.deepcopy(posicaogarra))
        contP = contP + 1
        listaentradasZ.pop(0)
        [indice_orgP, scoreP] = indice_score(copy.deepcopy(VP[0]),
copy.deepcopy(listapecX))
        VizP = [copy.deepcopy(VP), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status + 'P', copy.deepcopy(listaentradasZ),
copy.deepcopy(listapecX), copy.deepcopy(indice_orgP), copy.deepcopy(scoreP)]
        listasF += [VizP]
        contP = 0

    if contO == 0 and contT < nsaida:
      matriz_referencia = referencia_central(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))
      VO = busca_gulosa(copy.deepcopy(matriz_referencia),
copy.deepcopy(estado_armazem_asrs), copy.deepcopy(posicaogarra),
copy.deepcopy(custoX))
      contO = 1
      [indice_orgO, scoreO] = indice_score(copy.deepcopy(VO[0]),
copy.deepcopy(listapecX))
      VizO = [copy.deepcopy(VO), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status+'O',copy.deepcopy(listaentradasX),copy.deepcopy(list
apecX),copy.deepcopy(indice_orgO), copy.deepcopy(scoreO)]
      listasF += [VizO]
      contO = 0

    if contT < nsaida:
      listapecZ = copy.deepcopy(listapecX)
      VT = tira_peca(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecZ), copy.deepcopy(posicaogarra), copy.deepcopy(custoX))
      listapecZ.pop(0)
      contT = contT + 1
      contO = 0
      [indice_orgT, scoreT] = indice_score(copy.deepcopy(VT[0][0]),
copy.deepcopy(listapecX))
      VizT = [copy.deepcopy(VT[0]), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status+'T',copy.deepcopy(listaentradasX),copy.deepcopy(list
apecZ), copy.deepcopy(indice_orgT), copy.deepcopy(scoreT)]
      listasF += [VizT]
      contT = 0

    listasV += copy.deepcopy(listasF)
    listasV = remove_repetidos_final(copy.deepcopy(listasV))

    listasF = []

    if listasV[k][3] == [] and listasV[k][4] == []:
      listafinal += [copy.deepcopy(listasV[k])]

  for y in range(0, len(listasA)):
    listasV.remove(listasA[y])
  listasA = copy.deepcopy(listasV)

for i in range(0,len(listasV)):
  listasV[i][0][1] = round(listasV[i][0][1], 1)
  print("O vizinho ",i," é: ", listasV[i])

listafinal = remove_repetidos_final(copy.deepcopy(listafinal))
```

```
custofinais=[]
organizacaofinais=[]

for i in range(0,len(listafinal)):
  listafinal[i][0][1] = round(listafinal[i][0][1],1)
  custofinais +=[copy.deepcopy(listafinal[i][0][1])]
  organizacaofinais += [copy.deepcopy(listafinal[i][5])]

  print("O vizinho da lista FINAL ",i," é: ", listafinal[i])

menorcusto=min(custofinais)
menororganizacao=min(organizacaofinais)

menorpossivel=0
menororgpossivel=0
melhores_org=[]
custos_f_org=[]

for i in range(0,len(listafinal)):
  if listafinal[i][0][1] == menorcusto:
    menorpossivel = copy.deepcopy(listafinal[i])

  if listafinal[i][5] == menororganizacao:
    melhores_org += [copy.deepcopy(listafinal[i])]
    custos_f_org += [copy.deepcopy(listafinal[i][0][1])]

menorcusto_f_org=min(custos_f_org)

for i in range(0, len(melhores_org)):
  if melhores_org[i][0][1] == menorcusto_f_org:
    menororgpossivel = copy.deepcopy(melhores_org[i])

print("O armazém com método comum é: ", listafinal[0])
print("O menor movimento possível é: ", menorpossivel)
print("A melhor organização possível é: ", menororgpossivel)
```

APÊNDICE B – CÓDIGO DA FUNÇÃO PUTON E AUXILIARES

```
def puton(A, B, ciclo, produto, posgarra):
  #chama a função getspace, depois grasp, depois move, depois ungrasp para
determinado parâmetro.
  global customovimento
  global custogarra
  global posgarra2
  if ciclo == 0:
    custogarra=0
    customovimento=0
    posgarra2=copy.deepcopy(posgarra)
    A1 = A
    B1 = B
    getspace(B1, produto,posgarra)
    custogarra+= grasp(A1, produto, posgarra2)
    customovimento+= move(A1, B1, produto)
    [neighboor,posgarra] = ungrasp(B1, produto)
    costneighboor =customovimento + custogarra
  else:
    A2 = A
    B2 = B
    getspace(B2, produto,posgarra)
    custogarra = grasp(A2, produto,posgarra)
    customovimento = move(A2, B2, produto)
    [neighboor,posgarra2] = ungrasp(B2, produto)
    costneighboor = customovimento + custogarra

  return neighboor, costneighboor, posgarra

def getspace(Bx, produto,posgarra):
  # verificar se o espaço destino está livre
  x = Bx[0]
  y = Bx[1]

  if produto[x][y] == 0:
    variavel=0
      #print("O local destino está livre!")
  else:
    #print("É necessário encontrar um novo local para o bloco indesejado: ")
     getridof(Bx, produto, posgarra)

def getridof(Bx, produto,posgarra):
  # chamar a função puton com os parâmetros b_indesejado e local livre
  custos = cria_matriz(tama, tama, 99) #custo apenas da função
  ciclo = 1
  for i in range(0, tama):
    for j in range(0, tama):
      if produto[i][j] == 0:
        livrea = i
        livreb = j
        custos[i][j] = custo(Bx, [livrea, livreb])
  menor = custos[0][0]
  for i in range(0, tama):
    for j in range(0, tama):
      if custos[i][j] <= menor:
        menor = custos[i][j]
        im = i
        jm = j
        melhor = [im, jm]
  puton(Bx, melhor, ciclo, produto, posgarra)
```

```
def grasp(Ax, produto,posgarra):
  #print("FUNÇÃO GRASP")
  for i in range(0, tama):
    for j in range(0, tama):
        if produto[i][j] == Ax:
          b_inicio = Ax

  return custo(posgarra, Ax)*pesodesloca

def move(Ax, Bx, produto):
  # realizar a movimentação, ou seja, calcular um custo
  x = Ax[0]
  y = Ax[1]
  w = Bx[0]
  z = Bx[1]
  produto[w][z] = produto[x][y]
  produto[x][y] = 0
  return custo(Ax, Bx)

def ungrasp(Bx, produto):
  w = Bx[0]
  z = Bx[1]
  posgarra=[w, z]
  return produto, posgarra
```

APÊNDICE C – CÓDIGO DA FUNÇÃO *ENCONTRA_REF*

```
def encontra_ref(estado_armazem, listapecas):
  tipos = []
  coordenadastipos = []
  listafinal = []
  coordenadasfinal = []
  pesosfinal = []
  pesooriginal = []
  pesooriginalf = []

  [matrizP, listaP, listaDestino] = matrizpeso(tama)
  listaDestino = [[2,2],[2,1],[1,2],[2,0],[1,1],[0,2],[1,0],[0,1],[0,0]]

  for i in range(0, tama):
    for j in range(0, tama):
      if estado_armazem[i][j] != 0:
        tipos += [estado_armazem[i][j]]
        coordenadastipos += [[i, j]]
        pesooriginal += [matrizP[i][j]]

  # reorganiza os vetores
  for i in range(0, len(listapecas)):
    for j in range(0, len(tipos)):
      if listapecas[i] == tipos[j]:
         listafinal += [listapecas[i]]
         coordenadasfinal += [coordenadastipos[j]]
         pesooriginalf += [pesooriginal[j]]
         pesosfinal += [listaP[i]]
         tipos.remove(listapecas[i])
         coordenadastipos.pop(j)
         pesooriginal.pop(j)
         break

  listafinal += tipos
  coordenadasfinal += coordenadastipos
  pesooriginalf += pesooriginal
  estado_armazem = cria_matriz(tama, tama, 0)

  for i in range(0, len(listafinal)):
    if i < nsaida:
      aux = copy.deepcopy(listaDestino[i])
      estado_armazem[aux[0]][aux[1]] = listafinal[i]
    else:
      aux = copy.deepcopy(coordenadasfinal[i])
      if pesooriginalf[i] >= listaP[i] and estado_armazem[aux[0]][aux[1]] == 0:
        aux = copy.deepcopy(coordenadasfinal[i])
        estado_armazem[aux[0]][aux[1]] = listafinal[i]
      else:
        proxvazia = proxima_vazia(estado_armazem)
        aux = copy.deepcopy(proxvazia)
        estado_armazem[aux[0]][aux[1]] = listafinal[i]
  print("O estado do armazém da matriz_ref é: ", estado_armazem)
  return estado_armazem
```

APÊNDICE D – CÓDIGO DA FUNÇÃO *TIRA_PECA*

```
def tira_peca(estado_armazem_aux, listapecas, pos_garra, custoanterior):

  vetor_pecas_saidas = []
  posicoesproibidas = []
  custosaida=0
  coordsaida=[]
  for k in listapecas:
    for i in range(0, tama):
      for j in range(0, tama):
        if estado_armazem_aux[i][j]==k and [i,j] not in posicoesproibidas:
          coordsaida = [i, j]
          custosaida = custo([i,j], [tama - 1, tama - 1]) + 1
    posicoesproibidas += [coordsaida]

    vetor_pecas_saidas += [[k, coordsaida, custosaida]]

  vizinhosExit = []
  for k in range(0, len(listapecas)):
    estado_armazem_aux = saipeca(copy.deepcopy(vetor_pecas_saidas[k][1]),
estado_armazem_aux)
    custoS = vetor_pecas_saidas[k][2] + custo(pos_garra,
vetor_pecas_saidas[k][1])*pesodesloca + custoanterior

    pos_garra = [tama - 1, tama - 1]
    vizinhosExit += [[copy.deepcopy(estado_armazem_aux), copy.deepcopy(custoS),
copy.deepcopy(pos_garra)]]

  return vizinhosExit
```

APÊNDICE E – CÓDIGO DA FUNÇÃO *BUSCA_GULOSA*

```
def busca_gulosa(matriz_ref, estado_armazem, posclaw, custoT):
  vizinhanca = [estado_armazem]
  pesovizinho = []
  solucao = 0
  custooperacao = [0]
  custototal = 0
  loops = 0
  posicoesgarras = [[0, 0]]

  while (solucao == 0):
    nvcorretos = verfica_sol(estado_armazem, matriz_ref)
    if nvcorretos == tama * tama:
      solucao = 1

    else:
      solucao = 0

    if solucao == 0:
      [Objetos, Posicoes, Possibilidades] = coordenadas(estado_armazem)
      loops += 1
      # Geração de vizinhos
      for i in range(0, len(Objetos)):
        for j in range(0, len(Posicoes)):
          if Objetos[i][:] != Posicoes[j][:]:
            parametrosaux = [puton(Objetos[i][:], Posicoes[j][:], 0,
copy.deepcopy(estado_armazem), posclaw)]
            vizinhanca += [parametrosaux[0][0]]
            custooperacao += [parametrosaux[0][1]]
            posicoesgarras += [parametrosaux[0][2]]
            [vizinhanca, custooperacao, posicoesgarras] =
remove_repetidos(vizinhanca, custooperacao, posicoesgarras)

      for i in range(len(pesovizinho), len(vizinhanca)):
        pesovizinho.append(0)

      for i in range(0, len(vizinhanca)):
        pesovizinho[i] = [verfica_sol(vizinhanca[:][:][i], matriz_ref)]

        if pesovizinho[i] == [tama * tama]:
          variavel=1
      # Encontra o vizinho ideal
      for i in range(0, len(vizinhanca)):
        if pesovizinho[i] >= max(pesovizinho):
          pesomelhor = pesovizinho[i]
          posmelhor = copy.deepcopy(i)

      estado_armazem = vizinhanca[:][:][posmelhor]
      custototal += custooperacao[posmelhor]
      posclaw = posicoesgarras[posmelhor]

  return [estado_armazem, custototal+custoT,posclaw]
```

APÊNDICE F – CÓDIGO DA FUNÇÃO *CUSTO_CM*

```
def custo_cm(matriz):
  tamanho = len(matriz)
  coordenada_cm = [round((tamanho - 1) / 2), round((tamanho - 1) / 2)]

  linha_cm = coordenada_cm[0]
  coluna_cm = coordenada_cm[1]

  # ANÁLISE DA LINHA
  somasuperiorL = 0
  for i in range(linha_cm - 1, -1, -1):
    for j in range(0, len(matriz[i])):
      somasuperiorL += matriz[i][j]
  #print("A soma superior de linha é: ", somasuperiorL)

  somainferiorL = 0
  for i in range(linha_cm + 1, tama):
    for j in range(0, len(matriz[i])):
      somainferiorL += matriz[i][j]
  #print("A soma inferior de linha é: ", somainferiorL)

  somalinhas = abs(somainferiorL - somasuperiorL)
  #print("O somatório das linhas são: ", somalinhas)

  # ANÁLISE DA COLUNA
  somasuperiorC = 0
  for i in range(coluna_cm - 1, -1, -1):
    for j in range(0, len(matriz[i])):
      somasuperiorC += matriz[j][i]
  #print("A soma superior de coluna é: ", somasuperiorC)

  somainferiorC = 0
  for i in range(coluna_cm + 1, tama):
    for j in range(0, len(matriz[i])):
      somainferiorC += matriz[j][i]
  #print("A soma inferior de coluna é: ", somainferiorC)

  somacolunas = abs(somainferiorC - somasuperiorC)
  #print("O somatório das colunas são: ", somacolunas)

  customassa = (somacolunas ** 2 + somalinhas ** 2) ** 0.5
  #print("O custo de massa é igual a: ", customassa)

  return customassa
```

APÊNDICE G – CÓDIGO DA FUNÇÃO *REFERENCIA_CENTRAL*

```
def referencia_central(estado_armazem, listapec):
  novo_armazem = copy.deepcopy(estado_armazem)
  matriz_ref = encontra_ref(copy.deepcopy(novo_armazem),copy.deepcopy(listapec))
  vetor_armazem = []
  for i in range(0,tama):
    vetor_armazem += estado_armazem[i]
  vetor_coordenadas=[]
  for i in range(0, tama):
    for j in range(0, tama):
      vetor_coordenadas += [[i,j]]
  possiveis_referencias = []
  possiveis_armazens=[]
  vetor_masscost = []
  while(len(vetor_armazem)!=0):
    for i in range(0, tama):
      for j in range(0, tama):
        copiapeca = copy.deepcopy(estado_armazem[i][j])
        copiacoordenada = copy.deepcopy([i,j])
        novo_armazem[i][j] = copy.deepcopy(vetor_armazem[0])

        novo_armazem[vetor_coordenadas[0][0]][vetor_coordenadas[0][1]] =
copiapeca

        if novo_armazem[i][j] == matriz_ref[i][j]:
          masscost = round(custo_cm(copy.deepcopy(novo_armazem))-0.1,2)
        else:
          masscost = round(custo_cm(copy.deepcopy(novo_armazem)), 2)
        vetor_masscost += [copy.deepcopy(masscost)]
        possiveis_armazens += [copy.deepcopy(novo_armazem)]
        novo_armazem = copy.deepcopy(estado_armazem)
    vetor_armazem.pop(0)
    vetor_coordenadas.pop(0)

  vetorX=[]
  vetor_desconto=[]
  vetor_masscost1 = []
  vetor_masscost2 = []
  indice_desconto=0
  listaDestino = [[2,2], [2,1], [1,2], [2,0], [1,1], [0,2], [1,0], [0,1], [0,0]]
  dlist = copy.deepcopy(listaDestino)
  for i in list(possiveis_armazens):
    if i not in vetorX:
      vetorX += [i]
      for z in range(0, len(listapec)):
        zx = dlist[z][0]
        zy = dlist[z][1]
        if listapec[z] == i[zx][zy]:
          indice_desconto+=1
      vetor_desconto+=[copy.deepcopy(indice_desconto)]
      masscost1 = round(custo_cm(copy.deepcopy(i)), 2)
      masscost2 = round(custo_cm(copy.deepcopy(i)) -0.1*indice_desconto, 2)
      vetor_masscost1 += [copy.deepcopy(masscost1)]
      vetor_masscost2 += [copy.deepcopy(masscost2)]
      possiveis_referencias += [[copy.deepcopy(i), [copy.deepcopy(masscost1)],
[copy.deepcopy(masscost2)], [copy.deepcopy(indice_desconto)]]]
      indice_desconto = 0

  menormasscost = min(vetor_masscost2)
  melhor_referencia = 0
  for i in range(0,len(possiveis_referencias)):
    if possiveis_referencias[i][2] == [menormasscost]:
      melhor_referencia = possiveis_referencias[i]
  return melhor_referencia[0]
```

APÊNDICE H – CÓDIGO DA SOLUÇÃO B

```
estado_armazem_asrs = [[1,0,2],[0,3,0],[3,0,2]]
tama=len(estado_armazem_asrs)
validasaida = 0
validatipo = 0
matrizvalida = copy.deepcopy(estado_armazem_asrs)
pesopecas = []
pesodesloca = 0.5

##############INÍCIO DO PROGRAMA INSERE PECAS ##############
listaentradasX=[]
entradavazia=[]
nentradas=0
valida=0
custoX=0
listapecX=[]

contP=0
contT=0
contO=0

###Entrada e saída de peças exatamente igual ao apêndice A

posicaogarra = [posclawX,posclawY]
ref_cost=(len(listapecX)+len(listaentradasX))*2
ref_org=tama**2 - 1

listasV=[[[copy.deepcopy(estado_armazem_asrs), 0, copy.deepcopy(posicaogarra)],
[0, 0, 0],'',copy.deepcopy(listaentradasX), copy.deepcopy(listapecX)]]
matriz_referencia = encontra_ref(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))

listasF=[]

listasA=copy.deepcopy(listasV)

listafinal=[]

while(listasV!=[]):
  for k in range(0, len(listasV)):
    contP = copy.deepcopy(listasV[k][1][0])
    contO = copy.deepcopy(listasV[k][1][1])
    contT = copy.deepcopy(listasV[k][1][2])

    estado_armazem_asrs = copy.deepcopy(listasV[k][0][0])
    custoX = copy.deepcopy(listasV[k][0][1])
    posicaogarra=copy.deepcopy(listasV[k][0][2])
    status = copy.deepcopy(listasV[k][2])
    listaentradasX = copy.deepcopy(listasV[k][3])
    listapecX = copy.deepcopy(listasV[k][4])

    if len(listaentradasX) > 0:
      if contP < nentradas:
        listaentradasZ = copy.deepcopy(listaentradasX)
        VP = insere_melhor(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listaentradasZ), copy.deepcopy(custoX),copy.deepcopy(posicaogarra))
        contP = contP + 1
        listaentradasZ.pop(0)
        [indice_orgP, scoreP] = indice_score(copy.deepcopy(VP[0]),
copy.deepcopy(listapecX))
        media_custoP = copy.deepcopy(VP[1])/ copy.deepcopy(ref_cost)
        media_orgP = copy.deepcopy(indice_orgP)/copy.deepcopy(ref_org)
        media_geralP = round((media_custoP + media_orgP)/2,2)
        VizP = [copy.deepcopy(VP), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status + 'P', copy.deepcopy(listaentradasZ),
```

```
copy.deepcopy(listapecX), copy.deepcopy(indice_orgP), copy.deepcopy(scoreP),
copy.deepcopy(media_geralP)]
        listasF += [VizP]
        contP = 0

    if contO == 0 and contT < nsaida:
      matriz_referencia = referencia_central(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))
      VO = busca_gulosa(copy.deepcopy(matriz_referencia),
copy.deepcopy(estado_armazem_asrs), copy.deepcopy(posicaogarra),
copy.deepcopy(custoX))
      contO = 1
      [indice_orgO, scoreO] = indice_score(copy.deepcopy(VO[0]),
copy.deepcopy(listapecX))
      media_custoO = copy.deepcopy(VO[1]) / copy.deepcopy(ref_cost)
      media_orgO = copy.deepcopy(indice_orgO) / copy.deepcopy(ref_org)
      media_geralO = round((media_custoO + media_orgO) / 2,2)
      VizO = [copy.deepcopy(VO), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status+'O',copy.deepcopy(listaentradasX),copy.deepcopy(list
apecX),copy.deepcopy(indice_orgO), copy.deepcopy(scoreO),
copy.deepcopy(media_geralO)]
      listasF += [VizO]
      contO = 0

    if contT < nsaida:
      listapecZ = copy.deepcopy(listapecX)
      VT = tira_peca(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecZ), copy.deepcopy(posicaogarra), copy.deepcopy(custoX))
      listapecZ.pop(0)
      contT = contT + 1
      contO = 0
      [indice_orgT, scoreT] = indice_score(copy.deepcopy(VT[0][0]),
copy.deepcopy(listapecX))
      media_custoT = copy.deepcopy(VT[0][1]) / copy.deepcopy(ref_cost)
      media_orgT = copy.deepcopy(indice_orgT) / copy.deepcopy(ref_org)
      media_geralT = round((media_custoT + media_orgT) / 2, 2)
      VizT = [copy.deepcopy(VT[0]), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)],status+'T',copy.deepcopy(listaentradasX),copy.deepcopy(list
apecZ), copy.deepcopy(indice_orgT), copy.deepcopy(scoreT),
copy.deepcopy(media_geralT)]
      listasF += [VizT]
      contT = 0

    listasV += copy.deepcopy(listasF)
    listasV = remove_repetidos_final(copy.deepcopy(listasV))
    listasF = []

    if listasV[k][3] == [] and listasV[k][4] == []:
      listafinal += [copy.deepcopy(listasV[k])]

  for y in range(0, len(listasA)):
    listasV.remove(listasA[y])
  listasA = copy.deepcopy(listasV)

for i in range(0,len(listasV)):
  listasV[i][0][1] = round(listasV[i][0][1], 1)

listafinal = remove_repetidos_final(copy.deepcopy(listafinal))
custofinais=[]
organizacaofinais=[]
mediasfinais=[]

for i in range(0,len(listafinal)):
  listafinal[i][0][1] = round(listafinal[i][0][1],1)
  custofinais +=[copy.deepcopy(listafinal[i][0][1])]
  organizacaofinais += [copy.deepcopy(listafinal[i][5])]
  mediasfinais +=[copy.deepcopy(listafinal[i][7])]
  print("O vizinho da lista FINAL ",i," é: ", listafinal[i])
```

```
menorcusto=min(custofinais)
menororganizacao=min(organizacaofinais)
menormedia=min(mediasfinais)

menorpossivel=0
menororgpossivel=0
melhores_org=[]
custos_f_org=[]

for i in range(0,len(listafinal)):
  if listafinal[i][0][1] == menorcusto:
    menorpossivel = copy.deepcopy(listafinal[i])

  if listafinal[i][5] == menororganizacao:
    melhores_org += [copy.deepcopy(listafinal[i])]
    custos_f_org += [copy.deepcopy(listafinal[i][0][1])]

  if listafinal[i][7] == menormedia:
    menormediapossivel = copy.deepcopy(listafinal[i])

menorcusto_f_org=min(custos_f_org)

for i in range(0, len(melhores_org)):
  if melhores_org[i][0][1] == menorcusto_f_org:
    menororgpossivel = copy.deepcopy(melhores_org[i])

print("O armazém com método comum é: ", listafinal[0])
print("O menor movimento possível é: ", menorpossivel)
print("A melhor organização possível é: ", menororgpossivel)
print("A melhor média possível é: ", menormediapossivel)
```

APÊNDICE I – CÓDIGO DA SOLUÇÃO C

```
#DECLARAÇÃO DE VARIÁVEIS IMPORTANTES NO PROCESSO
posicaogarra = [posclawX,posclawY]
ref_cost=(len(listapecX)+len(listaentradasX))*2
ref_org=tama**2 - 1

listasV=[[copy.deepcopy(estado_armazem_asrs), 0, copy.deepcopy(posicaogarra)],
[0, 0, 0],'',copy.deepcopy(listaentradasX), copy.deepcopy(listapecX)]
matriz_referencia = encontra_ref(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))

listasF=[]

listasA=copy.deepcopy(listasV)

listafinal=[]

k=0
print(listasV[3])
print(listasV[4])
while(listasV[3] != [] or listasV[4] != []):
  contP = copy.deepcopy(listasV[1][0])
  contO = copy.deepcopy(listasV[1][1])
  contT = copy.deepcopy(listasV[1][2])

  estado_armazem_asrs = copy.deepcopy(listasV[0][0])
  custoX = copy.deepcopy(listasV[0][1])
  posicaogarra = copy.deepcopy(listasV[0][2])
  status = copy.deepcopy(listasV[2])
  listaentradasX = copy.deepcopy(listasV[3])
  listapecX = copy.deepcopy(listasV[4])

  listasV=[]

  if len(listaentradasX) > 0:
    if contP < nentradas:
      listaentradasZ = copy.deepcopy(listaentradasX)
      VP = insere_melhor(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listaentradasZ), copy.deepcopy(custoX),
copy.deepcopy(posicaogarra))
      contP = contP + 1
      listaentradasZ.pop(0)
      [indice_orgP, scoreP] = indice_score(copy.deepcopy(VP[0]),
copy.deepcopy(listapecX))
      media_custoP = copy.deepcopy(VP[1]) / copy.deepcopy(ref_cost)
      media_orgP = copy.deepcopy(indice_orgP) / copy.deepcopy(ref_org)
      media_geralP = round((media_custoP + media_orgP) / 2, 2)
      VizP = [copy.deepcopy(VP), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)], status + 'P', copy.deepcopy(listaentradasZ),
copy.deepcopy(listapecX), copy.deepcopy(indice_orgP), copy.deepcopy(scoreP),
copy.deepcopy(media_geralP)]
      listasF += [VizP]
      contP = 0

  if contO == 0 and contT < nsaida:
    matriz_referencia = referencia_central(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))
    VO = busca_gulosa(copy.deepcopy(matriz_referencia),
copy.deepcopy(estado_armazem_asrs), copy.deepcopy(posicaogarra),
copy.deepcopy(custoX))
    contO = 1
    [indice_orgO, scoreO] = indice_score(copy.deepcopy(VO[0]),
copy.deepcopy(listapecX))
    media_custoO = copy.deepcopy(VO[1]) / copy.deepcopy(ref_cost)
    media_orgO = copy.deepcopy(indice_orgO) / copy.deepcopy(ref_org)
```

```
    media_geralO = round((media_custoO + media_orgO) / 2, 2)
    VizO = [copy.deepcopy(VO), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)], status + 'O', copy.deepcopy(listaentradasX),
copy.deepcopy(listapecX), copy.deepcopy(indice_orgO), copy.deepcopy(scoreO),
copy.deepcopy(media_geralO)]
    listasF += [VizO]
    contO = 0

  if contT < nsaida:
    listapecZ = copy.deepcopy(listapecX)
    VT = tira_peca(copy.deepcopy(estado_armazem_asrs), copy.deepcopy(listapecZ),
copy.deepcopy(posicaogarra), copy.deepcopy(custoX))
    listapecZ.pop(0)
    contT = contT + 1
    contO = 0
    [indice_orgT, scoreT] = indice_score(copy.deepcopy(VT[0][0]),
copy.deepcopy(listapecX))
    media_custoT = copy.deepcopy(VT[0][1]) / copy.deepcopy(ref_cost)
    media_orgT = copy.deepcopy(indice_orgT) / copy.deepcopy(ref_org)
    media_geralT = round((media_custoT + media_orgT) / 2, 2)
    VizT = [copy.deepcopy(VT[0]), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)], status + 'T', copy.deepcopy(listaentradasX),
copy.deepcopy(listapecZ), copy.deepcopy(indice_orgT), copy.deepcopy(scoreT),
copy.deepcopy(media_geralT)]
    listasF += [VizT]
    contT = 0

  listasV += copy.deepcopy(listasF)
  listafinal = copy.deepcopy(listasV)
  custofinais = []
  organizacaofinais = []
  mediasfinais = []

  for i in range(0, len(listafinal)):
    listafinal[i][0][1] = round(listafinal[i][0][1], 1)
    custofinais += [copy.deepcopy(listafinal[i][0][1])]
    organizacaofinais += [copy.deepcopy(listafinal[i][5])]
    mediasfinais += [copy.deepcopy(listafinal[i][7])]

  menorcusto = min(custofinais)
  menororganizacao = min(organizacaofinais)
  menormedia = min(mediasfinais)

  menorpossivel = 0
  menororgpossivel = 0
  melhores_org = []
  custos_f_org = []

  for i in range(0, len(listafinal)):
    if listafinal[i][7] == menormedia:
      menormediapossivel = copy.deepcopy(listafinal[i])

  print("A melhor média possível é: ", menormediapossivel)

  listasV = copy.deepcopy(menormediapossivel)
  listasF = []
```

APÊNDICE J – CÓDIGO DA SOLUÇÃO D

```
#DECLARAÇÃO DE VARIÁVEIS IMPORTANTES NO PROCESSO
posicaogarra = [posclawX,posclawY]
ref_cost=(len(listapecX)+len(listaentradasX))*2
ref_org=tama**2 - 1

listasV=[[[copy.deepcopy(estado_armazem_asrs), 0, copy.deepcopy(posicaogarra)],
[0, 0, 0],'',copy.deepcopy(listaentradasX), copy.deepcopy(listapecX)]]
matriz_referencia = encontra_ref(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecX))
listasF=[]
listasA=copy.deepcopy(listasV)
listafinal=[]
listafinal_pt=[]
tamanhoV=0
listaAux=[]
solucao = 0
indicez=indicez-1

while(solucao != 1):
  for z in range(0, indicez+1):
    for k in range(0, len(listasV)):
      tamanhoV = copy.deepcopy(len(listasV))
      contP = copy.deepcopy(listasV[k][1][0])
      contO = copy.deepcopy(listasV[k][1][1])
      contT = copy.deepcopy(listasV[k][1][2])
      estado_armazem_asrs = copy.deepcopy(listasV[k][0][0])
      custoX = copy.deepcopy(listasV[k][0][1])
      posicaogarra = copy.deepcopy(listasV[k][0][2])
      status = copy.deepcopy(listasV[k][2])
      listaentradasX = copy.deepcopy(listasV[k][3])
      listapecX = copy.deepcopy(listasV[k][4])

      if len(listaentradasX) > 0:
        if contP < nentradas:
          listaentradasZ = copy.deepcopy(listaentradasX)
          VP = insere_melhor(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listaentradasZ), copy.deepcopy(custoX),
copy.deepcopy(posicaogarra))
          contP = contP + 1
          listaentradasZ.pop(0)
          [indice_orgP, scoreP] = indice_score(copy.deepcopy(VP[0]),
copy.deepcopy(listapecX))
          media_custoP = copy.deepcopy(VP[1]) / copy.deepcopy(ref_cost)
          media_orgP = copy.deepcopy(indice_orgP) / copy.deepcopy(ref_org)
          media_geralP = round((media_custoP + media_orgP) / 2, 2)
          VizP = [copy.deepcopy(VP), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)], status + 'P', copy.deepcopy(listaentradasZ),
copy.deepcopy(listapecX), copy.deepcopy(indice_orgP), copy.deepcopy(scoreP),
copy.deepcopy(media_geralP)]
          listasF += [VizP]
          if VizP[3] == [] and VizP[4] == []:
            listafinal_pt += copy.deepcopy([VizP])
          VizP =[]
          contP = 0

      if contO == 0 and contT < nsaida:
        matriz_referencia =
referencia_central(copy.deepcopy(estado_armazem_asrs), copy.deepcopy(listapecX))
        VO = busca_gulosa(copy.deepcopy(matriz_referencia),
copy.deepcopy(estado_armazem_asrs), copy.deepcopy(posicaogarra),
copy.deepcopy(custoX))
        contO = 1
        [indice_orgO, scoreO] = indice_score(copy.deepcopy(VO[0]),
copy.deepcopy(listapecX))
```

```
        media_custoO = copy.deepcopy(VO[1]) / copy.deepcopy(ref_cost)
        media_orgO = copy.deepcopy(indice_orgO) / copy.deepcopy(ref_org)
        media_geralO = round((media_custoO + media_orgO) / 2, 2)
        VizO = [copy.deepcopy(VO), [copy.deepcopy(contP), copy.deepcopy(contO),
copy.deepcopy(contT)], status + 'O', copy.deepcopy(listaentradasX),
copy.deepcopy(listapecX), copy.deepcopy(indice_orgO), copy.deepcopy(scoreO),
copy.deepcopy(media_geralO)]
        listasF += [VizO]
        if VizO[3] == [] and VizO[4] == []:
          listafinal_pt += copy.deepcopy([VizO])
        VizO = []
        contO = 0

      if contT < nsaida:
        listapecZ = copy.deepcopy(listapecX)
        VT = tira_peca(copy.deepcopy(estado_armazem_asrs),
copy.deepcopy(listapecZ), copy.deepcopy(posicaogarra), copy.deepcopy(custoX))
        listapecZ.pop(0)
        contT = contT + 1
        contO = 0
        [indice_orgT, scoreT] = indice_score(copy.deepcopy(VT[0][0]),
copy.deepcopy(listapecX))
        media_custoT = copy.deepcopy(VT[0][1]) / copy.deepcopy(ref_cost)
        media_orgT = copy.deepcopy(indice_orgT) / copy.deepcopy(ref_org)
        media_geralT = round((media_custoT + media_orgT) / 2, 2)
        VizT = [copy.deepcopy(VT[0]), [copy.deepcopy(contP),
copy.deepcopy(contO), copy.deepcopy(contT)], status + 'T',
copy.deepcopy(listaentradasX), copy.deepcopy(listapecZ),
copy.deepcopy(indice_orgT), copy.deepcopy(scoreT), copy.deepcopy(media_geralT)]
        listasF += [VizT]
        if VizT[3] == [] and VizT[4] == []:
          listafinal_pt += copy.deepcopy([VizT])
        VizT = []
        contT = 0

      for y in range(0, len(listasA)):
        listasV.remove(listasA[y])
      listasA = []
      listaAux += copy.deepcopy(listasF)
      listasF = []

    if listaAux!=[]:
      listasV = copy.deepcopy(listaAux)
    if z == indicez:
      listafinal += copy.deepcopy(listasV) + copy.deepcopy(listafinal_pt)
    listaAux = []
    listafinal = remove_repetidos_final(copy.deepcopy(listafinal))

  if len(listafinal) > 0:
    mediasfinais = []
    for i in range(0, len(listafinal)):
      listafinal[i][0][1] = round(listafinal[i][0][1], 1)
      mediasfinais += [copy.deepcopy(listafinal[i][7])]
      print("O vizinho da lista FINAL ", i, " é: ", listafinal[i])
    menormedia = min(mediasfinais)
    for i in range(0, len(listafinal)):
      if listafinal[i][7] == menormedia:
        menormediapossivel = copy.deepcopy(listafinal[i])
    print("A melhor média possível é: ", menormediapossivel)
    listasV = [copy.deepcopy(menormediapossivel)]
    if listasV[0][3] == [] and listasV[0][4]==[]:
      solucao = 1
    listafinal = []
    listasA=copy.deepcopy(listasV)
```

APÊNDICE L – RESULTADOS DA BUSCA GULOSA COM ÍNDICE DE BUSCA

pos_garra	[1,1]	ListaE	[2,2,2]
Armazém	[1,0,2], [0,3,0], [3,0,2]	ListaS	[1,2,3]

Índices	Menor custo de movimento	Menor $Score/P_r$	Método Padrão (sem IA)	Menor Média Geral	Busca Gulosa com $z = 2$
Ordem	1	2	3	4	5
Armazém Final	[2, 2, 2] [2, 3, 0] [0, 0, 0]	[0, 2, 2] [3, 0, 0] [2, 0, 2]	[0, 2, 2] [2, 3, 2] [0, 0, 0]	[2, 2, 2] [2, 0, 0] [0, 3, 0]	[0, 2, 2] [2, 3, 2] [0, 0, 0]
Custo Movimento	18,5	28,1	22,1	21,1	21,9
pos_garra	[2,2]	[2,2]	[2,2]	[2,2]	[2,2]
status	POTTPPT	TOPPPTOT	PPPTTT	POTPPOTOT	POPPTTT
$Score/P_r$	6,32	1,0	4,47	3,61	4,47
G	1,17	1,23	1,2	1,11	1,19
Índices	Busca Gulosa com $z = 3$	Busca Gulosa com $z = 4$	Busca Gulosa com $z = 5$	Busca Gulosa com $z = 6$	Busca Gulosa com $z = 7$
Ordem	6	7	8	9	10
Armazém Final	[2, 2, 2] [2, 3, 0] [0, 0, 0]	[0, 2, 2] [2, 3, 2] [0, 0, 0]	[2, 2, 0] [2, 3, 2] [0, 0, 0]	[2, 2, 2] [2, 3, 0] [0, 0, 0]	[2, 2, 0] [2, 3, 2] [0, 0, 0]
Custo Movimento	19,5	22,1	20,1	18,5	20,2
pos_garra	[0,0]	[2,2]	[2,2]	[2,2]	[2,2]
$status$	POPTTOTP	POPPTTT	POTTOPPT	TTPPPT	PTTOPPT
$Score/P_r$	6,32	4,47	4,47	6,32	4,47
G	1,21	1,2	1,12	1,17	1,12